Unified SYSTEM ON CHIP RESTAPI SERVICE (USOCRS)

University of Oulu
Information Processing Science
Master's Thesis
David Ochia
2023

Abstract

This thesis investigates the development of a Unified System on Chip RESTAPI Service (USOCRS) to enhance the efficiency and effectiveness of SOC verification reporting. The research aims to overcome the challenges associated with the transfer, utilization, and interpretation of SoC verification reports by creating a unified platform that integrates various tools and technologies.

The research methodology used in this study follows a design science approach. A thorough literature review was conducted to explore existing approaches and technologies related to SOC verification reporting, automation, data visualization, and API development. The review revealed gaps in the current state of the field, providing a basis for further investigation. Using the insights gained from the literature review, a system design and implementation plan were developed. This plan makes use of cutting-edge technologies such as FASTAPI, SQL and NoSQL databases, Azure Active Directory for authentication, and Cloud services. The Verification Toolbox was employed to validate SoC reports based on the organization's standards. The system went through manual testing, and user satisfaction was evaluated to ensure its functionality and usability.

The results of this study demonstrate the successful design and implementation of the USOCRS, offering SOC engineers a unified and secure platform for uploading, validating, storing, and retrieving verification reports. The USOCRS facilitates seamless communication between users and the API, granting easy access to vital information including successes, failures, and test coverage derived from submitted SoC verification reports. By automating and standardizing the SOC verification reporting process, the USOCRS eliminates manual and repetitive tasks usually done by developers, thereby enhancing productivity, and establishing a robust and reliable framework for report storage and retrieval. Through the integration of diverse tools and technologies, the USOCRS presents a comprehensive solution that adheres to the required specifications of the SOC schema used within the organization.

Furthermore, the USOCRS significantly improves the efficiency and effectiveness of SOC verification reporting. It facilitates the submission process, reduces latency through optimized data storage, and enables meaningful extraction and analysis of report data.

Keywords
Systems on Chip, SOC Verification Reports, Representational State Transfer, Application Programming Interface, Cloud Technologies, Integration

Supervisor
Dr., University Lecturer Elina Annanperä

Abbreviations

API - Application Programming Interface
AAD - Azure Active Directory
CRUD - Create, Read, Update, Delete
CI - Continuous Integration
CD - Continuous Delivery
DBMS - Database Management System
DSR - Design Science Research
DSM - Design Science Methodology
EDA - Electronic Design Automation
FEDS - Framework for Evaluation in Design Science Research
HATEOAS - Hypermedia as the Engine of Application State
JIT - Just-in-Time
JSON - JavaScript Object Notation
JWT - JSON Web Tokens
MVCC - Multi-Version Concurrency Control
ORM - Object-Relational Mapping
REST - Representational State Transfer
SDN - Software-Defined Networking
SOC - System on Chip
SQL - Structured Query Language
URI - Uniform Resource Identifier
URL - Uniform Resource Locator
UML - Unified Modeling Language
USOCRS - Unified System on Chip RESTAPI Service
VIP - Verification IP
VCS - Version Control System
WAL - Write-Ahead Logging

Foreword

I am happy to present this foreword for the USOCRS, which represents a significant milestone in my academic journey as a master's student at the University of Oulu. I would like to show my gratitude to the people who have played an important role in the successful completion.

To begin with, I would like to extend my sincere appreciation to my line manager, Mr. Matti Niemisto. From the very beginning, he has been a great source of support, providing valuable insights and the necessary information to ensure the completion of this project. His prompt response and belief in this endeavour have been instrumental in shaping its outcomes.

I would also like to express my deep appreciation to Dr. Elina Annanperä, my thesis supervisor. Despite the time constraints, she graciously took on the responsibility of supervising my work and invested considerable time and effort into guiding me through the remaining part of the writing, helping me meet the university's required standards. Her expertise and insightful feedback have greatly contributed to the refinement and improvement of this paper.

I would also like to express my gratitude to Mr. Mika Mäntylä, for his valuable contributions and feedback which also help to complete this paper. His expertise and experience in the field have provided valuable insights and greatly improved the overall quality and effectiveness of this paper. Additionally, I want to acknowledge and appreciate Mr. Markus Kelanti, my thesis inspector.

I hope that the USOCRS presented in this paper will have a meaningful impact and serve as a valuable resource for future research. I am confident that the insights gained from this project will pave the way for further advancements in this domain.

Once again, I express my deepest gratitude to everyone involved in making this research endeavour a reality. Without their support, expertise, and encouragement, the successful completion of this thesis would not have been possible.

David Ochia

Oulu, June 19, 2023

Contents

1. Introduction

In today's rapidly advancing world, the use of microprocessors, specifically Systems on Chip (SoC), has enabled portable and high-performance computing devices. The demand for SoC devices has grown significantly due to their ability to meet specific customer needs and the increasing number of design constraints, such as low power consumption, high performance, and small form factors (Ishtiaq, et al., 2021). The development and deployment of the Internet of Things (IoT) have also been facilitated by the compatibility and flexibility of SoC devices i.e., bringing various components and functionalities into a single chip, providing customized technologies for different user needs, especially for seniors and specific applications (Ishtiaq, et al., 2021). According to Moore's Law, the number of integrated circuits in manufacturers' systems doubles every 18 months, driving the research, development, and planning of semiconductor systems (Moore, 1965). As a result, thorough testing, and validation of SoC devices have become essential, leading to the generation of vast amounts of verification reports.

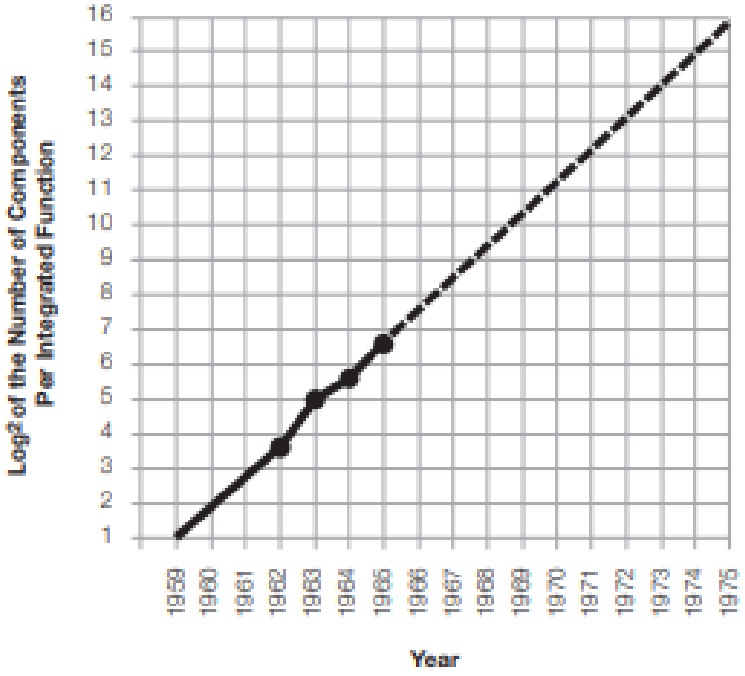

Figure 1. Moore's Law shows the increasing development of integrated circuits in the years (Moore, 1965)

The **"log2 of the number of components per integrated function"** is a way to express the exponential growth in transistor density and integration capabilities predicted by Moore's Law in a concise and quantifiable manner (Moore, 1965). SoC devices are complex integrated circuits that combine various components and functionalities, including processors, memory, and communication interfaces, into a single chip (Chakravarthi, 2019). The development of SOC devices for products and services relies heavily on SOC verification reporting.

In the IP/SOC teams at Nokia, SoC engineers currently rely on a manual approach for generating reports, which involves copying data directly from the SoC verification report files and inputting them into Excel sheets. This manual process demands a significant

amount of effort and leads to a decrease in productivity. Traditionally, Nokia's SOC verification and progress reporting have heavily depended on Electronic Design Automation (EDA) vendor tools (Synopsys Inc., 2023).

In the SOC development teams, the responsibilities are typically divided among Intellectual Property (IP) teams, each handling the design, integration, and verification of their respective sub-areas. The verification process for SOC IP development is time-consuming, involving the successful execution of hundreds, if not thousands, of test cases to ensure the quality and verification metrics of the IP. Two main metrics in SOC verification are functional coverage and code coverage. To monitor functional coverage, the EDA tools maintain a mapping between the features and test cases outlined in the test plan. This mapping allows the tools to produce results indicating the number of test cases that have been successfully passed and the level of feature verification achieved. Additionally, EDA tools track code coverage metrics such as line coverage, toggle coverage, branch coverage, statement coverage, block coverage, expression coverage, focused expression coverage, and Finite-State Machine coverage. These coverage results are stored by the EDA tools in proprietary data formats specific to the vendors. EDA began as a captive capability, with large OEMs employing software engineers to automate the design, implementation, and verification of chips (Synopsys Inc., 2023).

However, despite the availability of EDA tools, the reporting of SOC verification results and progress remains primarily a manual effort undertaken by SoC engineers. This manual reporting process leads to repetitive work and hampers productivity within the team. The engineers are required to extract the relevant information from the EDA tools and compile it into reports manually. Consequently, the manual approach to reporting not only consumes valuable time and effort but also limits the ability to visualize and analyze the verification progress efficiently. The reliance on a manual reporting process highlights the need for a more streamlined and automated approach to SOC verification reporting.

By implementing a unified SOC reporting system, the organization aims to address these challenges and improve the efficiency and effectiveness of verification report transfers, utilization, and interpretation. The RESTAPI service will enable seamless data access and facilitate the integration of various tools and technologies available within and beyond the organization. This innovative solution will abstract the data access layer between the SoC engineers and the users, providing a standardized and efficient method for managing verification reports.

1.1 Problem statement:

The manual effort required for SOC verification reporting presents challenges in terms of productivity, scalability, and data analysis. Reliance on EDA tools with vendor-proprietary data formats hinders the automation of result collection and limits advanced data visualization and analytics. Furthermore, the lack of a unified result format and storage complicates the reporting process. This lack of a unified SOC reporting system creates difficulties for companies like Nokia, which rely on SOC devices for their products and services.

1.2 The research question hence can be stated as follows:

> *How can we design, develop, and validate a RESTAPI service using available tools and technologies to enhance the efficiency and effectiveness of SoC validation reporting?*

1.3 Proposed Solution

The proposed solution is the development of a Unified System on Chip RESTAPI Service (USOCRS) for SOC Verification Reports using various tools and technologies available both within the organization and beyond. This system will automate the result collection of SoC verification reports and enable advanced data visualization and analytics with the API. A unified results format and storage will be used to simplify the reporting process and adhere to the organization's specified standards. This system will improve productivity, scalability, and data analysis or interpretation of the report data.

1.4 Scope

The proposed USOCRS aims to address the existing gaps in SOC verification report management and analysis. By utilizing various tools and technologies, both internal and external to the organization, the USOCRS intends to automate the collection, interpretation, and visualization of SOC verification reports.

The USOCRS aims to bridge the gap in SOC verification report management by offering an automated, standardized, and scalable solution. By leveraging the power of REST API, modern technologies, and validation mechanisms, the USOCRS empowers users to streamline the verification process, improve productivity, and make informed decisions based on comprehensive and reliable SOC verification report data.

1.4.1 Steps followed in the development process of the USOCRS:

- Requirements Analysis: A thorough analysis of the requirements and challenges faced by SoC engineers in managing and reporting verification reports is conducted. This entails a detailed examination of existing workflows, identification of pain points, and determination of the necessary functionalities for the RESTAPI service.
- Design and Implementation: Once the requirements are established, the design and implementation phase commences. This involves the creation of REST API services, including the design of user interfaces, database structures, and integration with existing systems. Design principles and best practices from the literature are integrated to ensure an efficient and effective design.
- Evaluation: The developed artifact is subjected to rigorous evaluation to assess its performance, usability, efficiency, accuracy, and user satisfaction. Evaluation metrics, as defined in the FEDS framework, are employed to assess these aspects. The evaluation results provide valuable insights for further improvements and validate the effectiveness of the developed RESTAPI service.
- Documentation and Communication: Documentation plays a vital role in the research methodology, as it ensures the proper dissemination of research knowledge and findings. Technical reports, conference papers, and presentations are prepared to communicate the research process, design decisions, evaluation

outcomes, and the developed artifact. This enables the sharing of knowledge and encourages further discussion and collaboration within the academic and industrial communities.

The research methodology employed in the development of the USOCRS project combines the principles of Design Science Research (DSR). By adopting a systematic and rigorous approach, the USOCRS project aims to create a practical and effective RESTAPI service to address the challenges faced by SoC engineers in managing and reporting verification reports. The methodology encompasses various stages, including requirements analysis, design and implementation, evaluation, and documentation, ensuring the development of a viable artifact and contributDesign Science Research steps followed in this work.

1.5 Paper structure

The chapters of this paper provide a comprehensive overview of the proposed USOCRS. The project's subject is introduced in Chapter 1, along with the goals, research questions, and overall purpose of the study in addition to the tools and technologies that will be used.

Chapter 2 will focus on the comprehensive literature review of related work and their reporting. Following the literature review includes a review of various tools employed in the development processes of the USOCRS. This chapter will also offer an overview of internal tools such as the Verification Toolbox and the Verification Report Schemas that will be utilized for verification report validation.

Chapter three will introduce the design science research methodology which is used in this work including design steps followed to achieve the desired results. Design Science Methodology (DSM) is a research approach that involves the development of an artifact or a solution to address a specific problem. It is a problem-solving methodology that seeks to design and create innovative solutions to practical problems in a specific domain.

The design and architecture of the suggested system will be the main topics of chapter four. This will include overall system requirements such as functional and non-functional requirements. It also includes the use cases for various scenarios the for the system, engineers, and users. The system's implementation will be covered in detail in Chapter Five, along with the use of various technologies such as Fastapi as a web framework for creating the REST API, data models, algorithms, testing, and deployment of the system.

The evaluation of the system, including its performance and results, will be presented in chapter six. Chapter seven will cover the Discussion of the results and the future work of the system such as gaps and features which might be integrated into the system for more functionality and usability. Conclusions will be covered in Chapter Eight based on a summary of the project's primary results and the study goals and questions.

2. Literature Review

This chapter provides background information on literature related to this work, related work in the field of SOC verification reporting and the use of RESTAPI services in SoC development was look into. The literature review aims to explore existing research, methodologies, and technologies used in the development and implementation of SOC verification reporting systems or similar. By looking into the previous studies by other authors, we can identify gaps in the current literature and highlight the significance and novelty of the proposed USOCRS. In addition to that, we will discuss various technologies and concepts which have been used in the development of this application and similar systems.

2.1 IP/SoC Verification

The growing complexity of SoC designs due to market demand has resulted in an increased need for the SoC verification process. However, the investment in automating the verification process has been relatively limited compared to other design tools. The EDA industry has primarily focused on refining high-profile design tools like synthesis, place & route, extraction, and analysis, while the automation of verification remains largely a manual process usually done by designers or engineers (Wilson, 2010). However, further research and development are necessary to bridge the gap between manual debugging and fully automated debug tools. This advancement would ultimately lead to a reduction in overall verification time and enhance the productivity of SoC design.

He, Guo, Zhao, & Jin (2020) conducted an extensive literature review on the topic of formal verification for System-on-Chip (SoC) security. Their objective was to provide a comprehensive overview of the current state-of-the-art formal verification techniques specifically applied to SoC security. The authors employed a systematic approach by searching multiple databases, including IEEE Xplore, ACM Digital Library, and ScienceDirect, using relevant keywords such as "SoC security," "formal verification," and "hardware security" to gather relevant research studies. They revealed several formal verification techniques that are commonly employed in the context of SoC security. These techniques include model checking, theorem proving, and SAT/SMT-based techniques. He et al. (2020) discussed the unique advantages and limitations associated with each method and provided real-world examples to illustrate their applications in SoC security. For instance, they highlighted the ability of model checking to exhaustively analyze all possible system states, thus verifying the security properties of SoC designs. They also emphasized how theorem proving enables formal proof of correctness and security properties through rigorous mathematical reasoning. Additionally, the authors elucidated how SAT/SMT-based techniques leverage Boolean satisfiability solving and constraint solving to assess and verify the security aspects of SoCs (He, Guo, Zhao, & Jin, 2020).

Throughout their review, He et al. (2020) critically evaluated the existing research and addressed the strengths and weaknesses inherent in different formal verification techniques. They also identified several challenges encountered when applying formal verification to SoC security, such as scalability concerns with large-scale designs, the need for automated verification methodologies, and the integration of formal techniques with other verification approaches (He, Guo, Zhao, & Jin, 2020).

Ray, Peeters, Tehranipoor, & Bhunia (2017) emphasize that security assurance in SoC devices encompasses a wide range of factors due to the inherent complexity of modern computing devices. Identifying security objectives proves challenging as it requires consideration of design features, architectural parameters, security requirements of the operating system, applications, and user expectations. However, given the time-to-market constraints, security assurance activities should focus on architecture and validation components that are not already covered by other activities. This necessitates a comprehensive understanding of various designs, architectures, and validation flows by security architects and validators to identify potential gaps that could undermine the system's security objectives. The authors underscore the importance of validating deterministic security requirements, which are validation objectives directly derived from security policies (Ray, Peeters, Tehranipoor, & Bhunia, 2017).

Ray, Peeters, Tehranipoor, & Bhunia (2017) argue that these requirements must be validated to ensure the overall security and privacy of the system. Due to time-to-market constraints, the security validation organization should prioritize targets that are not covered by other validation activities, thereby avoiding resource duplication. This approach places the responsibility on the security validation organization to gain a holistic understanding of the entire spectrum of SoC design validation and identify specific gaps in security (Ray, Peeters, Tehranipoor, & Bhunia, 2017).

(Deshpande Anil, 2008) focuses on the role of Verification IP (VIP) cores in SOC verification. The paper emphasizes the importance of automated test generation, simulation, and analysis techniques in enhancing the verification process. It provides valuable insights into the use of VIP cores for verifying SOC interconnects and their contribution to overall design correctness and reliability (Deshpande Anil, 2008).

(Sivakumar, 2020) presents a blog post discussing the comparison between IP and SOC verification. It offers a comparative analysis of the differences and challenges between IP and SOC verification. The post highlights the complexities associated with SOC verification, particularly in verifying IP blocks and the interconnects between them.

IP/SoC verification is an important part of computing that provides successful design and implementation of SoC devices. The importance of IP/SoC verification can be summarized as follows:

- Optimized verification and examination flow: For IP core-based SoC designs, a simplified verification and examination of the device in development contribute significantly to the success of the product (Deshpande Anil, 2008).
- Covered test cases: Because complicated SoCs use pre-verified and stable IP, SoC verification engineers typically prefer targeted test cases to verify how the entire system behaves and collect metrics of the results (Sivakumar, 2020).
- Handle the complexity of modern computing devices: Given the complexity of modern computing devices and their continued demand for additional functionality, both IP and SoC verification processes are important today (Wilson, 2010).
- Access to industry logs: Synopsys VC Verification IP (VIP) gives verification engineers access to the industry's latest protocols, interfaces, and storage needed for verification (Synopsys, 2023).

By examining the reviewed papers, several relevant themes and findings emerge that can inform the proposed solution. Firstly, the papers discuss the challenges associated with

SOC verification, such as the manual effort required for reporting, limitations in automation, and the lack of unified result formats and storage (He, Guo, Zhao, & Jin, 2020). These challenges align with the problem statement of the proposed development of USOCRS, which aims to address the issues of productivity, scalability, and data analysis in SOC verification reporting. The reviewed papers also highlight the importance of automation and advanced data visualization in SOC verification. They emphasize the need to automate test generation, simulation, and analysis processes to ensure correctness, reliability, and efficiency. This aligns with the proposed solution of developing a USOCRS that automates the result collection of SOC verification reports and enables advanced data visualization and analytics. By incorporating these features, the proposed solution aims to overcome the manual effort required for reporting and provide advanced data analysis capabilities.

