Knowledge Graph-Based Traceability and Impact Analysis of Requirements in Software Development

MSc in Computer Science

Fatimah Oyindamola Aileru

Staffordshire University

A dissertation submitted in partial fulfilment of the requirements of Staffordshire University for the degree of Master of Science

May 2024.

ACKNOWLEDGEMENT

Table of Contents

List of Abbreviations

SD:	Software Development
SDLC:	Software Development Life Cycle
FR:	Functional Requirements
NFR:	Non-Functional Requirements
KG:	Knowledge Graph
BERT:	Bidirectional Encoder Representations from Transformers
NLP:	Natural Language Processing
AI:	Artificial Intelligence
IR:	Information Retrieval
CSE:	Continuous Software Engineering
API:	Application Programming Interface
SRS:	Software Requirement Specification
RDG:	Requirement Dependency Graph
WS4J:	WordNet Similarity for Java
ASD:	Agile Software Development
MANoR:	Managing Non-functional Requirements

Abstract

Due to the neglect of Non-Functional Requirements (NFRs) in agile software development, and methodological deficiency in the existing studies on software development, this research explores the application of Knowledge Graphs (KGs) for traceability and impact analysis in software development, highlighting their potential to enhance the management of both functional and non-functional requirements. The study introduces a methodical approach to improve requirement traceability and impact analysis, ultimately facilitating better decision-making and increasing development efficiency. The research underscores the critical role of requirements in guiding various phases of the Software Development Life Cycle (SDLC) and introduces a basic web application for interaction with the Knowledge Graph. This dissertation also discusses the integration of machine learning techniques to automate the classification and analysis of requirements, which reduces manual overhead and enhances the precision of data input into the Knowledge Graph. These enhancements enable stakeholders to gain detailed insights into potential impacts of changes to requirements, which aids in strategic project management. Future research directions include expanding the application of KGs to different software development methodologies and integrating KGs with existing project management tools to enhance their utility across the software development industry.

Keywords: Knowledge Graphs, Software Development, Traceability, Impact Analysis, Software Development Life Cycle, Machine Learning, Requirements Management, Functional Requirements, Non-Functional Requirements, Graph database.

CHAPTER 1 INTRODUCTION

1.1 Introduction

Software development (SD) is the process of forming and maintaining applications through programming, documentation, testing, and problem fixes, culminating in a software product. From inception to completion, the creation of the intended software follows a predetermined and organized process (Werner et al., 2020). It is also an intricate and dynamic field that relies profoundly on the precision and clarity of requirements to navigate the complexities of its lifecycle. Recent research has underscored the pivotal role of requirements in guiding various phases of the Software Development Life Cycle (SDLC) (Werner et al., 2020). Hence, a software product must go through several processes in the SDLC to be developed and designed successfully. Nonetheless, different models offer distinct project life cycles and methodologies such as waterfall and agile and are dependent on several variables, such as team size, budget, schedule, and risk management.

In software development, defining all a software's needs at once is challenging because of the constantly shifting contexts and environments in which it is utilised (García-López et al., 2019). Therefore, requirements might change as a result of a variety of factors, including peer competition, regular system upgrades, divergent developer perspectives, shifting market conditions, and consumer demands (Saputri & Lee, 2021; Haider, et.al., 2019). The foundation upon which software will operate is defined by software requirements, making them an essential component of software development (Nurbojatmiko & Wibowo, 2021). These include a variety of elements such as processes, performance, controls, inputs, and outputs. Consequently, the software will operate as intended and offer a satisfying user experience if the software requirements are defined correctly.

However, issues identified in software development, such as unanticipated changes and ineffective handling of NFRs, emphasize the need for adept traceability techniques (Firdaus Arbain et al., 2020). These challenges stem from the dynamic nature of software projects where requirements frequently evolve due to technological advancements, market fluctuations, or changes in user expectations. Specifically, unanticipated changes can disrupt the development process, leading to missed deadlines, budget overruns, and products that fail to meet user needs. Furthermore, the neglect of NFRs compounds the complexity of software development, necessitating frameworks for requirement management (Rasheed et al., 2021). The oversight of NFRs often leads to software that may perform its intended functions but fails in areas critical to user satisfaction and operational stability, such as security, performance, and usability. This was reinforced by Luangwirya (2022), which detects conflicts within NFRs using conceptual graphs in billing systems, and a related study that introduces a requirement dependency graph modelling (Priyadi et al., 2019). This approach not only highlighted the hidden conflicts but also showcased how representations could aid stakeholders and developers in understanding and resolving these conflicts effectively. According to research stakeholders face challenges in understanding NFRs due to software project dynamics and evolving user needs, which can lead to compromised system performance, security vulnerabilities, and decreased satisfaction (Haindl et al., 2020).

Sequel to the above, this research seeks to explore two core aspects to address the above challenges related to software development and its requirements. First to examine the traceability in software requirements with a focus on the application of knowledge Graph-Based approach. Second to explore the impact analysis of Functional Requirements (FR) and Non-Functional Requirements (NFRs) in software development. With the scope and focus, it is expected that the study will make a novel contribution to software development and expand the frontier of knowledge on the subject matter.

1.2 Motivation

Recent industry trends indicate that despite the advancement in software development methodologies, the management of requirements remains a challenge, often leading to overlooked or inadequately managed requirements that significantly impact the quality and functionality of the final software products. A survey by the Standish Group reports that over 52% of software projects face challenges due to poor handling of NFRs (KÖSE, 2019). This mismanagement results in higher costs and decreased customer satisfaction, highlighting a critical need for innovative solutions. Knowledge Graphs (KGs) offer a promising approach by providing a holistic view of how both functional and non-functional requirements are related and impacts one another, ensuring comprehensive management, and understanding across all software requirements. The increasing complexity and demand for high-quality, efficient software projects underscore the importance of such innovative tools to streamline requirement management processes, improve project efficiency, reduce risks, and enhance overall software quality and stakeholder satisfaction.