Furthermore, the papers discuss the use of verification IP (VIP) and formal verification techniques in SOC design and security assurance (Ray, Peeters, Tehranipoor, & Bhunia, 2017). While the specific methodologies used in the reviewed papers are not mentioned, the insights gained from them can inform the development of the USOCRS. The proposed solution can leverage VIP and formal verification approaches to enhance the validation and interpretation of SOC verification reports, thereby improving the effectiveness and reliability of the USOCRS.

2.2 Electronic Design Automation (EDA) Tools

Electronic Design Automation (EDA) includes a different number of software, hardware, and services that are used to support different stages of semiconductor device development, including definition, planning, design, implementation, verification, and manufacturing (MacMillen, Butts, Camposano, Hill, & Williams, 2000). EDA tools play a very important part in helping the design and validation of semiconductor manufacturing processes, ensuring that designs meet manufacturing requirements from customers, and promoting the reuse of existing design components. These tools operate through three primary functions: simulation, design, and verification (MacMillen, Butts, Camposano, Hill, & Williams, 2000).

Simulation tools enable engineers to predict the behavior of a proposed circuit before its actual implementation. By providing insights into circuit performance, these tools help identify potential issues and refine designs before committing to manufacturing. Design tools, on the other hand, facilitate the assembly of circuit elements to realize a desired circuit function based on a given description. They streamline the design process by automating various tasks and reducing manual effort. Verification tools evaluate either the logical or physical representation of a chip to ensure correct connectivity and performance, providing confidence in the final design (MacMillen, Butts, Camposano, Hill, & Williams, 2000).

As the ASIC (Application-Specific Integrated Circuit) industry emerged, the demand for comprehensive tools to automate chip simulation, design, and verification increased. This led to the evolution of point-tool companies into broad-line suppliers offering a diverse range of software and hardware products. Synopsys, a prominent player in the EDA market, stands as the industry's leading provider of EDA technology. Synopsys offers an extensive portfolio of solutions catering to the design and verification of advanced chips. Moreover, the company provides top-notch products to support the development of secure, high-quality, and compliant software projects, particularly targeting web

application development with a focus on the model-view-controller design pattern (Synopsys Inc., 2023).

Coenrad (2020) paper provides a valuable overview of Electronic Design Automation (EDA) tools for superconducting circuits. (Coenrad, 2020) author underscores the crucial role of Electronic Design Automation (EDA) tools in facilitating the design and development processes of superconducting circuits. EDA tools offer diverse functionalities such as circuit simulation, layout design, and verification, enabling designers to analyze and optimize circuit performance (Coenrad, 2020). Coenrad (2020) emphasizes the need for specialized EDA tools capable of accommodating the unique characteristics of superconducting circuits, including the presence of Josephson junctions and the requirement for cryogenic environments. The paper critically evaluates the currently available EDA tools designed for superconducting circuits, discussing their capabilities, limitations, and areas that require improvement. Coenrad (2020) identifies several commercial and open-source tools commonly utilized in the design and analysis of superconducting circuits, such as SPICE-based simulators, layout editors, and electromagnetic simulators. The author highlights the challenges associated with these tools, including their limited support for superconducting-specific modeling and the absence of comprehensive cryogenic analysis features (Coenrad, 2020).

2.3 Automation and Web Services

Eito-Brun & Amescua-Seco (2018) explore the automation of quality reports in the aerospace industry. They examine the overall challenges faced in generating good and timely quality reports and propose an automated system to deal with these challenges. They highlight the importance of data integration, real-time monitoring, and intelligent analysis in improving the efficiency and effectiveness of quality reports. Eito-Brun & Amescua-Seco (2018) present a comprehensive framework that utilizes automated data collection, analysis, and visualization techniques to enhance the quality reporting process. This study contributes to the aerospace industry by providing a practical solution for automating quality reports, thereby reducing manual effort, and improving decision-making processes (Eito-Brun & Amescua-Seco, 2018).

The research methodology used by Eito-Brun & Amescua-Seco (2018) is empirical research where the authors conducted a case study in the aerospace industry to investigate the automation of quality reports. They collected data from multiple sources, including interviews, observations, and documentation, to analyze the current practices and propose an automated solution. They found that automation significantly improves the efficiency and accuracy of generating quality reports, reducing manual effort, and minimizing errors. Eito-Brun & Amescua-Seco (2018) explore the automation of quality reports in the aerospace industry. It discusses the challenges faced in manual reporting processes and highlights the benefits of automation. The insights from this paper can inform the automation aspects of the USOCRS, enabling the automatic collection of SOC verification reports and improving the productivity of the users.

In their paper, Srikant Kumar Mohanty (2015) addresses the verification challenges related to System-on-Chip (SOC) interconnects. The authors highlight the importance of test bench automation in improving the verification process. The paper introduces a methodology that incorporates automated test generation, simulation, and analysis techniques to ensure the correctness and reliability of SOC interconnects. The study emphasizes the significance of automation in overcoming the complexity and time-consuming nature of SOC verification. By proposing an automated approach, this

research contributes to the field of SOC design and verification by offering a practical solution to enhance the efficiency and accuracy of the verification process. The paper focuses on test bench automation for the verification challenges of SOC Interconnect (Srikant Kumar Mohanty, 2015).

Srikant Kumar Mohanty (2015) employed experimental research methodology for their study. The authors designed and implemented a test bench automation framework and performed experiments to evaluate its effectiveness in addressing the verification challenge. The results showed that the automation approach effectively enhances the verification process, improving reliability and reducing the time and effort required for testing. They discuss techniques and methodologies for automating the verification process. The approaches they presented in their study can be applied to automate the verification process within the USOCRS, enhancing the efficiency and effectiveness of SOC validation report transfers.

Petcu et al. (2011) investigate the design and development of an interoperability Application Programming Interface (API) for sky computing. The paper discusses the challenges associated with heterogeneous computing environments and proposes an API that facilitates interoperability and resource management in distributed systems. The authors present a comprehensive architecture that incorporates standardization and abstraction techniques to enable seamless communication and resource sharing among different platforms. Their research contributes to the area of cloud computing and distributed systems by providing a framework for building interoperable APIs, thereby enhancing the integration and efficiency of sky computing environments (Petcu, ciun, Neagul, Lazcanotegui, & Rak, 2011).

The research methodology they followed in their study is design science research. The author proposes a conceptual model and then develops and evaluates the API prototype based on the model's principles and requirements (Petcu, ciun, Neagul, Lazcanotegui, & Rak, 2011). The research emphasizes the importance of interoperability in cloud computing environments and proposes an API design to facilitate seamless communication between different cloud platforms. The study by Petcu et al. (2011) addresses the importance of building an interoperability API for sky computing. It discusses the design and implementation of an API to facilitate communication between different computing systems. The concepts and considerations discussed in this study can be leveraged to design and develop the API component of the USOCRS, enabling seamless data transfer and interoperability between various tools and technologies such as integration with client applications, mobile or other desktop apps. (Petcu et al., 2011)

Zhou, Li, Luo, & Chou (2014) focus on the design patterns for REST Application Programming Interfaces (APIs) in the area of Software-Defined Networking (SDN). The authors highlight the importance of designing REST APIs that are scalable, flexible, and capable of handling diverse network management tasks within the workflow. The paper presents a set of design patterns that facilitate the development of SDN Northbound APIs, enabling effective communication between the control plane and the application layer. By proposing these design patterns, the research contributes to the field of SDN by providing guidelines for building robust and interoperable APIs, thereby promoting the adoption and deployment of SDN solutions (Zhou, Li, Luo, & Chou, 2014).

The research methodology they utilized in their study is conceptual. The authors analyze existing REST API design patterns for SDN Northbound API by examining related literature, industry practices, and standards. They propose a set of design patterns based on their analysis and provide guidelines for designing SDN Northbound APIs,

contributing to the standardization and interoperability of software-defined networking solutions. Zhou et al. (2014) explore different design patterns and best practices for designing RESTful APIs. The insights from their study can guide the design and implementation of the RESTAPI component within the USOCRS, ensuring adherence to standardized and effective API design principles which are generally accepted and optimized for performance. (Zhou et al., 2014)

Kaur, Kaur, Kapoor, & Singh, (2021) focus on the design and development of a customizable web API for interoperability of antimicrobial resistance data. The authors highlight the challenges faced in integrating and exchanging antimicrobial resistance data and propose a web API that facilitates data interoperability and sharing among different stakeholders. The paper presents a comprehensive architecture that incorporates data standardization, modular design, and semantic interoperability to ensure efficient data exchange and utilization. This research contributes to the field of healthcare and data interoperability by offering a practical solution for integrating and sharing antimicrobial resistance data, thereby enabling effective decision-making and research in the domain (Kaur, Kaur, Kapoor, & Singh, 2021).

The research methodology employed in their work is design science research. The authors design and develop a customizable web API for the interoperability of antimicrobial resistance data. They describe the development process, including requirements analysis, system design, implementation, and testing (Kaur, Kaur, Kapoor, & Singh, 2021). The proposed API enables seamless data exchange and integration across different systems, facilitating collaboration and analysis in the field of antimicrobial resistance research. The concepts and approaches presented in this paper can be applied to design and develop a customizable and interoperable RESTAPI within the USOCRS, enabling seamless integration and exchange of SOC verification reports within the SOC/IP teams.

Bakar, Ismail, Idris, & Shukur (2015) explore the design of the seMeja API based on the Create, Read, Update, Delete, and Navigational (CRUD+N) concept. The authors discuss the importance of designing APIs that provide comprehensive functionality for data management and navigation. The paper presents the seMeja API, which incorporates the CRUD+N concept to facilitate data manipulation, querying, and navigation in web applications. The study contributes to the field of API design by providing a practical approach to building robust and user-friendly APIs that support CRUD operations along with navigational capabilities.

The research methodology used in this paper is design science research. The authors propose the design of the seMeja API based on the CRUD+N concept. They present the design principles and discuss the features and functionalities of the API (Bakar, Ismail, Idris, & Shukur, 2015). The seMeja API offers an efficient and flexible approach for data manipulation, supporting Create, Read, Update, Delete, and additional operations, enhancing the usability and functionality of the API. They discuss the design principles for creating APIs that support Create, Read, Update, Delete (CRUD) operations along with additional functionalities. The findings from this review can guide the design of the USOCRS API, incorporating CRUD operations and additional features required for efficient management and manipulation of SOC verification reports.

Im, Yim, & Kim (2012) propose a web service for automated Intellectual Property (IP) and System-on-Chip (SoC) verification. The authors highlight the challenges associated with IP/SoC verification and present a web service that utilizes distributed computing resources to improve the verification process. The paper discusses the architecture, implementation, and performance evaluation of the proposed web service. This research

contributes to the field of IP/SoC verification by offering a practical solution that harnesses the power of distributed computing to enhance the efficiency and scalability of the verification process. The paper focuses on a web service for automated IP/SoC verification. They employed the experimental research methodology in their study. The authors develop a web service platform and conduct experiments to evaluate its performance and effectiveness in IP/SoC verification. (Im et al., 2012)

The developed platform offers a scalable and distributed solution for IP/SoC verification, leveraging networked computers to improve efficiency and reduce the verification time (Im, Yim, & Kim, 2012). It highlights the benefits of leveraging networked resources for efficient verification processes. The insights from this review can inform the development of the USOCRS, enabling the utilization of networked resources to automate the collection and processing of SOC verification reports.

Zhu & Gao, (2014) propose a novel approach to generate properties for web service verification from a threat-driven model. The authors highlight the importance of ensuring the security and reliability of web services and introduce a methodology that systematically emanates verification properties from a threat-driven model. Their study presents a case study to demonstrate the effectiveness of the technique they have proposed. Zhu et al. (2014) research contributes to the field of web service verification by providing a systematic method for identifying and verifying security properties, thereby enhancing the trustworthiness and dependability of web services.

The research methodology used in this paper is conceptual research where the authors propose a novel approach to generating properties for web service verification from a threat-driven model. The approach is developed based on the analysis of related literature, threat models, and verification techniques (Zhu & Gao, 2014). The proposed approach enhances the security and reliability of web services by identifying potential threats and automatically generating properties for verification. The findings from this study by Zhu et al. (2014) can inform the design and implementation of security measures within the USOCRS, ensuring the integrity and confidentiality of SOC verification reports during transfer and storage. (Zhu et al., 2014)

(Matinolli, 2016) focuses on the design, implementation, and evaluation of a database for a software testing team. The author discusses the challenges faced by testing teams in managing and organizing testing-related data and proposes a database solution that supports efficient test case management, defect tracking, and reporting. The paper presents the design principles, database schema, and performance evaluation of the implemented database. This research contributes to the field of software testing by offering a practical solution for improving the efficiency and effectiveness of testing activities through effective data management.

This paper employs design science research methodology. The author designs, implements, and evaluates a database for a software testing team. The research involves identifying requirements, designing the database schema, implementing the system, and evaluating its performance and usability (Matinolli, 2016). The research highlights the importance of a well-designed database in supporting software testing activities and presents a practical solution for managing testing data effectively. It provides insights into database management and optimization techniques. The findings from this review can guide the development of the database component within the USOCRS, ensuring efficient storage and retrieval of SOC verification reports.

Méré, Jouault, Pallardy, & Perdriau (2022) provides feedback on the formal verification of UML models in an industrial context, specifically focusing on a smart device life cycle management system. Méré et al. (2022) discuss the challenges, benefits, and lessons learned from applying a formal verification technique to UML models. By applying these techniques to UML models, it becomes possible to identify and address potential design flaws or inconsistencies at an early stage. The paper presents a case study to demonstrate the practical application and impact of formal verification in the context of a real-world industrial system. This research contributes to the field of model-driven engineering by highlighting the importance of formal verification and providing insights into its application in an industrial setting. One of the key advantages of formal verification highlighted by Méré et al. (2022) is its ability to uncover design errors and inconsistencies that may not be easily detectable through informal methods. Formal verification provides a rigorous and systematic approach to analyzing the behavior and properties of UML models, ensuring that they adhere to specified requirements. (Méré et al., 2022)

The research involves applying formal verification techniques to a smart device life cycle management system and analyzing the results and lessons learned (Méré, Jouault, Pallardy, & Perdriau, 2022). The research demonstrates the applicability and benefits of formal verification techniques in verifying UML models, providing insights into improving the reliability and quality of smart device life cycle management systems. While the focus of this review is on smart device life cycle management systems, the findings can be relevant to the development of the USOCRS. By incorporating formal verification techniques into the design and implementation process, the USOCRS can enhance the reliability and accuracy of SOC verification reports.

Bansal & Ouda (2022) explored the combination of FastAPI and machine learning for continuous authentication based on behavioral biometrics. Their objective was to develop a system capable of continuously authenticating users using machine learning algorithms that analyze their behavioral biometrics. The study proposed a system that utilized FastAPI, a Python-based web framework, for data collection and processing, while employing machine learning algorithms, such as decision trees and support vector machines, for data analysis and user authentication. By gathering user data such as keystroke dynamics and mouse movements, the system could authenticate users based on their unique behavioral patterns. To evaluate the system's effectiveness, Bansal and Ouda collected a dataset from 50 participants and achieved an impressive 95% accuracy in user authentication.

This research has significant relevance to the proposed USOCRS. The utilization of FastAPI in the USOCRS can enable real-time data collection and processing of SoC verification reports, aligning to automate result collection and improve data analysis. Additionally, integrating machine learning algorithms, inspired by Bansal and Ouda's study, can enhance the interpretation of SoC validation reports within the USOCRS, allowing for advanced data visualization and analytics. Leveraging the insights and methodology from Bansal and Ouda's research, the USOCRS can benefit from the scalability, efficiency, and accuracy provided by FastAPI and machine learning in data processing and user authentication (Bansal & Ouda, 2022).

By incorporating the reviewed literature, we can further examine the pros and cons, identify gaps, and highlight their relevance to the proposed USOCRS for SOC Verification Reports.

- He et al. (2020) emphasizes the advantages of formal verification techniques in ensuring the security of SoCs. Formal verification provides rigorous analysis of

security properties and improves system reliability. The USOCRS can benefit from incorporating formal verification techniques to enhance the security aspects of SOC Verification Reports. (He et al., 2020)

- (Matinolli, 2016) highlights the importance of database management in software testing. Different database management systems, such as MySQL, can improve read performance and maintainability. Integrating a robust database management system within the USOCRS can enhance the storage and retrieval of SOC Verification Reports.
- Méré et al. (2022) provide insights into the challenges and benefits of formal verification in an industrial context. Practical feedback and industry-oriented approaches are crucial in the design of the USOCRS to ensure its effectiveness and applicability in real-world SOC device testing scenarios. (Méré et al., 2022)
- Srikant Kumar Mohanty et al. (2015) focus on the advantages of automating the verification process, which aligns with the automation goals of the USOCRS. Improved verification efficiency, reduced effort, and enhanced reliability can be achieved by incorporating automated test bench systems within the USOCRS. (Srikant Kumar Mohanty et al., 2015)
- (Eito-Brun & Amescua-Seco, 2018) shed light on the benefits and implications of automating quality reporting processes. The USOCRS can leverage automation techniques to improve efficiency, data accuracy, and timely decision-making in the generation of SOC Verification Reports.
- Yeon-Ho Im et al. (2012) propose a web service for automated IP/SoC verification. While the paper lacks a comprehensive discussion of challenges and gaps, the idea of automation presented aligns with the objectives of the USOCRS in terms of improving productivity and scalability. (Yeon-Ho Im et al., 2012)

2.3.1 Areas of Agreement

Automation

Several papers e.g., Eito-Brun & Amescua-Seco (2018), and Srikant Kumar Mohanty (2015)) highlight the benefits of automation in improving efficiency, accuracy, and reliability in different domains, such as quality reporting and SOC verification.

API Design and Interoperability

The papers by Petcu, ciun, Neagul, Lazcanotegui, & Rak (2011), Zhou, Li, Luo, & Chou (2014), Kaur, Kaur, Kapoor, & Singh (2021), Bakar, Ismail, Idris, & Shukur (2015), and Zhu & Gao (2014) emphasize the importance of API design and interoperability in their respective contexts, recognizing the need for standardized and flexible solutions to facilitate communication and data exchange between systems.

Verification

The papers by Im, Yim, & Kim (2012), Zhu & Gao (2014), and Méré, Jouault, Pallardy, & Perdriau (2022), highlight the significance of verification techniques in different domains, including IP/SoC verification, web service verification, and formal verification of UML models. They emphasize the need for reliable verification processes to ensure system correctness and security.

2.3.2 Potential Areas of Disagreement

While the works of literature above do not directly contradict each other, there may be differing perspectives or approaches in the following areas:

Design Patterns

The paper by Zhou et al. focuses on REST API design patterns for SDN Northbound API, while other papers may adopt different design approaches or patterns for their specific API implementations. These variations in design choices could lead to differences in opinions or approaches.

Contextual Differences

Each paper addresses a specific problem within a particular domain. Due to variations in the application domains, methodologies, and goals, there may be differences in how the authors approach and prioritize certain aspects of their research.

The reviewed papers cover topics such as quality reports in the aerospace industry, test bench automation, API design for interoperability, web service verification, and database design for software testing. By analyzing these papers, we have identified common trends, advancements, and research gaps in the field of automation. The reviewed literature highlights the significance of automation in enhancing efficiency, reliability, and interoperability in different domains, thereby providing valuable insights for future research and development efforts.

This literature reviews collectively contributes to the proposed development of the USOCRS by providing valuable insights and recommendations across various aspects. They offer guidance on API design, automation of verification processes, security considerations, database management, interoperability, and formal verification. By leveraging these insights, the USOCRS can address the challenges outlined in the problem statement and provide an efficient, scalable, and unified system for SOC verification report management.

2.4 Technologies

The USOCRS is developed using various tools and technologies that will be discussed in this chapter. The system is implemented in Python, which is the same programming language used by some of the SoC teams in the organization. The Python FASTAPI framework is used to build the system, providing a wide range of tools and features for web application development, including data hints and automated Swagger documentation (SmartBear Software, 2023). The FASTAPI framework includes modules like SQLAlchemy, enabling seamless integration with SQL databases such as Postgres. Uvicorn, which is recommended for FASTAPI, is used as the server to build standalone Python applications. FastAPI follows openapi standards like Swagger, making it simple to create and configure web application APIs.

To document the API, the USOCRS utilizes the Swagger framework. Swagger is an open-source framework that facilitates the design, development, and documentation of RESTful APIs. It offers a web-based interface that allows developers to explore and test APIs, enhancing their understanding of how to utilize the system's API. The system stores data in both MongoDB and Postgres databases. Postgres is an open-source relational

database management system known for its high performance, scalability, and reliability. It is extensively used in web applications and supports the SQL language, enabling easy integration with programming languages like Python using modules such as Pydantic and SQLAlchemy.

RESTful API

In a study conducted by Rauf, Vistbakk, & Troubitsyna (2018), they focused on verifying stateful services that utilize REST APIs. They emphasized the importance of designing stateful services that adhere to the principles of REST, such as statelessness and extensibility. To achieve this, they proposed a dependable system design approach that includes a behavioral interface for services. This interface ensures that service users invoke the service correctly, while service developers implement the required functionality. To validate service implementations and perform conformance testing of service compositions, the authors introduced a method that utilizes generated behavioral interface skeletons and design models. The services examined in their study included scenarios like booking, payment, confirmation, cancellation, refund, and deletion.

To construct the design models, Rauf et al. used a stepwise development approach in Event-B. They represented the resource model, which encompasses the service resources along with their data attributes, links, and properties. The behavioral model was represented using a UML class diagram and a UML protocol state machine with state invariants. This comprehensive approach aimed to analyze the consistency of service design models with REST constraints. The authors also discussed various methods available for specifying web service compositions, ranging from formal techniques to XML-based standards. Their proposed approach effectively addressed issues such as inconsistent design, model checking of service specifications, and the challenge of dealing with a large number of resources, known as the state explosion problem.

According to RedHat Inc (2020), REST is a simple interface for transferring information without relying on additional software layers. It is commonly used as a management API for performing CRUD (Create, Read, Update, Delete) operations on resources. REST is particularly suitable for straightforward CRUD APIs due to its high level of abstraction and ease of use. Gupta (2022) mentions that REST APIs are stateless and control access through local endpoints. They require HTTPS for secure communication, rate limits to control the number of API calls, authentication mechanisms to ensure authorized access, and mechanisms to access the business logic of the application. Different authentication schemes can be employed, including Basic Authentication, API Key, JSON Web Tokens (JWT), OAuth 2.0, Token-Based Authentication, Cookie-Based Authentication, and OpenID. These schemes help protect against unauthorized access and limit API abuse. HTTPS and cryptographic signatures provide an optimal level of protection for web services. Data validation plays a crucial role in ensuring that data conforms to the API specification, and Flask is a popular framework for developing API services. Swagger can be used to create a flexible unified API, while servers should be protected with firewalls and WSGI servers.

Now, let's delve into more detail about how REST API services can enhance the efficiency of USOCRS:

1. Automated report collection: REST API services introduce automation to the data collection process, resulting in time savings and a reduced risk of human error. By automating data collection, manual effort in SOC verification reporting is

eliminated, making the process more efficient and reliable (Zhou, Li, Luo, & Chou, 2014).

2. Simplified reporting process: REST API services simplify the reporting process by providing a unified format for results and a centralized storage system. The use of USOCRS streamlines the reporting process according to predefined standards, ensuring consistency and ease of access for stakeholders (Kaur, Kaur, Kapoor, & Singh, 2021) and (RedHat Inc., 2020).
3. Abstraction of data access: REST API services abstract the complexity of data access from stakeholders. Instead of directly accessing the SOC verification report file, users interact with the API endpoints, reducing the risk of human error and ensuring controlled and secure access to the data (Kaur, Kaur, Kapoor, & Singh, 2021).
4. Real-time reporting and monitoring: REST API services facilitate real-time reporting and monitoring capabilities. SOC teams can access the latest data and generate reports on demand, enabling proactive decision-making and faster incident response (Kaur, Kaur, Kapoor, & Singh, 2021).
5. Integration with other systems: REST API services enable seamless integration with other systems within the security infrastructure. This integration can include connecting with SIEM (Security Information and Event Management) systems, incident management platforms, and ticketing systems, enhancing the overall effectiveness and efficiency of the USOCRS (Rauf, Vistbakk, & Troubitsyna, 2018).

FASTAPI

FastAPI is a cutting-edge web framework specifically designed for building APIs utilizing Python programming and leveraging standard Python-type hints. It has gained significant relevance and reputation for its exceptional performance and efficiency in developing high-performance APIs. FastAPI aims to overcome the limitations of other Python web frameworks like Flask, Django, and Pyramid. It stands out as one of the fastest Python web frameworks available today, as demonstrated by benchmarks (Rajkotia, 2021). This speed and performance make it a good choice for the USOCR, where fast and efficient data processing is necessary to facilitate ease of verification reporting. One of the key advantages of FastAPI is its seamless integration of modern Python features such as type hints and async/await syntax. This allows developers to write clean, concise, and maintainable code while harnessing the power of asynchronous programming within their development. With the USOCR dealing with large amounts of complex data and operations, FastAPI's efficient code execution contributes to faster response times and improved overall system performance.

Additionally, FastAPI provides automatic generation of OpenAPI documentation and clients for your API. This means that developers can effortlessly document their APIs and automatically generate client code, saving time and effort. The USOCR can benefit from this feature by ensuring clear documentation of their APIs, promoting easier integration with external systems, and facilitating collaboration with developers that will be using the system. FastAPI is vast with an active and vibrant community, which contributes to its continuous improvement and extensive support. It integrates seamlessly with other prominent Python libraries like SQLAlchemy, Tortoise ORM, and Pydantic, further enhancing its versatility and utility (Bansal & Ouda, 2022).

The USOCR can leverage this active community to seek assistance, share experiences, and collaborate with other developers facing similar challenges. FastAPI's ease of setup and configuration, combined with its exceptional performance compared to popular

frameworks like Node.js and Go, make it an ideal choice for the USOCRS. FastAPI's speed and scalability make it well-suited to meet the USOCRS requirements in terms of data processing and overall system performance. Also, by adopting FastAPI, the USOCR can benefit from a powerful and efficient web framework that facilitates the development of high-performance APIs. Its compatibility with Python and its modern features make it an excellent choice for managing complex data operations and ensuring seamless integration with external systems. With FastAPI's outstanding performance and extensive community support, the USOCR can streamline its development process and deliver robust and efficient solutions for its data-intensive workflows.

SQL and NoSQL Databases

Databases are an important part of all modern software development which needs a means to store their data. They provide a means to store and retrieve data efficiently and consistently (de Oliveira VF, 2022). The two most popular types of databases currently in use in many applications are SQL and NoSQL databases. SQL databases are based on the relational model and use SQL (Structured Query Language) for querying and managing data. On the other hand, NoSQL databases are non-relational and use various data models such as documents, key-value, graphs, etc (Wisal Khan, 2022). SQL databases provide a fixed schema or fixed structure that defines the structure of the data and the relations between different entities in the database i.e., they are used for structuring data and mapping their relation to other data within the database. SQL databases are widely used in enterprise or robust applications where data consistency and reliability are very important. They provide ACID (Atomicity, Consistency, Isolation, and Durability) (Rouse, 2020) properties that ensure that queries are executed reliably. This is especially crucial for the USOCR, as accurate and up-to-date data is essential for various Olympic committee operations, including athlete selection, event planning, and performance analysis.

PostgreSQL is a robust open-source object-relational database management system that is continuously enhanced by a large community of developers to cater to the evolving needs of users (PostgreSQL, 2023). It offers a wide range of data types, operators, and functions, which contribute to its versatility and make it a preferred choice for databases. One of the key strengths of PostgreSQL is its support for ACID transactions, ensuring the reliability and consistency of data in the development of the USOCRS. Moreover, PostgreSQL excels in efficiently indexing and querying large datasets, making it highly suitable for handling substantial amounts of data (PostgreSQL, 2023).

PostgreSQL is extensively utilized in enterprise applications where data integrity and reliability are of importance. In terms of standards compliance, PostgreSQL adheres to a significant number of features mandated by the SQL:2016 Core conformance, with at least 170 out of 179 mandatory features supported. This encompasses diverse data types such as integers, numerics, strings, Booleans, structured data, date/time, arrays, ranges/multirange, UUIDs, documents, geometries, customizations, data integrity constraints, unique constraints, primary keys, foreign keys, exclusion constraints, explicit and advisory locks, concurrency control, performance indexing, advanced indexing techniques, query planning and optimization, index-only scans, multicolumn statistics, transactions, nested transactions via savepoints, Multi-Version Concurrency Control, parallelization of read queries and B-tree index building, table partitioning, all transaction isolation levels defined in the SQL standard (including Serializable), a just-in-time compilation of expressions, reliability features, disaster recovery mechanisms, write-ahead logging, and various replication modes such as asynchronous, synchronous, and

logical replication (PostgreSQL, 2023). This ensures that the USOCR can leverage the full range of SQL capabilities to effectively manage and analyze its data.

The Mongo Shell is an important component of MongoDB's open-source distributions, providing users with the ability to query, update, and perform administrative operations on data (Alexander S. Gillis, 2023). MongoDB, a NoSQL DBMS, follows a single master architecture for data consistency, supplemented by secondary databases that maintain copies of the primary database. Additional technologies supporting MongoDB include MongoDB Stitch, Atlas Global Clusters, Mobile, and newer updates (Alexander S. Gillis, 2023). MongoDB is designed to handle large volumes of structured and unstructured data and offers vertical and horizontal scalability. It provides various features such as replication, load balancing, schema-less database structure, horizontal scalability, sharding, storage engines, and aggregation. The advantages of MongoDB include its schemaless nature, allowing for flexible data storage, as well as its compatibility with various data processing frameworks like Hadoop and Spark (Gillis, 2023). The scalability and sharding capabilities of MongoDB contribute to the effectiveness of the USOCRS in handling growing data volumes (Alexander S. Gillis, 2023). The USOCRS benefits from MongoDB's ability to scale horizontally and manage increasing data demands efficiently.

By utilizing both SQL and NoSQL databases, the USOCR can effectively address its diverse data needs. SQL databases like PostgreSQL provide robustness, data consistency, and reliability, ensuring the integrity of critical information. On the other hand, NoSQL databases like MongoDB offer flexibility, scalability, and compatibility with modern data processing frameworks, enabling efficient management of dynamic and unstructured data.

Azure Active Directory (AAD)

Azure Azure Active Directory (AAD) is a cloud-based solution for identity and access management, offering secure authentication and authorization for applications and services using user credentials (Microsoft, 2023). AAD serves as Microsoft's native identity management service in the cloud, catering to organizations of all sizes, including Nokia. It encompasses various features that enhance user identity protection and organizational security, including multi-factor authentication, conditional access, and identity protection. Additionally, AAD provides developers with the necessary tools to incorporate authentication and authorization into their applications. It supports widely adopted protocols like OAuth2 and OpenID Connect, enabling developers to authenticate users and obtain access tokens for accessing protected resources within organizational workflows or applications (Microsoft, 2023) One notable feature of AAD is Multi-Factor Authentication (MFA), which adds a layer of security to the authentication process. MFA requires users to provide multiple pieces of evidence, such as a password and a code sent to their registered phone, to access protected resources. This helps mitigate the risk of unauthorized access, even if a user's password is compromised (Microsoft, 2023).

Another valuable feature offered by AAD is Conditional Access, which empowers administrators to define rules and policies governing user access to resources within their organization or application (Microsoft, 2023). These policies can be based on factors such as user location, device type, or the sensitivity of the resource being accessed. Furthermore, Azure AD Identity Protection utilizes machine learning algorithms to identify and prevent identity-based attacks and data breaches. By analyzing user activity, such as login patterns, this feature detects anomalies that may indicate a potential attack or compromise. In response, Azure AD Identity Protection can automatically block user access or prompt additional authentication steps to verify their identity (Microsoft, 2023).

CI/CD Tools

Agile methodologies prioritize customer collaboration, adaptability to changes, and the regular delivery of functioning software. Consequently, CI/CD integration is an approach in software development that centers around continuous testing, integration, and deployment of software changes. CI/CD, supported by both development and operations teams working in an agile manner with either a DevOps or site reliability engineering approach, aims to frequently provide applications to customers through the incorporation of automation throughout the development stages. The acronym CI/CD encompasses various meanings, including continuous integration, continuous delivery, and continuous deployment. Depending on the level of automation integrated into the CI/CD pipeline, CI/CD workflows involve a significant amount of ongoing automation and continuous monitoring. Continuous integration (CI) is particularly crucial in cloud-native development, enabling developers to merge their code changes back to a shared branch more frequently (RedHat Inc., 2022). Continuous delivery (CD) extends CI by automating the release of a production-ready build to a code repository. Lastly, continuous deployment represents the final stage of a mature CI/CD pipeline, automating the release of an application to production. While CI/CD practices reduce the risk associated with application deployment, they require substantial upfront investment. CI/CD tools, such as Jenkins, Tekton Pipelines, Spinnaker, GoCD, Concourse, Screwdriver, and managed CI/CD tools, aid teams in automating development, deployment, and testing processes. Additionally, DevOps tools like configuration automation, container runtimes, and container orchestration play a crucial role in CI/CD workflows (RedHat Inc., 2022).

Another aspect of the DevOps practice which is also part of the CI/CD automation is known as Continous Testing. Continuous testing is essential for the rapid deployment of an artifact to production. Continuous testing is a software testing process where tests are run continuously to identify bugs as soon as they are introduced into the repository. In a CI/CD pipeline, continuous testing is typically performed automatically, with each code change triggering a series of tests to ensure that the application is still working as expected before the changes are added. According to GitLab, (2023), eight fundamental elements of CI/CD help ensure maximum efficiency for the development lifecycle: a single source repository, frequent check-ins to the main branch, self-testing builds, static pre-build testing scripts, stable testing environments, and fewer context switching. CI/CD fundamentals include a single source repository, frequent check-ins to the main branch, self-testing builds, static pre-build testing scripts, stable testing environments, less firefighting, and less context switching (GitLab, 2023). The USOCRS application makes extensive use of some of these CI/CD tools such as Jenkins and Git. We will briefly give a review of the tools used within our application.

Graphic designers, software developers, and other professionals rely on Version Control Systems (VCS) to manage their development process and track changes in their work (Atlassian, 2023). VCS tools like RCS are commonly used to store different versions of files and enable users to revert to previous states, compare changes over time, and identify contributors responsible for specific modifications. While Centralized Version Control Systems (CVCS) such as CVS, Subversion, and Perforce offer benefits like improved collaboration and centralized control over projects, they also come with notable drawbacks (Atlassian, 2023).

A significant concern with CVCS is the reliance on a single server that houses all the versioned files, with multiple clients checking out files from this central location. This architecture introduces a potential point of failure, as any issues or failures with the

centralized server can lead to the loss of all data. For instance, if the central database's hard disk becomes corrupted or damaged without proper backups, the entire project could be lost (Atlassian, 2023). It's important to note that local VCS systems face similar risks since they typically store version history on the local machine. Any data loss or damage to the local storage can result in irretrievable loss of project information (Atlassian, 2023).

To mitigate the risks associated with a single point of failure, alternative VCS strategies like Distributed Version Control Systems (DVCS), such as Git, have gained popularity. DVCS enables users to create multiple copies (clones) of the repository, providing redundancy and reducing the likelihood of catastrophic data loss (Atlassian, 2023).Git is a distributed version control system that allows many developers to work on the same project at the same time wherever they are. It was developed by Linus Torvalds in 2005 the developer of Linux OS and has since become one of the most popular version control systems in use today (Bitbucket, 2023). Git is a free and well-managed open-source software by many developers contributing to the codebase. It allows developers to manage and maintain code and file changes. It uses a decentralized approach, i.e., every developer has a copy of the entire repository in their local machine or workspace, including all branches and the history of the project they are working on. It also provides a strong mechanism for branching and merging, staging, security, collaboration, pull requests, code reviews, and access controls (Driessen, 2010). Git is one of the essential tools in almost every software development, with many companies and open-source projects using it to manage their project codebase. It also has so far, many popular web-based repositories and services that allow projects to be managed in their system. Services such as GitHub, GitLab, and Bitbucket, provide many additional services which include issue tracking of the project in development, continuous integration, and deployment. As software development continues to evolve, Git remains an important part of every software development process.

In their study, Valentina (2015) look into the topic of Continuous Delivery using Jenkins, an Open-Source CI Platform. The paper explores how Jenkins evolved from being primarily a Continuous Integration tool to a powerful Continuous Delivery tool, owing to its adaptability and extensive ecosystem. Initially designed for automating build and test processes, Jenkins has gained popularity as the most widely used CI tool due to its customizable nature and vast collection of plugins developed by the Open-Source Community. This plugin-based architecture and open-source nature have encouraged developers worldwide to contribute to Jenkins, expanding its capabilities and integration with external tools and technologies. The project originated as an internal endeavor at Sun Microsystems and was later released as open-source software in 2005. In 2011, a dispute over the Hudson trademark led to a fork in the project, resulting in the creation of Jenkins. Jenkins is built with Java and can run on various operating systems. Its web-based graphical interface enables users to configure and manage their projects, while the extensive plugin ecosystem allows for further customization and functionality expansion (Valentina, 2015).

Jenkins has undergone significant enhancements and extensions to transform into a comprehensive Continuous Delivery (CD) platform, fostering collaboration among development (Dev), quality assurance (QA), and operations (Ops) teams through a unified orchestrator (Rayanagoudar, Hampannavar, Pujari, & Parvati, 2018). This evolution enables the chaining of various steps involved in cross-team pipelines and automates their execution, encompassing code checkout, unit tests, static code analysis, performance tests, release of binaries, and deployment into different environments (Rayanagoudar et al., 2018).

The integration capabilities of Jenkins extend to various tools such as Git, GitHub, and JIRA, making it a widely used open-source automation server in software development for continuous integration and continuous delivery (Rayanagoudar et al., 2018). In the realm of DevOps, Jenkins and Puppet/Chef are the most prevalent IT Automation Tools used for infrastructure setup, streamlining the installation of required software, middleware, and dependencies (Rayanagoudar et al., 2018).

Traceability holds a crucial role in software development, enabling the verification of compliance and the ability to trace the root cause of issues (Rayanagoudar et al., 2018). Jenkins serves as an orchestration tool for managing product lifecycles, including code-related processes and deployments. The Puppet and Chef plugin, built upon the Notification plugin, empowers Jenkins to receive deployment notifications from Puppet or Chef, providing a comprehensive artifact history for DevOps and simplifying error debugging (Rayanagoudar et al., 2018).

To address the challenges of CD, Jenkins and Workflow have emerged as the go-to orchestrators for various phases of the product lifecycle (Rayanagoudar et al., 2018). The Workflow engine, integrated as a plugin, introduces a Groovy Domain Specific Language that offers flexibility and customization, revolutionizing traditional scripting practices (Rayanagoudar et al., 2018). This plugin allows the definition of CD workflows, offers the ability to retry pipelines from the last successful checkpoint, enables human intervention through pausing, and provides a dynamic graphical visualization tool for improved control and debugging (Rayanagoudar et al., 2018).

Nevertheless, there are still two main challenges when transitioning from Continuous Integration (CI) to CD: versioning continuously shippable artifacts and capturing the environment information for artifact generation (Rayanagoudar et al., 2018). To address these, Jenkins incorporates the concept of snapshots to avoid excessive release versions and branches (Rayanagoudar et al., 2018). While Jenkins maintains artifact traceability within its ecosystem, capturing the environment used for artifact creation remains a limitation (Rayanagoudar et al., 2018).

Jenkins has made notable progress in facilitating complex workflow creation, enhancing traceability, reducing Time to Market, and improving productivity through plugins like Notification and Workflow Engine (Rayanagoudar et al., 2018). However, further advancements are necessary for Jenkins to fully embrace the CD revolution (Rayanagoudar et al., 2018).

Widiyanto, Anindito, and Azam (2020) provide an analysis of DevOps practices utilizing Docker Container as a platform for application development. The study focuses on designing continuous integration/continuous delivery (CI/CD) workflows to integrate git repositories into production servers. For the purpose of this research, the authors selected specific software tools, including Gitea as an open-source Git Server repository manager, Jenkins as an open-source automation server, and Docker as an open platform for application development, deployment, and execution. The research encompasses two main stages: the Software Setup Stage and the Software Integration Stage. In the Software Setup Stage, Gitea is installed on the server to serve as the primary repository for collaborative project development. Jenkins, an automation server, is used to execute software development commands automatically. Docker, on the other hand, is a lightweight and portable platform that enables developers to package applications and their dependencies into containers, ensuring consistent performance across various environments and infrastructures.

Moving to the Software Integration Stage, the researchers discuss creating credentials on Jenkins to establish authentication permissions for the Production Server. Docker is described as a popular Linux-based platform for containerization, enabling the packaging, shipping, and running of applications in isolated environments. Docker Compose, a tool for managing multi-container configurations in a single file, facilitates the creation and management of these containers. It offers a loosely isolated environment, known as a container, which includes all the necessary components for running an application.

Docker's container-based platform provides responsive deployment and scalability, allowing for the easy management of workloads by dynamically scaling applications and services based on business needs. As a lightweight and fast alternative to hypervisor-based virtual machines, Docker offers a cost-effective solution. The communication between the Docker client and daemon occurs through a REST API over UNIX sockets or a network interface. The Docker daemon (dockerd) is responsible for handling Docker API requests and managing Docker objects, such as images, containers, networks, volumes, plugins, and others. Docker Hub, a public registry, is the default location for finding Docker images (Docker Inc., 2023).

The Verification toolbox is an internal tool developed by engineers in Nokia for Nokia. Verification Toolbox is a cross-platform tool for testing both virtual and real embedded devices and systems on chips. It provides a consistent and well-organized data format for test results in SoC development, which can be transmitted using the Unified Verification Report data format, which is a JSON schema. The Python API is offered to incorporate the report format into automation utilizing Python scripts and command-line tools and is tested on Linux-based systems and semantic versioned, while it can also be used with Windows PCs by using virtualization. The decoupling of report data sources and report targets made possible by a single format enables the independent development of procedures. The report data format is intended to deliver the data in an orderly and predictable way, focus on communicating data produced by the report provider, provide references for traceability and reproduction, be simple to construct and maintain, and be adaptable to various testing phases and environments.

3. Problem and Research Methodology

3.1 Research method

This chapter provides an in-depth exploration of the research methodology which is used in the development of the USOCRS. The approach adopted in this study integrates Design Science Research (DSR) principles and incorporates relevant concepts from existing literature. By delving into the fundamental elements of DSR and their significance to the research objectives, this chapter aims to elucidate the step-by-step process employed in developing USOCRS, emphasizing the rationale behind each stage and the integration of pertinent research frameworks.

Design Science Research (DSR) Approach

In this study, the research methodology aligns with the Design Science Research (DSR) approach introduced by (Hevner, March, Park, & Ram, 2004). DSR is a problem-solving methodology that seeks to create and validate innovative artifacts aimed at addressing specific organizational challenges. The four key elements of DSR, namely design as an artifact, research contribution, research rigor, and communication of the research, serve as the foundation for the research methodology employed in this study.

Design as Artifact

The USOCRS project revolves around the development of a tangible artifact in the form of a RESTAPI service. This service is specifically designed to address the challenges faced by SOC engineers in managing and reporting verification reports. By creating this practical and effective solution, the USOCRS project aims to enhance the overall management and reporting of verification reports within SoC teams.

Research Contribution

The USOCRS project endeavors to make significant contributions to both theoretical and practical aspects of SoC verification. By creating a viable artifact, such as the RESTAPI service, the project aims to improve the existing methods of managing and reporting verification reports within SoC teams.

Research Rigor

To ensure the credibility and reliability of the research outcomes, the research methodology incorporates rigorous research methods throughout the development process. This includes conducting comprehensive literature reviews, following systematic design processes, and employing appropriate evaluation techniques to validate the artifact's effectiveness.

Communication of the Research

Effective communication of the research findings is crucial for disseminating knowledge and facilitating the wider adoption of the developed artifact. In line with this, the research methodology emphasizes clear and concise documentation of the research process, design decisions, and evaluation results. By doing so, the research outcomes can be effectively shared with relevant stakeholders, as well as the academic and industrial communities.

The research methodology incorporates the Framework for Evaluation in Design Science Research (FEDS) proposed by (Venable, Pries-Heje, & Baskerville, 2014). FEDS provides a structured framework for evaluating design science research, utilizing two dimensions: functional purpose and model.

The functional purpose evaluation goals in the USOCRS project are derived from the specific requirements of the RESTAPI service and the desired outcomes. These evaluation goals include assessing the performance, usability, efficiency, accuracy, and user satisfaction of the developed artifact. The model dimension in FEDS refers to the evaluation strategies and methods employed to assess the properties of the artifact. In the USOCRS project, a combination of quantitative and qualitative evaluation methods is utilized. This includes measuring various metrics such as upload/transfer time, validation accuracy, data extraction efficiency, and visualization effectiveness. Additionally, user surveys and feedback are collected to gauge user satisfaction and usability.

3.2 The steps in the Development of the USOCRS

The development of the USOCRS followed a systematic design science research methodology, incorporating the insights and principles outlined in the paper by (Hevner, March, Park, & Ram, 2004). The following steps were followed in the development of the USOCRS:

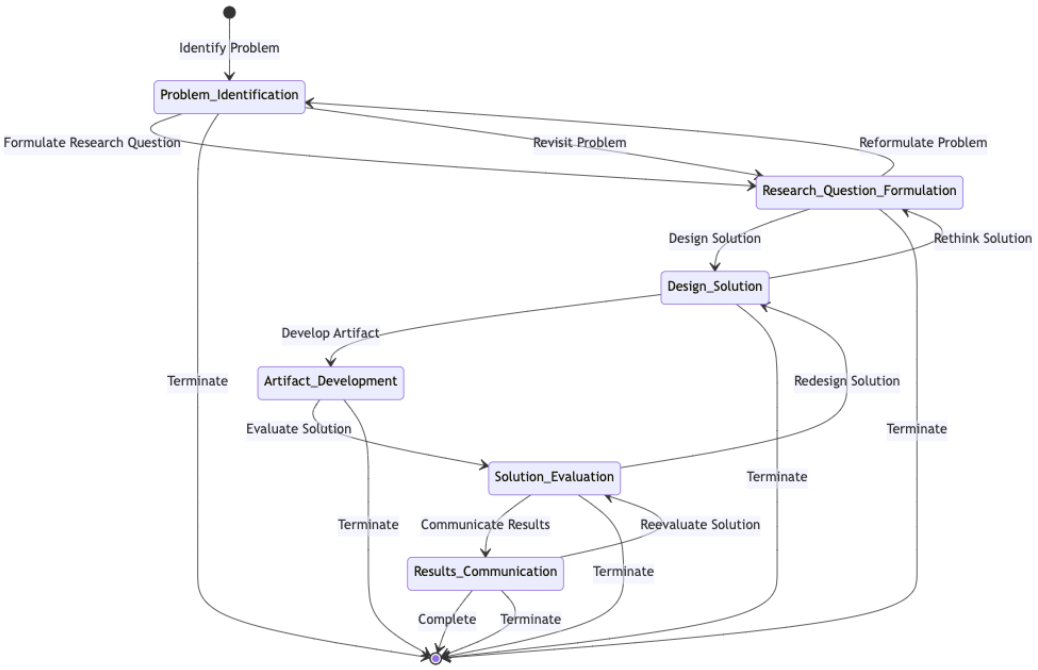

Figure 2. Design Science Research steps followed.

Problem Identification: The first step involved identifying the problem to be solved, which was the lack of a unified and standardized approach for uploading, transferring, and validating verification reports generated by SoC devices within the Nokia SoC Team in Oulu. This manual approach, involving copying data from verification report files to Excel sheets, was time-consuming and reduced productivity.

Research Question Formulation: The research question was formulated to guide the development of the USOCRS: "How can the USOCRS be designed, developed, and validated using available tools and technology to improve the efficiency and effectiveness of verification report transfers, utilization, and interpretation in the context of SoC device testing?"

Design Solution: The next step involved designing a solution to address the identified problem. The solution entailed the development of a USOCRS using various technologies such as Fastapi, SQL, and NoSQL databases, Azure Active Directory for authentication, and Nokia Cloud and Verification Toolbox for validation using specified Verification Schemas.

Artifact Development: The solution was implemented through the creation and deployment of the USOCRS. This involved constructing the necessary artifacts, including the development of the RESTAPI system and integration with the relevant tools and technologies.

Solution Evaluation: The developed solution was evaluated to ensure its effectiveness, efficiency, and alignment with the specific needs of the SoC team. Evaluation metrics included upload/transfer and validation time, the accuracy of the validation process, data extraction, analysis, visualization, and user satisfaction. The evaluation followed the principles outlined in the Framework for Evaluation in Design Science Research (FEDS) proposed by Venable et al. (2014).

Results Communication: The findings of the study were communicated and shared with relevant stakeholders within and beyond the SoC team. The results were presented clearly, highlighting any limitations encountered during the development and evaluation process. Recommendations for future development and optimization of the USOCRS were also provided.

By following this design science research methodology, the development of the USOCRS was carried out systematically, ensuring that the artifact addressed the identified problem and met the specific requirements of the SoC team. The insights from Hevner et al. (2004) and the relevant references discussed in this chat, including (Dresch, 2014)) and (Venable, Pries-Heje, & Baskerville, 2014), provided valuable guidance in conducting rigorous research, incorporating action research, and applying appropriate evaluation strategies.

4. System Architecture

In this section, the architecture of the USOCRS is investigated. The overall problem the system aims to solve has already been reported in the previous chapters and various components of the systems in development are presented. In a nutshell, USOCRS development's goal is to provide an API that enables SoC engineers within the organization to easily post their verification reports through the API for further analysis. It also provides an API for users to efficiently investigate and assess how well systems are doing as they are being developed, as well as to summarize successes and failures, test coverages, and other information. USOCRS will design architecture should focus on scalability, reliability, and security to ensure that it can handle large amounts of requests from multiple users at the same time without suffering performance degradation while providing a user-friendly interface. The API is designed in a way that it is easy to use and allows for smooth integration with other tools such as integration with Frontend and mobile applications or Power Apps via the exposed resources. The functional and non-functional requirement of the system is also described in this section.

The architecture encompasses the client side, the RestAPI service layer, and the underlying infrastructure. The section will briefly discuss the design principle followed to achieve the success of the application and highlights the importance of using modern technologies, such as containerization, to support the scalability and flexibility of the system architecture. The system architecture described in this paper provides a foundation for the successful implementation and deployment of the USOCRS.

4.1 Design Principles for USOCRS Development

These key principles outlined below are the set of rules which are used in the design of the RestAPI service (Len Bass, 2013) :

1. Modularity: The system is designed to be modular which means with well-defined components that can be independently created, tested, and maintained. This allows the application to be easily extended and facilitates the reuse of code.
2. Loose Coupling: The system components are loosely coupled to reduce dependencies and increase the flexibility of the components. The individual components which are separated may be easily replaced, tested, and maintained more easily with loose coupling since it doesn't affect the entire system.
3. Separation of Concerns: The system architecture separates different concerns into different modules or layers such as the routes layer, database layer, verification layer, etc. This separation allows better organization, maintainability, and understanding of the system's functionalities.
4. Scalability: The system is designed to handle a large volume of requests from many users at the same time. Thanks to the FastAPI asynchronous capability which also facilitates various requests asynchronously.
5. Reliability: The system is equipped with high reliability by providing fault tolerance, error handling, and robustness in the case of system failures. Redundancy, degradation, and recovery mechanisms are implemented to minimize downtime and ensure continuous operation such as data backup.
6. Security: To safeguard sensitive data and reduce unwanted access to the system, the system has a strong security mechanism in place. To validate user identities and manage access to sensitive resources, authentication, and authorization

procedures are put into place using the organization user account and AZURE Active Directory (AAD).

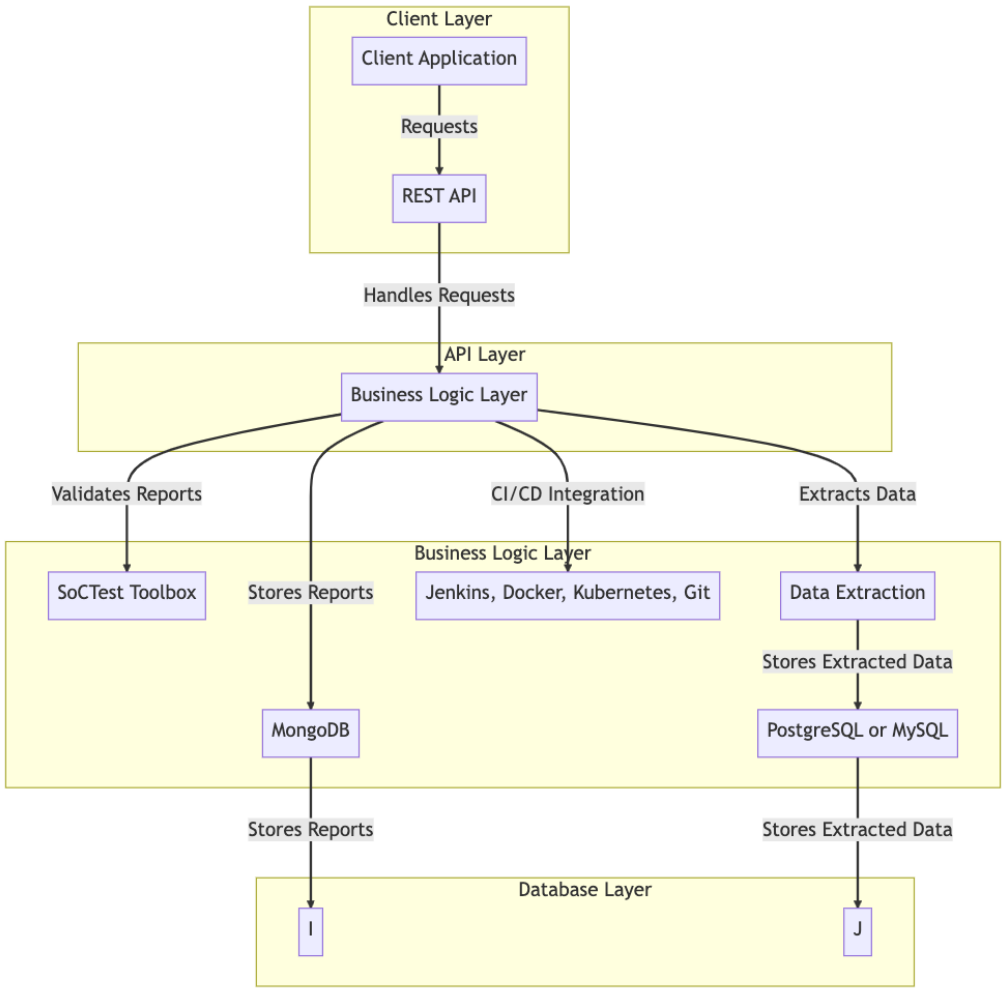

Figure 3. The overall system architecture of the USOCRS

RestAPI Service Layer:

The RestAPI service layer serves as the central component of the USOCRS system architecture. It handles incoming requests from its endpoints or client applications, processes the requests, and provides the necessary functionalities required by the system. The service layer consists of various API endpoints that enable report submission, view of necessary data, and so on. It communicates with the underlying components, such as the database and external services like Verification Toolbox, to fulfill the requested operations.

Containerization:

Containerization technologies, such as Docker, can be employed to package the microservices and their dependencies into an isolated environment. Containerization provides a lightweight and portable solution, ensuring consistent deployment across

different environments. It enables efficient resource utilization and simplifies the deployment and management of the USOCRS. With this, the application is easily sharable for various purposes and uses.

Infrastructure:

The infrastructure layer includes the underlying components and resources required to support the USOCRS. This includes the database such as MongoDB and Postgres database for storing and retrieving SoC verification reports, as well as any external services integrations, such as integration of authentication and authorization services or third-party APIs like Microsoft. The infrastructure is designed to meet the non-functional requirements of scalability, reliability, and security of the system.

This section has presented the system architecture of the USOCRS, including the RESTAPI service layer, and the underlying infrastructure. By leveraging modern architectural approaches, such as containerization, the system architecture provides the necessary flexibility, scalability, and reliability to support the post, analysis, and viewing of SoC verification reports during SoC development. The system architecture described in this paper serves as a steppingstone for the successful implementation and deployment of the USOCRS.

4.2 System requirements

System requirements are vital for software development as they define the functionality needed to meet customer requirements and reduce implementation costs (Siedle, 2015). These requirements usually are taken from the client or customer which can be an individual, company, or groups of people involved in the project like government and so on. Understanding the system requirements is essential for aligning the development process and features with the client's needs. The customer's requirements hold the most importance and are usually the most comprehensive. Additionally, new software projects define initial functional requirements, while projects involving microservices and Web API already have some of these requirements established (Oksa, 2016). In web application development, basic database operations like CRUD for data entity types are necessary. Higher-level interactions involve data aggregation, including options like view models that present reduced data to users. Web applications should prioritize data security. By considering these factors and meeting both functional and non-functional requirements, software projects can successfully fulfill user needs and ensure system effectiveness.

The key objectives of the USOCRS are as follows:

- Automation of Result Collection: The USOCRS will automate the process of collecting SOC verification reports, eliminating the need for manual intervention. By integrating with existing verification tools and technologies, the service will streamline the retrieval of report data, saving time and reducing errors.
- Advanced-Data Visualization and Analytics: The USOCRS will provide an API that enables advanced data visualization and analytics capabilities. This will allow engineers, researchers, and managers to gain valuable insights from the verification reports, facilitating better decision-making and improving the overall verification process.
- Unified Result Format and Storage: The USOCRS will employ a unified result format and storage mechanism, ensuring consistency and adherence to

organizational standards. This approach simplifies the reporting process and enhances interoperability across different verification tools and systems.

- Improved Productivity and Scalability: By automating the reporting process, the USOCRS eliminates repetitive manual work for engineers, freeing up their time for more critical tasks. The system's scalability enables it to handle larger volumes of verification reports efficiently, accommodating the growing complexity and size of SoCs.
- Ease of Data Access: The USOCRS facilitates easy and secure access to SOC verification reports, enabling engineers, researchers, and managers to examine and analyze the data effectively. The API-based approach allows for seamless integration with other systems and tools, empowering users to extract valuable insights from the verification reports.
- Integration of Modern Technologies: The USOCRS leverages modern technologies such as FastAPI for server development, cloud technologies, SQL and NoSQL databases, and Azure Active Directory for secure authentication. This integration ensures a robust and secure infrastructure for the service, enhancing its reliability and scalability.
- Validation and Compliance: The USOCRS incorporates validation mechanisms using the Verification Toolbox and Verification Schemas developed by Nokia. This ensures the consistency, reliability, and compliance of the verification reports, adhering to industry standards and best practices.
- CI/CD Integration: The USOCRS will use Continuous Integration/Continuous Deployment (CI/CD) practices to automate essential processes such as data backup, testing, building, and deployment. This integration ensures a reliable and up-to-date system, enabling efficient optimization, management, and maintenance of the service.

4.2.1 System Requirement Collection Service USOCRS

The system requirement for the USOCRS was collected through various meetings with the line manager, where the application and its functionalities were discussed in detail. This subchapter provides an overview of how the system requirement was gathered, including the discussion on technology choices and the specific requirements identified by the line manager.

1. Meeting with the Line Manager: The requirement-gathering process started with a meeting with the line manager, who described the USOCRS application, its purpose, and the type of data it will process. The line manager outlined the key functionalities and data flow of the application, giving insights into the expected behavior and outcomes of the application.
2. Description of Application and Data Handling: During the meeting, the line manager described the application's primary function, which is to receive unified reports through API calls and store these reports in a database to reduce the latency of data. The USOCRS service needs to handle these reports and perform various operations on them. Additionally, it was discussed that the service should maintain a log of the received API calls, including details such as the date, time, and caller ID.
3. Technology Discussion: The choice of technologies for API development was also deliberated in the meeting. Flask and FastAPI were the options considered, and the benefits of FastAPI, which were discussed earlier in this conversation, led

to the final decision of selecting FastAPI as the preferred framework for the USOCRS development.

4. Specific Requirements Identified: The line manager also identified several specific requirements for the USOCRS, which were discussed and documented during the meeting.

These requirements include:

a. Log API Calls: The service should keep a log of the received API calls, recording details such as the date, time, and caller ID. Additional log details to be captured will be specified.
b. Report Storage: The service should store the received reports as they are for future use. This ensures that the original reports are preserved and can be accessed when needed.
c. Report Validation: The service should validate the correctness of the reports. This involves performing checks and verification to ensure that the received reports adhere to the expected format and content based on the organization's specifications.
d. Artifactory Integration: The service should propagate the validated reports to Artifactory, which is responsible for further processing. This integration allows seamless data transfer and collaboration between systems.
e. Report ID Generation: The service should create a unique ID for each report, facilitating efficient tracking and management of the reports throughout the system.
f. Report Processing and Database Storage: The service should process the reports and store the relevant information in a database. This includes storing data such as the report ID, report date, program, project, instance, regression target, milestone, RTL version, version, test plan location, JIRA key, the total number of cases, number of passing cases, number of failing cases, functional coverage, and code coverage.
g. Individual Test Case Storage: The service should store information about individual test cases included in the reports. This includes linking each test case to the corresponding report ID, capturing the test case name, test case result, and information about any requirements associated with the test case.
h. Database Backup Management: As the service will handle critical data, it should include functionality to manage database backups since this is not provided as a service by the Nokia cloud service. Regular backups ensure data integrity and facilitate disaster recovery processes.
i. Data Security: The service should include security of the data processed by the application which might contain critical information. Hence, access to the service should only be possible to users within the organization's ecosystem.
j. Data access: The system should allow multiple calls to the API without downtime.

By gathering these requirements from the line manager, a comprehensive understanding of the expected functionality and features of the USOCRS was obtained. These requirements act as the basis for the development and implementation of USOCRS.

4.3 Functional Requirements:

This section provides an overview of the functional requirements of the USOCRS. The USOCRS aims to provide an API that simplifies the effective submission, analysis, and investigation of SoC verification reports in the context of system-on-chip (SoC) development. Functional requirements are an important part of every project as it shows the exact features expected of the system in development (AltexSoft, 2021).

This paper presents the key functional requirements of the USOCRS, which include report submission via HTTP Post verb, report viewing, and report tracking. These requirements are important for supporting the development and verification process of USOCRS designs. The paper also highlights the significance of scalability, reliability, and security in ensuring the successful operation of the USOCRS. The functional requirements of the system are shown in the table below.

Table 1. Functional Requirements

Requirements	Description
User Authentication	The system should support Microsoft Azure Active Directory (AAD) authentication. Provide easy authentication for users within the organization. Users should be able to authenticate using their Microsoft accounts to access the API i.e., their working account. OAuth2 with Authorization Code flow should be implemented for secure authentication.
Verification Report Submission	SoC engineers should be able to easily post their verification reports through the API via the command line, terminal, or web. The API should accept verification reports as a file in a specified format (e.g., JSON or Text). The system should validate the report using the Verification Toolbox and store the submitted reports in a database for further analysis to reduce latency. The system should extract this data from the stored database and store the new data in a Relational Database (RDBMS) like Postgres Database.
Verification Report Deletion	SoC engineers should be able to easily delete uploaded reports by report ID or report IDENTIFIER via the command line, terminal, or web. The system should delete the report from the database.
System Assessment	The API should provide functionality for users to view and assess the progress of system development through the exposed API resources. Users should be able to access information about system successes, failures, and test coverage from the submitted report. The API should allow users to retrieve summarized information and metrics related to the system verification report.
Reporting and Analytics	The system should generate comprehensive reports and visualizations based on the submitted verification reports. Reports should include analysis results, system performance metrics, and test coverage information.

4.3.1 Non-Functional Requirements:

This section provides an overview of the non-functional requirements of the USOCRS. The USOCRS aims to provide a reliable, scalable, and secure API for the submission, analysis, and investigation of SoC verification reports. This section presents the key non-functional requirements of the USOCRS, including performance, scalability, reliability, security, and usability. These requirements are critical for ensuring the successful operation and adoption of the USOCRS (AltexSoft, 2021). The paper also highlights the importance of adhering to industry best practices and standards to meet these non-

functional requirements effectively. Below are the key non-functional requirements of the system.

Table 2. Non-Functional Requirements

Requirement	Description
Scalability	The system should be able to handle many requests from multiple users concurrently.
Reliability	The API should be highly reliable and available to users at all times. Fault tolerance mechanisms and redundancy should be in place to minimize downtime.
Security	The API should ensure the security of data and communications through the internet. Data encryption and secure communication protocols such as HTTPS will be used to protect sensitive information. Access control mechanisms will be implemented to enforce data privacy and integrity.
Integration	The API should be designed to allow smooth integration with other tools. The API should expose resources and endpoints that facilitate integration with external systems and allow Integration with other tools such as frontend applications, mobile applications, and Power Apps should be supported through the exposed API resources.
Usability	The API should be easy to use and have a well-documented interface. Clear documentation, including API endpoints, parameters, and responses, will be provided. Error handling and error messages should be informative and user-friendly.
Performance	The system should be optimized for performance to ensure fast response times that's why FastAPI has been a choice of framework. The API is designed to minimize response times and ensure low latency for report submission, and analysis. Efficient data processing processes, caching mechanisms offered by the framework, and optimized database queries can contribute to achieving high performance. Performance testing should be conducted to identify and address any performance bottlenecks.

These requirements provide flow to the design and development of the USOCRS, making sure that it meets the needs of SoC engineers and users of the system within the organization ecosystem.

4.4 Generating Use Cases for the USOCRS

Use cases for the USOCRS were generated based on the system requirements gathered from my meetings with the line manager. The use cases provide a detailed description of the interactions between actors and the system, outlining the specific functionalities and scenarios that the USOCRS should support. This subchapter provides an overview of how the use cases were generated for the USOCRS.

1. Analyzing System Requirements: To generate the use cases, the system requirements identified during the meeting with the line manager were carefully analyzed. Each requirement was examined to determine the various actions and interactions that the USOCRS needs to support. The identified requirements provided a clear understanding of the system's goals, inputs, outputs, and expected behaviors.
2. Identifying Actors: Actors in a use case represent the entities or individuals interacting with the system. In the context of the USOCRS, the primary actors can be identified as follows:
    a. User: The user, such as a software engineer or a test engineer, initiates API calls and interacts with the USOCRS to perform various operations.
    b. USOCRS: Refer to as "System" acts as a secondary actor that receives the validated reports from the API for further processing.

4.4.1 User cases

Use Case Name: Receive and Process SoC Verification Report

Primary Actor: System

Preconditions:

- The system is running and connected to the internet.
- The necessary dependencies like FastAPI, MongoDB driver, Postgres Driver, and Verification toolbox have been installed.

Postconditions:

- The verification report is received.
- The report is validated with the Verification toolbox.
- The report is stored in the MongoDB database.
- Meaningful data is extracted from the stored report.
- The system stores the extracted report in a new database (Postgres)
- The data is presented via Python FastAPI RESTful to the user.
- The complete system is running in a cloud environment through CI/CD integration.
- Produced is stored in Artifactory.

Main Success Scenario:

1. The system receives the verification report in JSON format.
2. The system validates the stored report using the Verification toolbox.
3. The system stores the report in the MongoDB database.
4. The system extracts meaningful data from the stored report.
5. The extracted data is stored in a new database (MySQL or Postgres)
6. The system creates a Python FastAPI to present the extracted data to the user.
7. The user accesses the API and receives the extracted data.
8. The system integrates with CI/CD tools like Jenkins, Docker, Kubernetes, and Git to automate the deployment process and store the complete system in Artifactory.

Extensions:

1a. If the simulation report is not received in JSON format:

- The system sends an error message to the user and terminates the process.

2a. If the report fails validation with the Verification toolbox:

- The system sends an error message to the user and terminates the process.

3a. If the report cannot be stored in the MongoDB database:

- The system sends an error message to the user and terminates the process.

4a. If meaningful data cannot be extracted from the stored report:

- The system sends an error message to the user and terminates the process.

5a. If the report cannot be stored in the new database:

- The system sends an error message to the user and terminates the process.

6a. If the Python FastAPI cannot be created to present the extracted data:

- The system sends an error message to the user and terminates the process.

8a. If the integration with CI/CD tools fails:

- The system sends an error message to the user and terminates the process.

This use case represents the process of the USOCRS API processes. The USOCRS API conveniently and efficiently processes and validate the SoC verification reports using the Verification Toolbox schema format. The use of the SoC Verification Test Toolbox is to validate the report making sure the report conforms to the required specification of the SoC schema.

Use Case: SoC Engineer - Verification Report Submission

Actor: SoC Engineer

Goal: Submit verification reports through the USOCRS for further analysis.

Preconditions:

- The SoC engineer is authenticated and has access to the USOCRS API.

Main Flow:

1. The SoC engineer initiates the SoC verification report submission process.
2. The SoC engineer prepares the SoC verification report as a file in the required format (e.g., JSON or Text).
3. The SoC engineer sends an HTTP POST request to the designated endpoint of the USOCRS API, including the verification report as a file payload to the API.
4. The USOCRS API receives the POST request and validates the file format and content of the verification report.
5. If the verification report passes the validation, the USOCRS API stores the report in the database for further analysis.

6. The USOCRS API sends a successful response back to the SoC engineer with stored IDs, indicating that the verification report has been successfully submitted.

Alternative Flow:

5a. If the verification report fails, the validation:

1. The USOCRS API sends an error response back to the SoC engineer, specifying the validation errors encountered.
2. The SoC engineer corrects the validation errors and resubmits the verification report.

Postconditions:

- The verification report is stored in the USOCRS system database for further use.
- The SoC engineer receives a confirmation of the successful verification report submission.

Exceptions:

1. If the SoC engineer is not authenticated or does not have access to the USOCRS API:
    - The USOCRS API returns an unauthorized access error response, and the verification report submission process is terminated.
2. If there is a communication failure between the SoC engineer and the USOCRS API:
    - The USOCRS API returns an error response indicating the failure, and the verification report submission process is terminated.
3. If there are technical issues or system errors during the submission process:
    - The USOCRS API returns an error response specifying the issue, and the verification report submission process is terminated.

This use case represents the process of an SoC engineer using the USOCRS API to submit verification reports. The USOCRS API provides a convenient and efficient way for SoC engineers to contribute to the analysis of system development by securely submitting their verification reports. The use of the API ensures that the reports are standardized and easily integrable with other tools and workflows.

Use Case: Analyzing System Verification Progress

Actor: User

Description:

The user, a stakeholder, or a member of the development team, wants to efficiently view and assess the progress of the SoC system using the USOCRS. The API provides features and resources that enable the user to access relevant information, analyze successes and failures, monitor test coverages, and retrieve needed data from the service.

Preconditions:

- The user has appropriate credentials or privileges to access the USOCRS API.

- The user has access to a client application (e.g., a Power App, web-based dashboard, or a mobile app) that interacts with the API.

The flow of Events:

1. The user logs into the client application and navigates to the verification progress section.
2. The client application sends a request to the USOCRS API to retrieve the relevant verification progress data.
3. The API authenticates the user and validates the request.
4. The API fetches the necessary data from the underlying database or verification system.
5. The API returns the verification progress data to the client application.
6. The client application presents the progress information in a user-friendly manner, such as charts, graphs, or tables.
7. The user can interact with the presented data to drill down into specific details or filter the information.
8. The client application may provide options to generate custom reports based on the verification progress data.
9. The user can analyze the system's successes, failures, and test coverages to gain insights into the verification process.
10. If desired, the user can export the progress information or reports in different formats for further analysis or sharing.

Alternate Flow:

- If the user's authentication fails or the request is invalid, the API returns an error response, and the client application notifies the user of the issue.

Postconditions:

- The user has accessed and analyzed the verification progress information using the USOCRS API.
- The user can make informed decisions regarding the system's development based on the insights gained from the verification progress analysis.

This use case demonstrates how the user can use the USOCRS API to efficiently monitor and assess the progress of the system through the SoC verification reports. By accessing the API's features and resources, the user can analyze successes, failures, and test coverages, gaining meaningful insights into the system's development phases. The integration with client applications or power apps provides a user-friendly interface for presenting the verification progress data and allows for customization and export options to facilitate further analysis and reporting.

5. Implementation

The previous chapter is a review of the system architecture of the USOCRS which we also include a use case for the system, a use case for the SoC engineer, and a use case for the users of the system. This chapter, discussion on the implementation of USOCRS. This section will concentrate on the tools, technologies, and frameworks used to develop the service, as well as the data models and algorithms used to support its functionality. This section also provides a detailed analysis of the SQL models used in the USOCRS.

5.1 Data Types and Models

The USOCRS is a RESTAPI service that aims to provide a unified platform for managing system-on-chip designs. The service relies on a well-defined database schema, represented by SQL models which will be extracted from the SoC verification reports. Examination of the SQL models used in the USOCRS and explaining their data types and relationships are looked into in detail. The SQL models define the database schema for the service and include tables such as Detail, Coverage, Implementation, InputParameter, OutputParameter, Parameter, Subject, Summary, TestPlan, Batch, User, and Report. This paper explores the data types and relationships between these models, emphasizing their significance in the USOCRS database system.

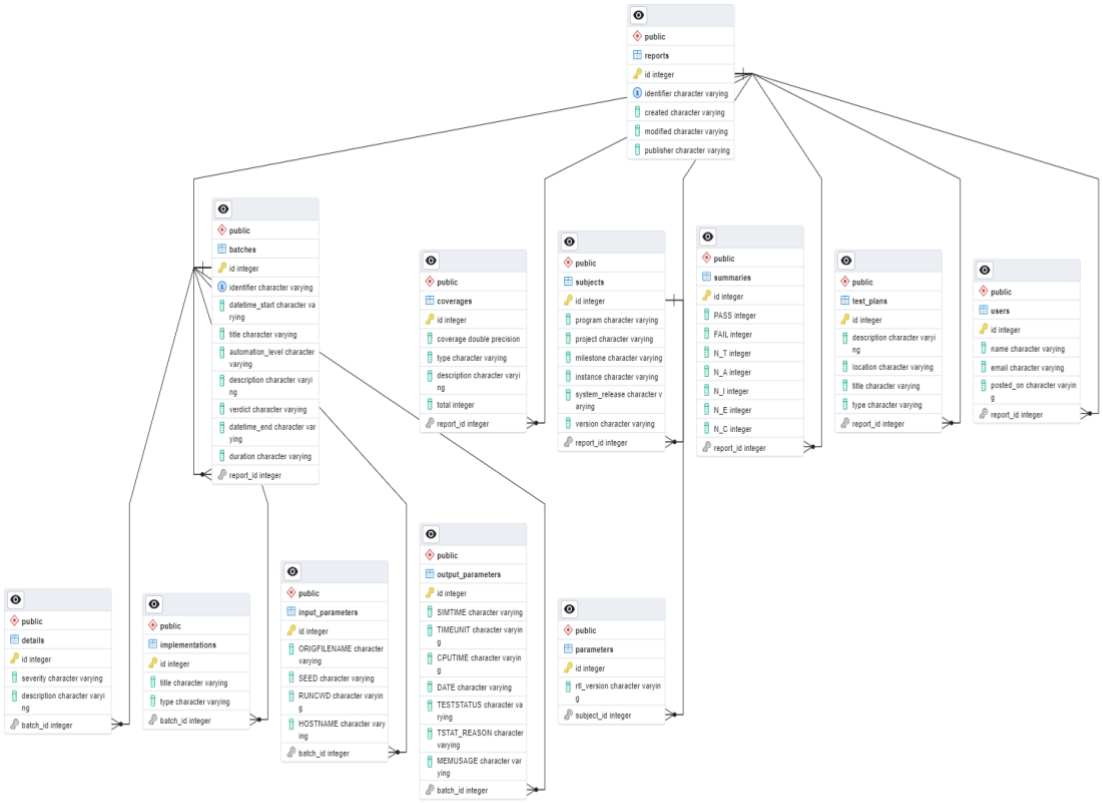

Figure 4. Postgres Entity-Relationship Diagram (ERD) image of the System Models.

5.1.1 SQL Models and Data Types

Detail Model

The Detail model represents the "details" table in the database. It contains the following attributes:

- id: An integer representing the primary key.
- severity: A string representing the severity of the detail.
- description: A string describing the detail.

Coverage Model

The Coverage model corresponds to the "coverages" table. It includes the following attributes:

- id: An integer representing the primary key.
- coverage: A float representing the coverage percentage.
- type: A string representing the type of coverage.
- description: A string describing the coverage.
- total: An integer representing the total coverage.

Implementation Model

The Implementation model represents the "implementations" table. It consists of the following attributes:

- id: An integer representing the primary key.
- title: A string representing the title of the implementation.
- type: A string representing the type of implementation.

InputParameter Model

The InputParameter model corresponds to the "input_parameters" table. It includes the following attributes:

- id: An integer representing the primary key.
- ORIGFILENAME: A string representing the original filename.
- SEED: A string representing the seed value.
- RUNCWD: A string representing the running working directory.
- HOSTNAME: A string representing the hostname.

OutputParameter Model

The OutputParameter model represents the "output_parameters" table. It contains the following attributes:

- id: An integer representing the primary key.
- SIMTIME: A string representing the simulation time.
- TIMEUNIT: A string representing the time unit.
- CPUTIME: A string representing the CPU time.
- DATE: A string representing the date.
- TESTSTATUS: A string representing the test status.
- TSTAT_REASON: A string representing the test status reason.

- MEMUSAGE: A string representing memory usage.

Parameter Model

The Parameter model corresponds to the "parameters" table. It includes the following attribute:

- id: An integer representing the primary key.
- rtl_version: A string representing the RTL (Register Transfer Level) version.

Subject Model

The Subject model represents the "subjects" table. It contains the following attributes:

- id: An integer representing the primary key.
- program: A string representing the program associated with the subject.
- project: A string representing the project associated with the subject.
- milestone: A string representing the milestone associated with the subject.
- instance: A string representing the instance associated with the subject.
- system_release: A string representing the system release associated with the subject.
- version: A string representing the version associated with the subject.

Summary Model

The Summary model corresponds to the "summaries" table. It includes the following attributes:

- id: An integer representing the primary key.
- PASS: An integer representing the number of passed tests.
- FAIL: An integer representing the number of failed tests.
- N_T: An integer representing the total number of tests.
- N_A: An integer representing the number of tests with an "A" status.
- N_I: An integer representing the number of tests with an "I" status.
- N_E: An integer representing the number of tests with an "E" status.
- N_C: An integer representing the number of tests with a "C" status.

TestPlan Model

The TestPlan model represents the "test_plans" table. It includes the following attributes:

- id: An integer representing the primary key.
- description: A string describing the test plan.
- location: A string representing the location of the test plan.
- title: A string representing the title of the test plan.
- type: A string representing the type of the test plan.

Batch Model

The Batch model corresponds to the "batches" table. It consists of the following attributes:

- id: An integer representing the primary key.
- identifier: A string representing the unique identifier of the batch.
- datetime_start: A string representing the start datetime of the batch.

- title: A string representing the title of the batch.
- automation_level: A string representing the automation level of the batch.
- description: A string describing the batch.
- verdict: A string representing the verdict of the batch.
- datetime_end: A string representing the end datetime of the batch.
- duration: A string representing the duration of the batch.

User Model

The User model represents the "users" table. It includes the following attributes:

- id: An integer representing the primary key.
- name: A string representing the name of the user.
- email: A string representing the email address of the user.
- posted_on: A string representing the posted datetime of the user.

Report Model

The Report model corresponds to the "reports" table. It contains the following attributes:

- id: An integer representing the primary key.
- identifier: A string representing the unique identifier of the report.
- created: A string representing the creation datetime of the report.
- target: A string representing the coverage of the report.
- modified: A string representing the modification datetime of the report.
- publisher: A string representing the publisher of the report.

5.1.2 Relationships between Models

Detail and Batch Relationship

The Detail model has a many-to-one relationship with the Batch model. Each detail belongs to a specific batch, and the relationship is established through the foreign key constraint on the "batch_id" column in the Detail table.

Coverage and Report Relationship

The Coverage model has a many-to-one relationship with the Report model. Each coverage entry belongs to a specific report, and the relationship is established through the foreign key constraint on the "report_id" column in the Coverage table.

Implementation and Batch Relationship

The Implementation model has a many-to-one relationship with the Batch model. Each implementation belongs to a specific batch, and the relationship is established through the foreign key constraint on the "batch_id" column in the Implementation table.

InputParameter and Batch Relationship

The InputParameter model has a many-to-one relationship with the Batch model. Each input parameter entry belongs to a specific batch, and the relationship is established through the foreign key constraint on the "batch_id" column in the InputParameter table.

OutputParameter and Batch Relationship

The OutputParameter model has a many-to-one relationship with the Batch model. Each output parameter entry belongs to a specific batch, and the relationship is established through the foreign key constraint on the "batch_id" column in the OutputParameter table.

Parameter and Subject Relationship

The Parameter model has a many-to-one relationship with the Subject model. Each parameter belongs to a specific subject, and the relationship is established through the foreign key constraint on the "subject_id" column in the Parameter table.

Subject and Report Relationship

The Subject model has a one-to-one relationship with the Report model. Each subject is associated with a specific report, and the relationship is established through the foreign key constraint on the "report_id" column in the Subject table.

Summary and Report Relationship

The Summary model has a one-to-one relationship with the Report model. Each summary is associated with a specific report, and the relationship is established through the foreign key constraint on the "report_id" column in the Summary table.

TestPlan and Report Relationship

The TestPlan model has a one-to-one relationship with the Report model. Each test plan is associated with a specific report, and the relationship is established through the foreign key constraint on the "report_id" column in the TestPlan table.

Batch and Report Relationship

The Batch model has a many-to-one relationship with the Report model. Each batch belongs to a specific report, and the relationship is established through the foreign key constraint on the "report_id" column in the Batch table.

User and Report Relationship

The User model has a one-to-one relationship with the Report model. Each user is associated with a specific report, and the relationship is established through the foreign key constraint on the "report_id" column in the User table.

5.2 Algorithms

The USOCRS uses several algorithms to support the functionality and efficiency of the system. These algorithms include authentication and authorization algorithms, file upload and delete algorithms, and data retrieval algorithms.

Figure 5. Class Diagram for the USOCRS

In the image above, several classes are showing the entities and components which are in the USOCRS. The authentication and authorization algorithms are used to ensure that only authorized users within the organization ecosystem can access the system resources. The system uses a token-based authentication mechanism (OAuth2), where a user logs in using their credentials or via Azure Active Directory (AAD), and a token is received from the authentication external service that is used for following requests to the API. SoC engineers and users can obtain access tokens by authenticating with their credentials. These access tokens are then used to authorize access to protected resources within the RestAPI service.

The RestAPI service exposes certain endpoints that allow SoC engineers to post and delete SoC verification reports and for users to retrieve information about the system's progress such as test plans and coverages. These endpoints are implemented using the FastAPI framework, which provides automatic data validation of incoming requests with Pydantic and generation of interactive documentation with the swagger open documentation. The system also uses a multipart form data upload mechanism, which allows users to upload files into the system using the exposed endpoints.

The verification reports, user information extracted from the token received from the OAuth2 authentication, and other relevant data are stored first in the Mongo database to reduce data latency, then the system extracts relevant information into a new PostgreSQL

database. The implementation uses SQLAlchemy, an Object-Relational Mapping (ORM) library, to interact with the database and perform operations such as data retrieval, insertion, update, and deletion.

The necessary error handling parts are implemented to ensure robustness and user-friendly error messages in case of failures or problems with the system. Custom exception classes are used to capture and handle different types of errors, providing informative responses to the users of the system.

The RESTful API is divided into two parts one for the developers (DEV-REPORT) and the other for the user (CLIENT-REPORT). Below are a few lists of exposed endpoints and their responses.

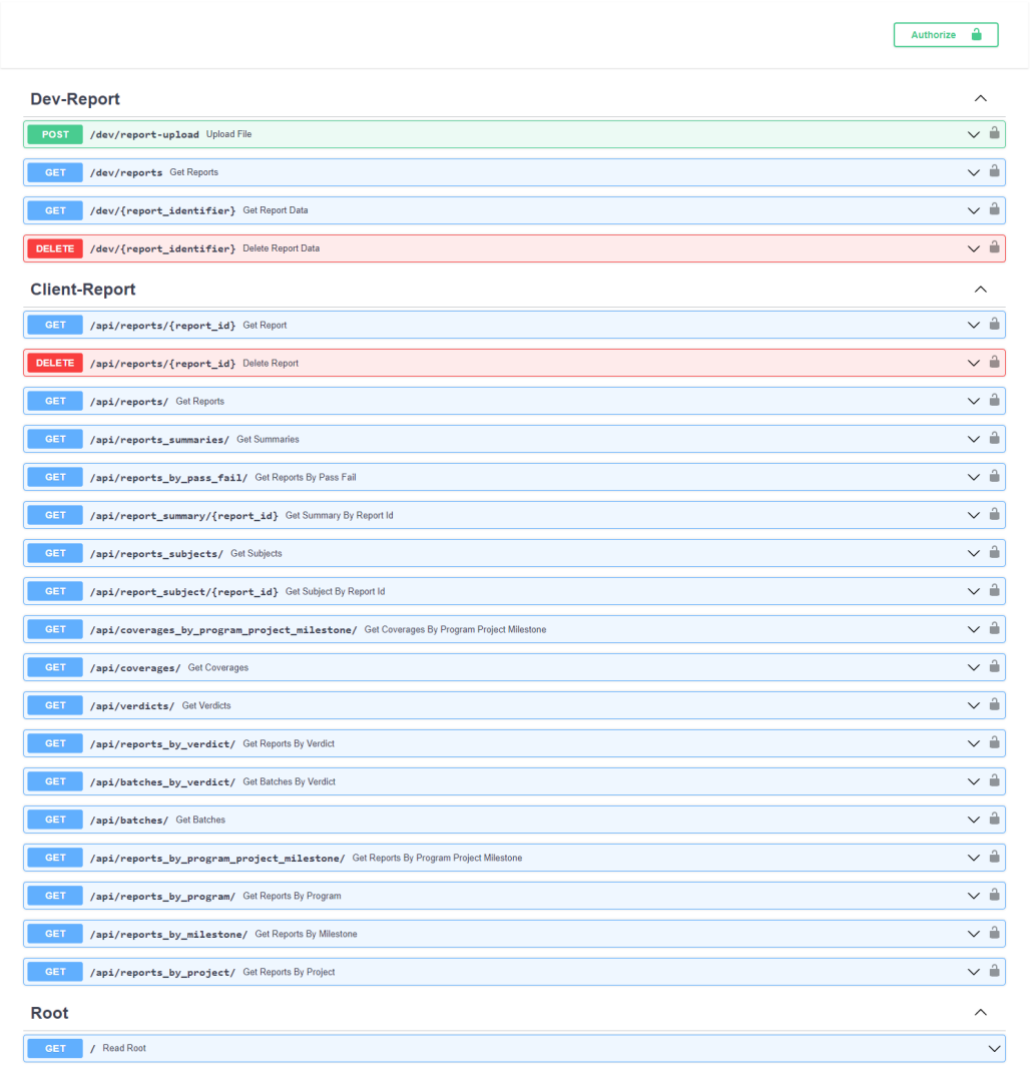

Figure 6. Swagger view of current Endpoints.

Base URL

The base URL assuming the deployment is on the local host for the API is “http://localhost:8000/

Endpoints For the SoC Developer

Endpoint: ‘/dev/report upload

Method: POST

Description: Uploads an SoC verification report for a System-on-Chip (SoC) device by the developer.

Request Body:

- `file`: The verification report file (multipart/form-data)

Response:

- 200 OK: The verification report was successfully uploaded.
    - Body: INTEGER of inserted ID or IDs of the report.
- 400 Bad Request: The request is invalid or missing the required parameters.
- 401 Unauthorized: The user is not authenticated.
- 403 Forbidden: The user is authenticated but does not have the necessary permissions to upload the report.
- 500 Internal Server Error: An unexpected error occurred during the upload process.

Endpoint: `/dev /{report_ identifier}`

Method: GET

Description: Retrieves the details of a specific verification report.

Parameters:

- `report_id`: The ID of the verification report

Response:

- 200 OK: The verification report details were successfully retrieved.
    - Body: JSON object containing the verification report details.
- 401 Unauthorized: The user is not authenticated.
- 403 Forbidden: The user is authenticated but does not have the necessary permissions to access the report.
- 404 Not Found: The specified report ID does not exist.
- 500 Internal Server Error: An unexpected error occurred during the retrieval process.

Endpoint: `/dev/reports`

Method: GET

Description: Retrieves a list of all verification reports.

Response:

- 200 OK: The list of verification reports was successfully retrieved.

    - Body: JSON array containing the verification report objects.
- 401 Unauthorized: The user is not authenticated.
- 403 Forbidden: The user is authenticated but does not have the necessary permissions to access the reports.
- 500 Internal Server Error: An unexpected error occurred during the retrieval process.

Endpoint: `/dev /{report_identifier} `

Method: DELETE

Description: Deletes a specific verification report.

Parameters:

- `report_id`: The ID of the verification report to delete.

Response:

- 204 No Content: The verification report was successfully deleted.

- 401 Unauthorized: The user is not authenticated.

- 403 Forbidden: The user is authenticated but does not have the necessary permissions to delete the report.

- 404 Not Found: The specified report ID does not exist.

- 500 Internal Server Error: An unexpected error occurred during the deletion process.

Endpoints For the User

These reports are generated from the extracted report data which are stored in the Postgres database.

Endpoint: `/API /{report_ id}`

Method: GET

Description: Retrieves the details of a specific verification report found by report id.

Parameters:

- `report_id`: The ID of the verification report to search.

Response:

- 200 OK: The verification report details were successfully retrieved.
    - Body: JSON object containing the verification report details.
- 401 Unauthorized: The user is not authenticated.
- 403 Forbidden: The user is authenticated but does not have the necessary permissions to access the report.
- 404 Not Found: The specified report ID does not exist.

- 500 Internal Server Error: An unexpected error occurred during the retrieval process.

Endpoint: `/API/reports`

Method: GET

Description: Retrieves a list of all verification reports from the system.

Response:

- 200 OK: The list of verification reports was successfully retrieved.
    - Body: JSON array containing the verification report objects.
- 401 Unauthorized: The user is not authenticated.
- 403 Forbidden: The user is authenticated but does not have the necessary permissions to access the reports.
- 500 Internal Server Error: An unexpected error occurred during the retrieval process.

Endpoint: `/API /{report_id} `

Method: DELETE

Description: Deletes a specific verification report find by id.

Parameters:

- `report_id`: The ID of the verification report to delete.

Response:

- 204 No Content: The verification report was successfully deleted.
- 401 Unauthorized: The user is not authenticated.
- 403 Forbidden: The user is authenticated but does not have the necessary permissions to delete the report.
- 404 Not Found: The specified report ID does not exist.
- 500 Internal Server Error: An unexpected error occurred during the deletion process.

5.3 Testing

Testing is an important aspect of software development, and the USOCRS is no exception to this. The system uses several testing frameworks to ensure that it is functioning correctly some in the implementation stage at the time of the writing. These testing frameworks such as pytest, TestClient from FASTAPI, and Swagger UI provided by the FastAPI docs for open documentation standards.

Swagger UI is a web-based interface that allows developers to explore and test the RestAPI service. It provides a user-friendly interface for making requests to the API and inspecting the responses. More information on swagger has already been discussed in the previous chapters.

5.4 Deployment

The USOCRS is deployed using a containerization approach. Docker is used to containerize the application, which provides a lightweight and portable environment for running and sharing the service. Docker allows the service to be easily deployed on different deployment environments, such as development, testing, and production, without the need for additional configuration for the application.

6. System Evaluation

In this chapter, the evaluation of the USOCRS using a research methodology that aligns with Design Science Research (DSR) principles were looked into. Our main objective is to develop and validate a solution, the RESTAPI service, which addresses the challenges faced by SoC engineers when managing and reporting verification reports. Throughout this chapter, we outline a step-by-step process that incorporates relevant concepts from existing literature, highlighting the significance of each stage and the integration of appropriate research frameworks.

The research methodology employed follows the DSR approach introduced by Hevner et al., (2004). DSR is a problem-solving methodology that aims to create and validate an artifact to tackle specific organizational challenges. The methodology used here focuses on four key elements of DSR: design as an artifact, research contribution, research rigor, and communication of the research. These elements form the foundation of our research methodology.

Design as an artifact involves developing a tangible solution, in the case here, the USOCRS, which effectively addresses the management and reporting issues surrounding verification reports in SoC teams in the organization ecosystem. The research contribution of the USOCRS project aims to make significant advancements in both theoretical and practical aspects of SoC verification. To ensure the credibility and reliability of the research outcome, rigorous research methods throughout the development process were used. This includes conducting thorough literature reviews, following systematic design processes, and employing appropriate evaluation techniques to validate the effectiveness of the artifact. Effective communication of the research findings is also essential for sharing knowledge and facilitating the adoption of the developed artifact i.e., The USOCRS.

6.1 Evaluation Methodology

In the research methodology, it includes the Framework for Evaluation in Design Science Research (FEDS) proposed by (Venable, Pries-Heje, & Baskerville, 2014). FEDS provides a structured framework for evaluating design science research, focusing on two dimensions: functional purpose and model. For the USOCRS project, the functional purpose evaluation goals are derived from the specific requirements of the RESTAPI service, encompassing aspects such as performance, usability, efficiency, accuracy, and user satisfaction. In terms of the model dimension, a combination of quantitative and qualitative evaluation methods was applied to it. This includes measuring various metrics related to upload/transfer time, validation accuracy, data extraction efficiency, and visualization effectiveness. Additionally, the collection of user surveys and feedback to gauge user satisfaction and usability was employed.

The development of our USOCRS follows a systematic design science research methodology, guided by the insights provided by Hevner et al. (2004). The process involves several steps, including problem identification, research question formulation, design solution, artifact development, solution evaluation, and results communication. By adhering to this methodology, it ensures that the developed artifact effectively addresses the identified problem and fulfills the specific requirements of SoC teams. The valuable guidance from relevant references, such as Dresch et al. (2014) and Venable et al. (2014), helps in conducting rigorous research and applying suitable evaluation strategies.

6.2 Data Collection Process and Task

The data collection process conducted for the USOCRS makes use of feedback forms and tasks created using Google Forms, which was designed to gather feedback from engineers and stakeholders regarding the current state of the USOCRS. The data collection process aimed to assess user authentication, report submission and validation, system assessment and reporting, error handling, and overall usability. This review provides a comprehensive analysis of the data collection process and evaluates the effectiveness of the tasks assigned.

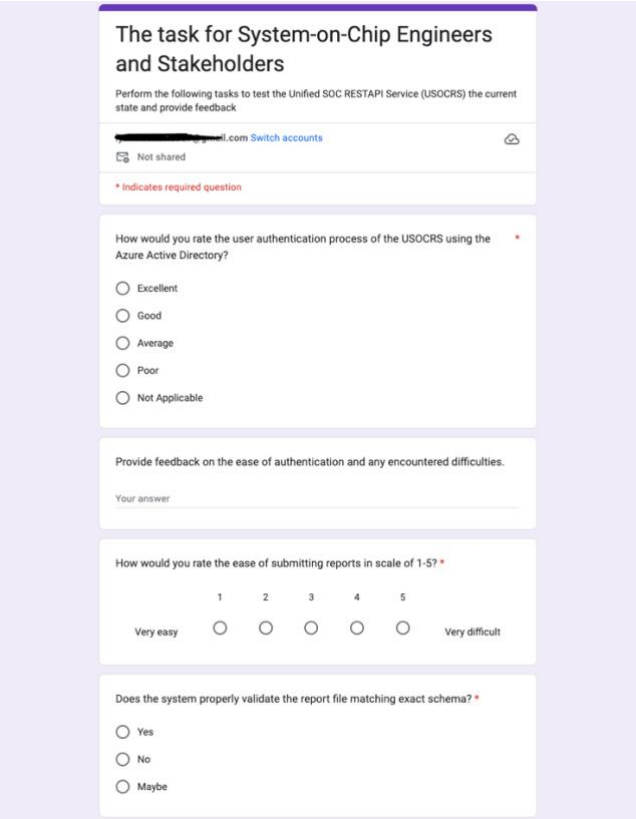

Figure 7. Feedback Form sample.

6.2.1 Test Organization for the USOCRS

The test organization for the USOCRS followed a structured approach to evaluate the system's performance, usability, and effectiveness in meeting the required functionality. The evaluation process involved manual testing by engineers within the organization, who were provided with specific tasks and a feedback form to gather their experiences and insights. The diagram below shows the various task that will be performed by the users of the system.

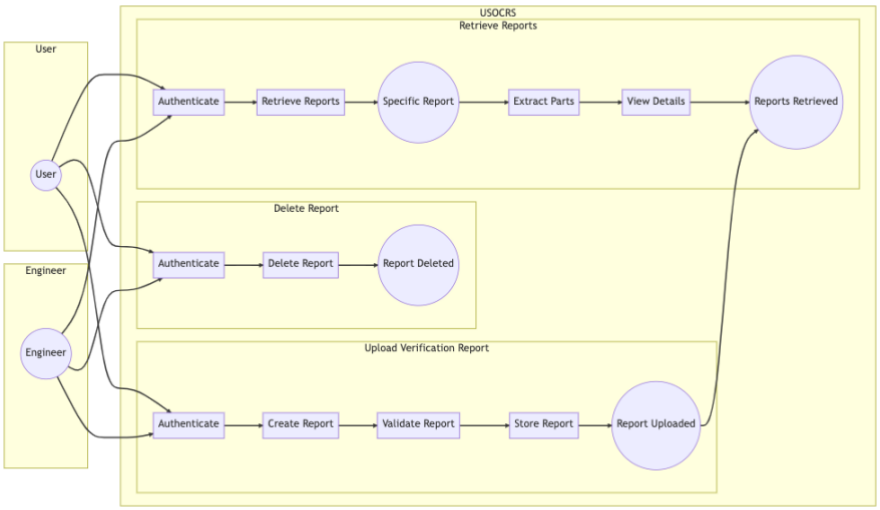

Figure 8. The task for users of USOCRS.

The diagram above shows the process flow of the USOCRS based on the use case identified and discussed in Chapter 4. It shows the process of receiving and processing SoC verification reports, the submission of verification reports by SoC engineers, and the examination of system verification progress by users.

Test Setup

The test process began by contacting approximately five engineers who were assigned to evaluate the USOCRS. Each engineer received an email containing the necessary information, including the feedback form, individual tasks to perform, and a link to clone the repository containing the USOCRS codebase. This ensured that the engineers had access to the system and could set it up on their local machines.

Task Instructions

The engineers were provided with clear instructions on how to set up the USOCRS system on their local machines. This included steps to authenticate with their work credentials, clone the repository, and start up the system. The instructions guided them on performing specific tasks, interacting with the RESTAPI service, and evaluating different functionalities of the system.

Engineer Tasks

From the engineer's perspective, the following tasks were assigned to evaluate the USOCRS:

- Authenticating: The engineers were required to log in using their work credentials to test the system's authentication process. This task aimed to validate if the engineers could successfully authenticate with their credentials.
- Posting SoC Verification Report: The engineers were tasked with creating and posting a verification report using the designated endpoint for report upload. This task tested the functionality of the API endpoint responsible for receiving and storing the report, ensuring it worked properly.
- Retrieving Reports: The engineers were instructed to retrieve the verification reports stored in the database using the designated USOCRS endpoint. This task evaluated the system's ability to accurately retrieve and present the stored data to the engineers.
- Retrieve Specific Report: The engineers were asked to test the system's capability to retrieve a specific report by calling the REST API. This task aimed to validate if the API could successfully retrieve a specific report based on the provided report ID.
- Deleting Reports: The engineers were assigned the task of deleting a verification report using the RESTAPI service. This task ensured that the USOCRS could remove a report from the database when requested, testing the system's deletion functionality.

User Tasks

From the user's perspective, the following tasks were assigned to evaluate the USOCRS:

- Authenticating: Similar to the engineer's task, the users were required to authenticate with their work credentials to test the system's authentication process.
- Retrieving Reports: The users were instructed to retrieve the verification reports stored in the database using the designated USOCRS endpoint. This task assessed the system's ability to accurately retrieve and present the stored data to the users.
- Retrieve Specific Report: The users were asked to test the system's capability to retrieve a specific report by calling the REST API. This task validated the system's ability to retrieve a specific report based on the provided report ID.
- Viewing Specific Parts of the Report: The users were tasked with accessing specific parts of the verification report, such as successes, failures, or test coverages. This task aimed to evaluate the RESTAPI service's capability to provide filtered and targeted information to users.

Feedback Collection

To collect feedback and insights from the engineers and users, a feedback form created using Google Forms was utilized. The feedback form consisted of specific questions related to the tasks assigned, system functionality, and user experience. The questions covered various areas, including user authentication, report submission and validation, system assessment and reporting, error handling, and overall usability. The engineers and users were requested to fill out the feedback form, providing their ratings, feedback, and suggestions for improvement.

The data collected through the feedback form and the completion of assigned tasks provided valuable insights into the performance, usability, and effectiveness of the USOCRS. The evaluation process allowed engineers and users to test the system's

functionalities, interact with the RESTAPI service using tools like Swagger UI and the terminal, and provide feedback based on their experiences. This feedback will aid in identifying areas of improvement, addressing any issues or difficulties encountered, and refining the USOCRS to better meet the required functionality and user satisfaction of the API's intended purpose.

6.3 Results

The evaluation of the USOCRS yielded valuable findings regarding its functionality, usability, and security. Five participants provided feedback, shedding light on the system's strengths, weaknesses, and areas for improvement.

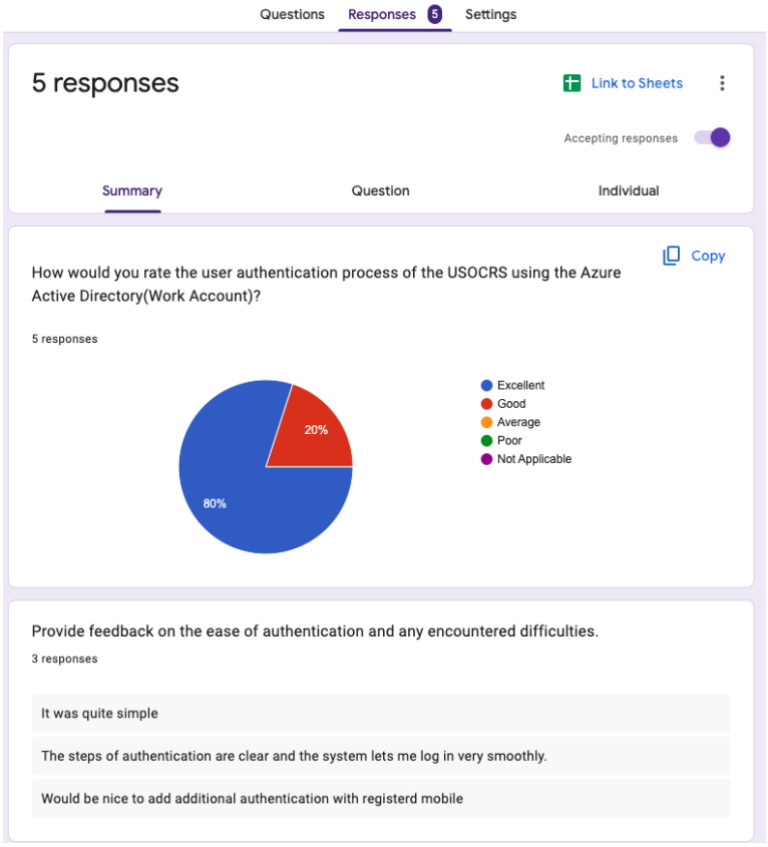

Figure 9. Sample of the responses received.

In the evaluation of the USOCRS, the incorporation of a feedback mechanism, as illustrated in Figure 9, plays a crucial role. This feedback mechanism serves as a valuable component for assessing the system's effectiveness and gathering practical insights that can be utilized to refine and enhance the system further. By adopting this approach, the study aligns with the design science research paradigm proposed by Hevner et al. (2004), contributing to the progress of knowledge within the realm of information systems research.

User authentication using Azure Active Directory (Work Account) received positive ratings, with 80% of participants considering it excellent and the remaining 20% rating it as good. Users found the authentication process easy to follow, experiencing no difficulties during login. One participant suggested incorporating additional authentication using registered mobile devices.

Submitting reports were deemed straightforward, with 80% of participants giving it the highest rating of 5. No challenges were encountered during the submission process. Regarding report validation, 80% of participants confirmed that the system properly validated reports according to the exact schema. However, a few participants expressed uncertainty (20%) in their responses.

Participants unanimously agreed (100%) that the system stored and retrieved reports correctly when using the GET methods. Similarly, 80% of participants acknowledged that the retrieved reports contained all the necessary fields. The system's performance in storing and retrieving reports was rated as excellent by all participants (100%), indicating efficient and fast operations. The system's assessment and reporting features were found to be highly useful by participants, assisting in identifying areas for improvement and streamlining processes.

Participants appreciated the system's error-handling mechanism, as all respondents (100%) confirmed its proper functioning. The error messages were considered clear and informative, aiding users in understanding the causes of process failures. When evaluating the overall usability and effectiveness of the USOCRS, 60% of participants rated it 9 out of 10, while the remaining 40% gave it a perfect score of 10 out of 10.

The system's security received a perfect rating of 10 out of 10 from all participants, indicating a high level of protection for user authentication and data storage.

Based on their testing experience, participants provided valuable feedback and suggestions. Improving the user interface, integrating with Power BI or a graphical interface, and developing a mobile app for inexperienced users were among the recommendations. Additionally, participants expressed the need for a larger dataset to test the system's performance in real-world scenarios. Below are some of the comments obtained from the user.

User Authentication Process

The user authentication process using Azure Active Directory (Work Account) received positive ratings from participants. They found the authentication steps clear and straightforward, enabling smooth logins. One participant specifically mentioned,

> *"The steps of authentication are clear and the system lets me log in very smoothly." Another participant suggested, "Would be nice to add additional authentication with registered mobile."*

Ease of Submitting Reports

Participants expressed that submitting reports through the USOCRS was an easy and seamless process. They appreciated the user-friendly interface and clear guidance provided. One participant mentioned,

> *"It was quite simple. The system lets me submit reports smoothly."*

Report Validation

The majority of participants confirmed that the USOCRS properly validated report files according to the exact schema. One participant specifically stated,

> *"Yes, the system properly validates the report file matching the exact schema."*

However, a participant expressed some uncertainty, saying,

> *"Maybe the system does not properly validate the report file."*

Further investigation is needed to address any potential issues.

Report Storage and Retrieval

Participants unanimously agreed that the USOCRS effectively stored and retrieved reports using the GET methods. They praised the system for maintaining the integrity of stored reports and ensuring the completeness of retrieved reports. One participant remarked,

> *"Yes, the system properly stores the report when calling the GET methods to retrieve the reports."*

System Performance

The system's performance in storing and retrieving reports received excellent ratings from all participants. They commended the USOCRS for its efficient handling of large volumes of data. One participant mentioned,

> *"Excellent! I can easily store and retrieve documents, and it's fast as well."*

Error Handling Mechanism

Participants found the system's error handling mechanism effective, with clear and understandable error messages. One participant mentioned,

> *"It was easy to decipher. The error messages are clear and help users understand the reason for process failures."*

Another participant suggested,

> *"But a more detailed error message is needed for the front end. For example, duplicate report only contains a server error and returns a duplicate report error, but the user needs to be informed where these duplicates are in the report if possible."*

Usability and Effectiveness

Participants highly rated the overall usability and effectiveness of the USOCRS in facilitating SOC verification reporting. They acknowledged its role in streamlining processes, reducing bottlenecks, and enhancing communication channels. One participant commented,

> *"It's very useful. It helps identify areas where improvements can be made, such as streamlining processes, reducing bottlenecks, or enhancing communication channels."*

Security

Participants expressed a high level of confidence in the system's security, particularly regarding user authentication using the Azure Active Directory and data storage. They gave the system a perfect score in terms of security, reflecting its ability to protect user credentials and ensure data security within the organization's ecosystem.

Recommendations for Improvement

Based on the feedback provided by participants, several recommendations for improvement were identified. These include developing a user-friendly interface, incorporating additional authentication methods, integrating with data visualization tools like Power BI, considering a mobile application for inexperienced users, increasing the dataset size for testing, and providing more detailed error messages. Participants shared their thoughts on potential improvements, such as:

> *"Providing a UI would be helpful," "Frontend or integration with Power BI will be great," and "Though this is manual testing, a fully deployed application will be ideal, including a simplified graphical interface or mobile app for inexperienced users who can't set up the system."*

The evaluation of the USOCRS highlighted its effective functionality, usability, and security. Participants provided positive feedback, emphasizing the system's ease of use, excellent performance, and robust security mechanisms. The recommendations offered by participants provide valuable insights for further enhancing the system and addressing areas for improvement. The USOCRS has the potential to streamline SOC verification reporting and contribute to the efficiency of security operations. Continued improvements based on user feedback will ensure the system remains effective and aligned with user requirements and industry standards.

To summarize this section, the evaluation highlighted the USOCRS's strengths, including seamless user authentication, easy report submission, proper validation and storage of reports, excellent performance, and a high level of security. Participants found the assessment and reporting features valuable. The feedback and suggestions will contribute to further enhancing the system, ensuring it becomes even more effective and user-friendly for SOC verification reporting.

7. Discussion

This chapter provides a comprehensive discussion of the findings obtained from evaluating the User Security Operations Center Reporting System (USOCRS). The evaluation aimed to evaluate various areas of the system, including user authentication, report submission, validation, storage, retrieval, performance, error handling, usability, and security. The analysis includes the feedback received from engineers, including their comments and suggestions, to understand the strengths and weaknesses of the USOCRS and provide recommendations for improvement.

The evaluation of the USOCRS has resulted in insightful results regarding the system's functionality and usability. These findings highlight its strengths, including easy user authentication, easy report submission, proper validation and storage of reports, excellent performance, and a high level of security. The feedback and suggestions provided by participants are instrumental in further enhancing the system, ensuring its effectiveness and user-friendliness in SOC verification reporting.

The analysis of the results confirms that the USOCRS effectively addresses the original research question of designing, developing, and validating a unified SOC reporting system. Its features, such as user authentication using Azure Active Directory, straightforward report submission, proper validation and storage of reports, efficient performance, and robust security mechanisms, directly improve the efficiency and effectiveness of SOC validation report transfers, utilization, and interpretation in SOC device testing.

The positive ratings and feedback from participants regarding the system's functionality, usability, and security reaffirm its ability to streamline processes, reduce bottlenecks, and enhance communication channels in SOC verification reporting. These results also align with the design science research paradigm, proposed by Hevner et al. (2004), contributing to the advancement of knowledge in information systems research.

Furthermore, the recommendations provided by participants offer valuable insights for future improvements. Suggestions such as developing a user-friendly interface, incorporating additional authentication methods, integrating with data visualization tools like Power BI, and providing more detailed error messages can further enhance the system's functionality and user experience. The future work section highlights potential areas for improvement, including scalability, implementing additional features, optimizing performance, resource management using Kubernetes, developing mobile and web applications for easy report access, and viewing, and incorporating automated testing methodologies.

By addressing these recommendations and focusing on future work areas, the USOCRS can continue to evolve and meet the evolving needs of engineers and organizations. This will further improve the efficiency and effectiveness of SOC validation report transfers, utilization, and interpretation, ultimately resolving the original research question and contributing to the advancement of SOC device testing.

7.1 Future Work

The implementation and evaluation of the USOCRS have provided valuable insights into its capabilities and performance. This section investigates the potential areas for future

work and improvement, based on the discussions and requirements identified in previous chapters. Despite the positive evaluation results, there are areas where the USOCRS can be further improved. The future work encompasses enhancing scalability, implementing additional features, and optimizing performance to meet the evolving needs of engineers and organizations in general.

To accommodate the increasing needs of the organization and ensure the system can handle large volumes of requests without having problems with its performance, scalability, and performance optimization are crucial parts of such a system. One approach which is much of interest in modern software development is to explore the adoption of microservices architecture (Fowler, 2014) to decompose the system into smaller, loosely coupled services that can be independently maintained and scaled. This would allow for more efficient resource utilization and better performance of the USOCRS.

Furthermore, adding caching mechanisms, such as Redis (Carlson, 2013) can significantly improve response times by storing data that are always accessed by users in memory. Caching can improve the load on the database and reduce the overall response time for subsequent requests when the APIs are called. As the USOCRS expands and attracts more users, it becomes imperative to strengthen the authentication mechanisms. One potential future enhancement is the already integrated OAuth2 (D. Hardt, 2012) a widely adopted industry standard for secure authentication and authorization. By implementing OAuth2, the system can delegate the responsibility of user authentication to trusted identity providers, such as Microsoft Azure Active Directory, ensuring robust security and faultless integration with other applications.

To ensure scalability and efficient resource management of the USOCRS, integrating the system with Kubernetes can be useful in the long run. Kubernetes is a powerful and widely used container orchestration platform that can help manage and scale the system's components based on requirements or desired state. By using the power of Kubernetes, the system can on its own dynamically allocate resources, handle load balancing, and provide high availability to users. This integration can enhance the system's scalability, allowing it to handle large amounts of requests without having poor performance (Brendan Burns, 2019). One of the potential future directions for the USOCRS is to develop mobile and web applications for users to easily access and view the report information. These applications can provide a user-friendly interface, allowing engineers and stakeholders to efficiently navigate and interact with the system efficiently. Integration with front-end applications can provide real-time updates and notifications, enhancing the user experience and improving collaboration within the organization. The testing of the USOCRS is conducted manually at the time being. Adding automated testing methodologies into the system workflow can in improve the efficiency and reliability of the testing process of the system. Automated tests can be developed to cover many test scenarios, including positive and negative test cases, performance testing, and stress testing. Integration with popular testing frameworks and tools, such as Pytest or Selenium, can make more efficient the testing process and provide continuous feedback on the system's performance and functionality.

8. Conclusion

The main purpose of this report was to design, develop, and validate the USOCRS. Its purpose is to improve the efficiency and effectiveness of transferring, utilizing, and interpreting SoC verification reports. The project focuses on developing a unified REST API service for SOC reports by utilizing various tools and technologies available within and outside the organization.

To achieve this, a design science research methodology was followed, involving several steps. Initially, a thorough literature review was conducted to explore existing approaches and technologies related to SOC verification reporting, automation, data visualization, and API development. The literature review provides useful insights into the current state of the field and identified gaps that required further investigation which is then accessed in the development of the USOCRS. Next, a system design and implementation plan were worked out, which comprises the use of technologies such as FASTAPI, SQL and NoSQL databases, Azure Active Directory for authentication, and Nokia Cloud. The verification Toolbox was used for SoC report validation. Finally, the system was manually tested, and user satisfaction with its functionality was evaluated through feedback forms.

Although the project is still undergoing development to meet all the forthcoming essential requirements, the findings of this study demonstrate the successful creation and implementation of the USOCRS. This service offers a unified platform for SOC engineers to securely upload, validate, store, and retrieve verification reports when needed. It facilitates efficient communication between users and the API, providing easy access to crucial information such as successes, failures, and test coverage derived from submitted SoC verification reports. The USOCRS automates and standardizes the SOC verification reporting process, eliminating the need for manual and repetitive tasks performed by SOC engineers in their day-to-day activities. It increases productivity and establishes a secure and reliable platform for storing and accessing verification reports when needed. By integrating various tools and technologies like FASTAPI, SQL and NoSQL databases, Azure Active Directory, and Nokia Cloud, the project offers a comprehensive solution for SOC verification reporting. The utilization of the Verification Toolbox by the organization ensures that the submitted reports adhere to the required specifications of the SOC verification report schema standardized and implemented within the organization.

One of the significant advantages of the USOCRS is its ability to improve the efficiency and effectiveness of SOC verification reporting. It achieves this by streamlining the submission process, reducing latency through optimized data storage, and providing meaningful data extraction techniques and analysis as endpoints. The system enhances progress monitoring and facilitates informed decision-making through a comprehensive analysis of the report data.

It's important to acknowledge potential biases in the USOCRS. The evaluation was conducted with a limited number of participants, and their feedback may not fully represent the diverse range of users and scenarios in SOC device testing. Additionally, participants' familiarity and prior experience with similar systems may have influenced their perspectives and ratings. Therefore, it is crucial to consider these biases when interpreting the evaluation results and implementing further improvements to the USOCRS.

To mitigate these biases, future evaluations of the USOCRS should involve a larger and more diverse group of participants, including individuals with varying levels of expertise and experiences with SOC reporting systems. This approach would provide a more objective and well-rounded assessment of the system's performance, usability, and effectiveness.

Additionally, biases might exist in the selection and implementation of tools and technologies used in developing the USOCRS. Personal preferences, availability, or organizational constraints could influence these choices, limiting the system's compatibility with alternative options. These biases may impact the scalability, performance, and overall effectiveness of the USOCRS. Therefore, it is essential to carefully consider and evaluate different alternatives to ensure the chosen tools and technologies align with industry standards.

To conclude, this project successfully addresses the challenges associated with SOC verification reporting by designing, developing, and implementing the USOCRS. The USOCRS significantly improves the efficiency and effectiveness of verification report transfers, utilization, and interpretation. Integrating various cutting-edge tools and technologies provides SOC engineers with a unified and secure platform for uploading, validating, storing, and retrieving verification reports. This project contributes to the field by offering a comprehensive solution and paves the way for future research and development.

References

Alexander S. Gillis, B. B. (2023, 03). What is MongoDB? Retrieved from TechTarget: https://www.techtarget.com/searchdatamanagement/definition/MongoDB

AltexSoft. (2021, 07 23). Functional and Nonfunctional Requirements: Specification and Types. Retrieved from AltexSoft: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/

Atlassian. (2023). What is version control? Retrieved from Atlassian: https://www.atlassian.com/git/tutorials/what-is-version-control

Bakar, M. A., Ismail, S., Idris, S., & Shukur, Z. (2015). seMeja API Design Based on CRUD+N Concept. Journal of Computer Science.

Bansal, P., & Ouda, A. (2022). Study on Integration of FastAPI and Machine Learning for Continuous Authentication of Behavioral Biometrics. 022 International Symposium on Networks, Computers, and Communications (ISNCC), 1-6.

Bitbucket. (2023). What is Git? Retrieved from Atlassian: https://www.atlassian.com/git/tutorials/what-is-git

Brendan Burns, J. B. (2019). Kubernetes Up & Running: Dive into the Future of Infrastructure. Beijing, Boston, Farnham, Sebastopol, Tokyo: O'Reilly Media, Inc.

Carlson, J. (2013). Redis in Action. Shelter Island, NY: Manning Publications Co.

Chakravarthi, V. S. (2019). A Practical Approach to VLSI System on Chip (SoC) Design. Switzerland: Springer, Cham.

Coenrad, F. J. (2020). Electronic Design Automation tools for superconducting circuits. Journal of Physics: Conference Series.

D. Hardt, E. (2012, 10). The OAuth 2.0 Authorization Framework. Retrieved from Internet Engineering Task Force (IETF): https://datatracker.ietf.org/doc/html/rfc6749

de Oliveira VF, P. M. (2022). SQL and NoSQL Databases in the Context of Industry 4.0. Machines, 1-20.

Deshpande Anil. (2008, 7 14). Verification of IP Core-Based SoCs. Retrieved from Design And Reuse: https://www.design-reuse.com/articles/18032/verification-ip-core-soc.html

Docker Inc. (2023). Docker overview. Retrieved from Docker: https://docs.docker.com/get-started/overview/

Dresch, A. L. (2014). Design Science Research. Design Science Research, 67-102.

Driessen, V. (2010, 01 05). A successful Git branching model. Retrieved from Nvie: https://nvie.com/posts/a-successful-git-branching-model/

Eito-Brun, R., & Amescua-Seco, A. (2018, 06). Automation of Quality Reports in the Aerospace Industry. IEEE Transactions on Professional Communication, pp. 166-177.

Fowler, M. (2014). Microservices: a definition of this new architectural term. Retrieved from martinfowler: https://martinfowler.com/articles/microservices.html

GitLab. (2023). What is CI/CD? Retrieved from GitLab: https://about.gitlab.com/topics/ci-cd/

Gupta, L. (2022, 4 7). What is REST? Retrieved from Restful API: What is REST

He, J., Guo, X., Zhao, Y., & Jin, Y. (2020). Formal verification for SoC security. Frontiers in Hardware Security and Trust; Theory, design, and Practice.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS Quarterly, 75-105.

IBM Inc. (2023). What is a NoSQL database? Retrieved from What is a NoSQL database?: https://www.ibm.com/topics/nosql-databases

Im, Y.-H., Yim, S.-H., & Kim, J. B. (2012). A web service for automated IP/SoC verification using computers on network. International SoC Design Conference (ISOCC), 398-401.

Ishtiaq, A., Khan, M. U., Ali, S. Z.-e.-Z., Habib, K., Samer, S., & Hafeez, E. (2021). A review of system-on-chip (soc) applications in the Internet of Things (IoT) and medical. International conference on advances in mechanical engineering, 1-10.

JFrog Team. (2020, 12 20). What Is Artifactory? | JFrog. Retrieved from JFrog: https://jfrog.com/blog/what-is-artifactory-jfrog/

Kaur, J., Kaur, J., Kapoor, S., & Singh, H. (2021). Design & development of customizable web API for interoperability of antimicrobial resistance data. Scientific Reports.

Kumar, M. S., Suchismita, S., & SK, M. (2015). Test bench automation to overcome the verification challenges of SOC Interconnect. 015 International Conference on Man and Machine Interfacing (MAMI), Bhubaneswar, India, 1-4.

Len Bass, P. C. (2013). Quality Attributes. In P. C. Len Bass, Software Architecture in Practice (pp. 78-207). Upper Saddle River, NJ • Boston • Indianapolis • San Francisco New York • Toronto • Montreal • London • Munich • Paris • Madrid Capetown • Sydney • Tokyo • Singapore • Mexico City: Pearson Education.

MacMillen, D., Butts, M., Camposano, R., Hill, D., & Williams, T. W. (2000, 12 12). An Industrial View of Electronic Design Automation. IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, pp. 1428-1448.

Matinolli, I. (2016). Designing, implementing, and evaluating a database for a software testing team. University of Oulu Department of Information Processing Science. Retrieved from http://jultika.oulu.fi/files/nbnfioulu-201612033204.pdf

Méré, M., Jouault, F., Pallardy, L., & Perdriau, R. (2022). Feedback on the formal verification of UML models in an industrial context: the case of a smart device life cycle management system. Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems.

Microsoft. (2023, 2 21). What is Azure Active Directory? Retrieved from Azure Active Directory: https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis

Microsoft. (2023, 2 28). What is Conditional Access? Retrieved from Azure Active Directory: https://learn.microsoft.com/en-us/azure/active-directory/conditional-access/overview

Moore, B. G. (1965, April 19). Cramming more components. Electronics magazine.

Oksa, M. (2016). WEB API DEVELOPMENT AND INTEGRATION FOR MICROSERVICE FUNCTIONALITY IN WEB APPLICATIONS. UNIVERSITY OF JYVÄSKYLÄ DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS, 77.

Petcu, D., ciun, C. C., Neagul, M., Lazcanotegui, I., & Rak, M. (2011). Building an Interoperability API for Sky Computing. International Conference on High-Performance Computing & Simulation.

PostgreSQL. (2023). What is PostgreSQL? Retrieved from PostgreSQL: https://www.postgresql.org/about/

Rajkotia, D. (2021, 12 30). Which Python framework is the fastest? Retrieved from DEV Community: https://dev.to/dhruv_rajkotia/which-python-framework-is-fastest-2fgo

Rauf, I., Vistbakk, I., & Troubitsyna, E. (2018). Formal Verification of Stateful Services with REST APIs Using Event-B. IEEE International Conference on Web Services (ICWS), 131-138.

Ray, S., Peeters, E., Tehranipoor, M. M., & Bhunia, S. (2017). System-on-Chip Platform Security Assurance: Architecture and Validation. Proceedings of the IEEE, 21-37.

Rayanagoudar, S., Hampannavar, P. S., Pujari, J., & Parvati, V. (2018, June). Enhancement of CI/CD Pipelines with Jenkins BlueOcean. International Journal of Computer Sciences and Engineering, pp. 1048-1053.

RedHat Inc. (2020, 05 8). What is a REST API? Retrieved from RedHat: Understanding APIs: https://www.redhat.com/en/topics/api/what-is-a-rest-api

RedHat Inc. (2022, 5 11). What is CI/CD? Retrieved from RehHat: https://www.redhat.com/en/topics/devops/what-is-ci-cd

Rouse, M. (2020, 3 30). Atomicity Consistency Isolation Durability. Retrieved from Techopedia: https://www.techopedia.com/definition/23949/atomicity-consistency-isolation-durability-acid-database-management-system

Siedle, J. (2015). System Requirements. Retrieved from University of Missouri–St. Louis: https://www.umsl.edu/~sauterv/analysis/F2015/System%20Requirements.html.htm

Sivakumar, P. R. (2020, 03 17). IP vs SoC Verification - Maven Silicon. Retrieved from Maven Silicon: https://www.maven-silicon.com/blog/ip-vs-soc-verification/?amp=1

SmartBear Software. (2023). OpenAPI Specification. Retrieved from Swagger: https://swagger.io/specification/v3/

Srikant Kumar Mohanty, S. S. (2015). Test bench automation to overcome the verification challenges of SOC Interconnect. 2015 International Conference on Man and Machine Interfacing (MAMI), Bhubaneswar, India, 1-4.

Suad Kajtazovic, C. S. (n.d.). AUTOMATIC GENERATION OF A VERIFICATION PLATFORM FOR HETEROGENEOUS SYSTEM DESIGNS. Graz: Institute for Technical Informatics Graz University of Technology.

Synopsys Inc. (2023). What is EDA (Electronic Design Automation)? Retrieved from Synopsys: https://www.synopsys.com/glossary/what-is-electronic-design-automation.html

Synopsys. (2023). Verification IP | Synopsys Verification. Retrieved from Synopsys: https://www.synopsys.com/verification/verification-ip.html

Valentina., A. (2015). Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery. Armenise, V. (2015). Continuous Delivery with Jenkins: Jenkins Solutions to Implement Continuous Delivery. 2015 IEEE/ACM 3rd International Workshop on Release Engineering. doi:10.1109/releng.2015.19.

Venable, J., Pries-Heje, J., & Baskerville, R. (2014). FEDS: a Framework for Evaluation in Design Science Research. European Journal of Information Systems, 77-89.

Widiyanto, A. D., Anindito, B., & Azam, M. N. (2020, 11 3). 20 Implementation of Docker and Continuous Integration / Continuous Delivery for Management Information System Development. International Journal of Electrical Engineering and Information Technology, pp. 20-24.

Wilson, R. (2010, 06 4). Bringing automation to debugging in the SoC verification struggle. Retrieved from EDN: https://www.edn.com/bringing-automation-to-debugging-in-the-soc-verification-struggle/

Wisal Khan, T. K. (2022). SQL and NoSQL Databases Software architectures performance analysis and assessments-A Systematic Literature review. arXiv preprint arXiv:2209.06977.

Zhou, W., Li, L., Luo, M., & Chou, W. (2014). REST API Design Patterns for SDN Northbound API. 28th International Conference on Advanced Information Networking and Applications Workshops.

Zhu, Y., & Gao, H. (2014). A Novel Approach to Generate the Property for Web Service Verification from Threat-Driven Model. Applied Mathematics & Information Sciences, 8, 657-664.

Appendix A. Sample of Task and Responses

Task

The task for System-on-Chip Engineers and Stakeholders

All tests are recommended to be performed via the generated API documentation on http://localhost:8000/docs or Postman. Test via curl is possible but requires a manual adding of authorization args with an authorization token.

Evaluate the USOCRS in its current state using by completing the task below.

1. Authenticate using Microsoft Azure Active Directory (AAD) authentication:
    a. Access the API "http://localhost:8000/docs" and authenticate using your Microsoft account via the **Authorize** option.
    b. You will get a pop-up page that will request your authorization, select the checkbox at the bottom before the login button and leave the rest empty then click login.
2. Verification Report Submission:
    a. Submit verification reports file through the API POST "http://localhost:8000/dev/report-upload" using the web. Take note of the IDs from the response, which will be needed later.
    b. Ensure the reports are in the specified format (e.g., JSON or Text) and comply with the Verification Schema.
    c. Verify the effectiveness of the validation process using the Verification Toolbox and report any errors or issues identified. Try uploading a random text or JSON file to verify the system properly stores only the SoC verification report based on schema specification.
3. Validation Process and Storage:
    a. Get all reports through the API GET "http://localhost:8000/dev/reports" to verify reports the report was uploaded successfully using the API.
    b. Using the ID from the upload response or uuid from the report get a report through the API GET "http://localhost:8000/dev/report-id" to verify reports the report was uploaded successfully using the API.
4. Verification Report Deletion:
    a. Delete uploaded verification reports using the provided methods (command line, terminal, or web) and report the ease of deletion.
    b. Using the ID from the report delete a report through the API DELETE "http://localhost:8000/dev/report-id" to delete a verification report that was uploaded using the API.
5. System Assessment and Reporting:
    a. Access the API "http://localhost:8000/client/" client resources to view the extracted data by calling different endpoints.
6. Error Handling:
    a. Try calling any of the API endpoints without authentication and note the error messages.
    b. Give a random ID on the GET or DELETE methods of the API and note the error messages.

Play around with other endpoints which are available on the API documentation and provide comments or feedback where necessary.

Please complete the tasks and provide detailed feedback using the provided feedback form questions. Additionally, feel free to provide any further comments or suggestions for the improvement and development of the system.

Thank you for participating.

Responses

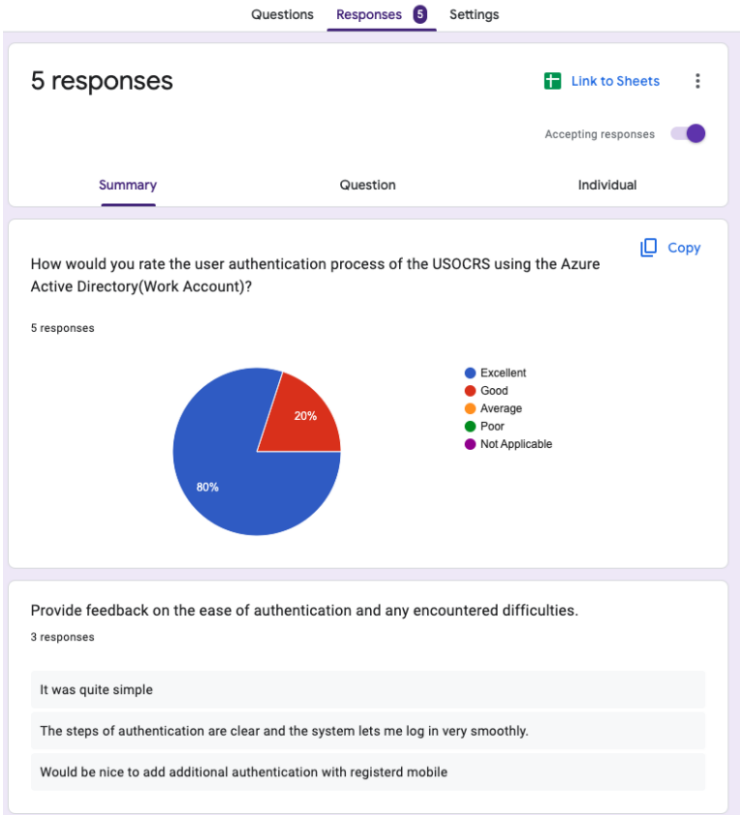

Figure 10. Sample of the responses received.

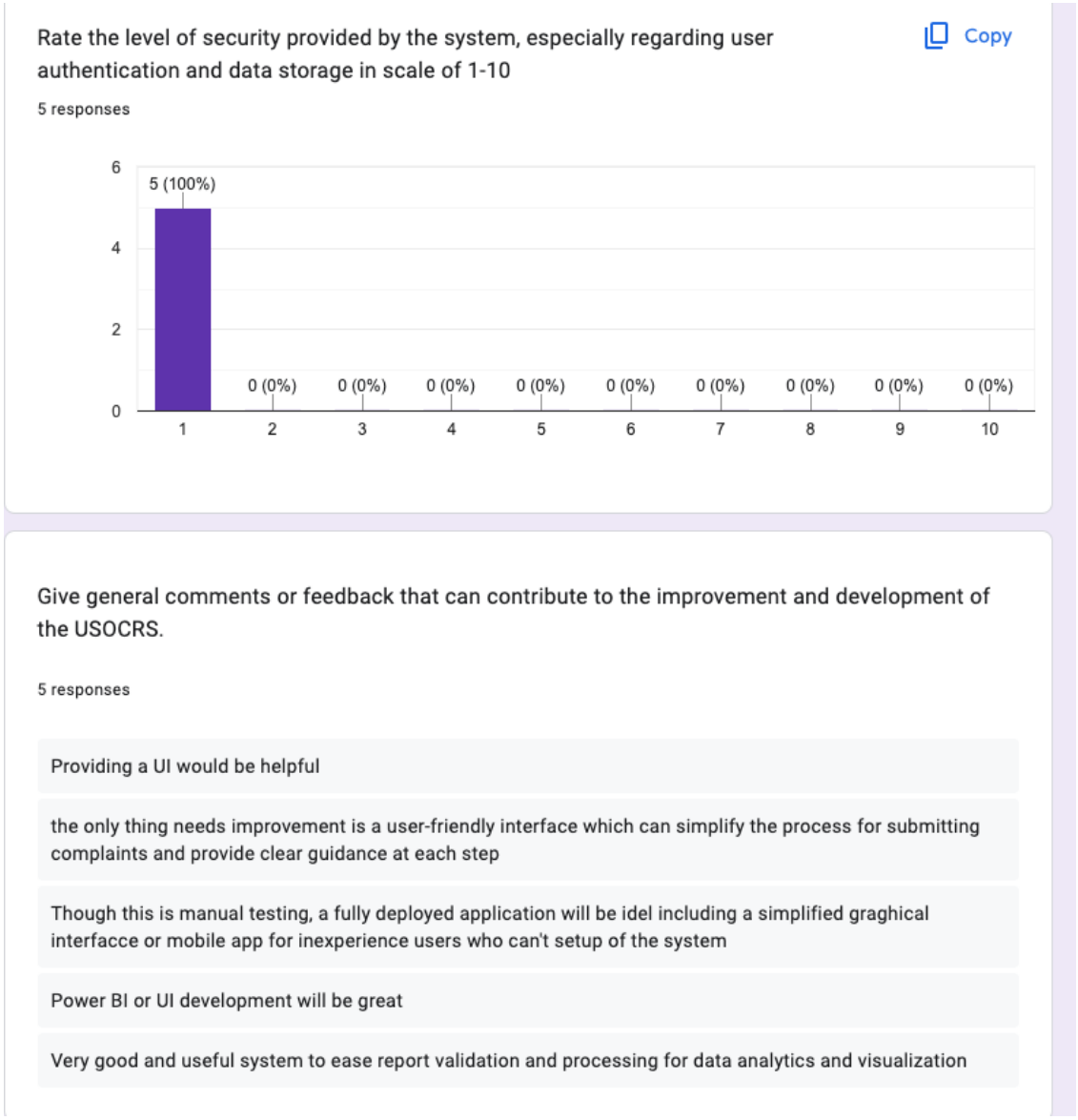

Figure 11. Sample of the responses received.

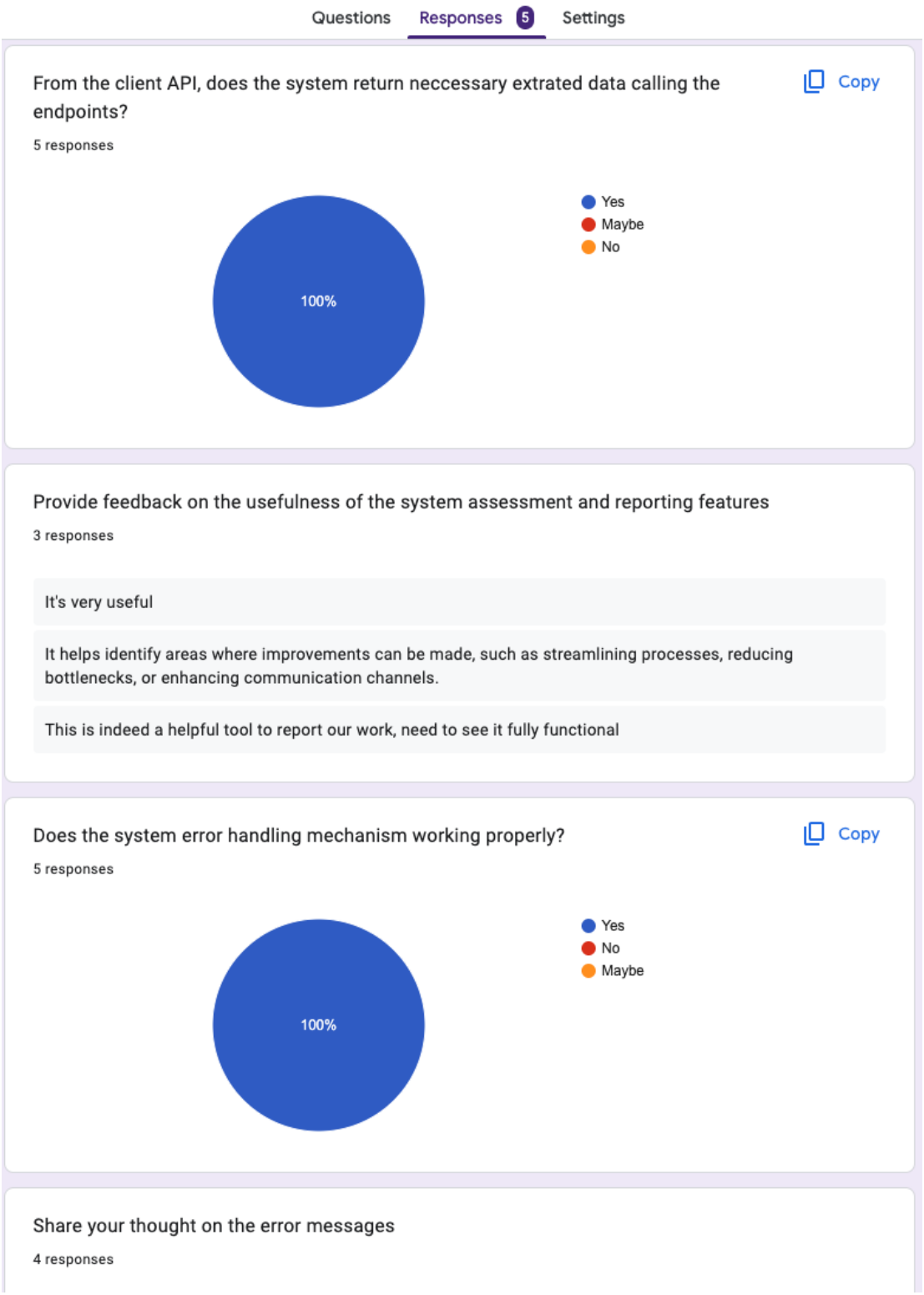

Figure 12. Sample of the responses received.