1.3 Aim

The research aims to explore how knowledge graphs can be utilised for traceability and impact analysis of requirements in software development. It seeks to investigate their role in establishing relationships between functional and non-functional requirements and assessing the implications of changes of the requirements on software development. Through this exploration, the study aims to advance methodologies for requirement management, enhancing the efficiency and adaptability of software development processes.

1.4 Objectives

- To identify and classify entities and relationships of functional and non-functional requirements from textual dataset using machine learning techniques.

- To define nodes, entity relationships, properties and establish traceability links using knowledge graph for functional and non-functional requirements relationships.
- To utilise the graph model for impact analysis on requirement relationships, providing insights into the effects of changes.
- To design, implement and test a web application that interact with the knowledge graph on the change impact of requirements.
- To critically evaluate the knowledge graph -based approach in traceability, impact analysis of software requirements.

1.5 Deliverables

- A model that classifies functional and non-functional requirements embedded in a textual dataset containing both entities and relationships.
- A graph data model representing software requirements, defining the nodes, edges, and properties.
- A detailed report describing the implementation and functionality of the KG, including its ability to enhance requirement traceability and impact analysis.
- A validated web application that utilises the KG for real-time impact analysis and traceability between functional and non-functional requirements.

CHAPTER 2 LITERATURE REVIEW

2.1 Software Development and Software Development Lifecycle

Software development is a vital process for creating applications that fulfil user needs that encompass multiple stages: design, coding, testing, and maintenance. This iterative process thrives on the flexibility to adapt as technologies advance and as user and business requirements evolve (Güncan & Onay Durdu, 2021). Hussain et al. (2019), stated that quality software is characterised by its user-centric design, reliable performance, maintainability, scalability, and security. Such software not only meets users' demands but also offers a competitive edge by optimising resources and adhering to time and budget constraints. The software development life cycle (SDLC) provides a structured pathway for software creation, ensuring projects' effectiveness and success. As seen in Fig. 1, this lifecycle begins with planning, where objectives and resources are outlined, followed by requirements gathering, design, implementation, testing, and maintenance (Pargaonkar, 2023). Kuhlmann et al. (2022) stated that over time, the SDLC has evolved to address development challenges. In the early days of computing, models like waterfall were predominant, where each phase was completed before moving on to the next (Senarath, 2021). With the rise of the Internet and more complex user needs, methodologies such as agile emerged, emphasising continuous integration, deployment, and feedback (Thesing et al., 2021).

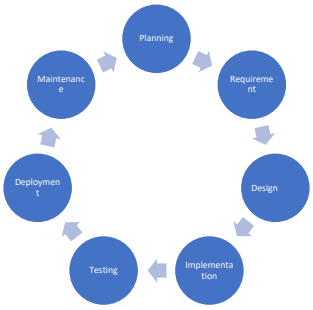


Figure 1: Software Development Lifecycle

2.2 Software Development Methodologies

While the SDLC provides the overarching structure and guidance, the methodologies employed during each phase play a defining role in the efficiency and success of the software development process. Agile methodologies present a dynamic and iterative framework for software development (John & Sharma, 2024). Agile is rooted in a set of principles and practices that prioritise adaptability, collaboration, and incremental progress. The iterative development characteristic of agile allows for the continuous evolution of software through incremental cycles (see Fig.2), enabling more flexibility in responding to changing requirements and customer feedback (Munteanu & Dragos, 2021). This approach contrasts starkly with traditional methods like waterfall, where each phase of development is typically completed before moving on to the next. They often work best in projects where requirements are stable and clearly defined from the outset (Sinha et al., 2021).

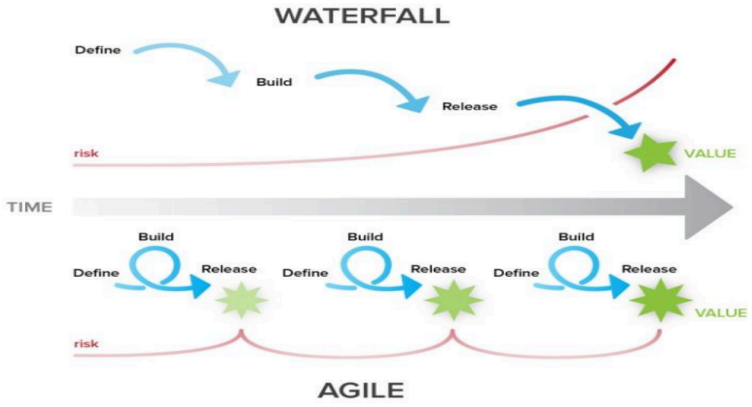


Figure 2: Waterfall and Agile Methodologies Adopted from(Jangra, 2023).

Agile methodologies have gained immense popularity in the software development industry due to their ability to navigate the complexities of modern projects effectively. The emphasis on collaboration within cross-functional teams fosters real-time communication and knowledge sharing, contributing to quicker decision-making processes (Strode et al., 2022).

Waterfall methods may suit projects with well-defined and stable requirements, agile on the other hand excels in environments where change is inevitable, and close collaboration with end-users is paramount. The dynamic nature of agile methodologies aligns seamlessly with the fast-paced and evolving software development requirements, making it a preferred choice for many organisations seeking efficient and responsive development processes (Alhazmi & Huang, 2020).

2.3 Requirement Management

Within the software development life cycle (SDLC), requirements management is fundamental to the process, involving tasks such as requirements gathering, implementation, and management (P. Srivastava, 2023). While requirements management is concerned with conducting thorough analyses and skilfully managing requirements changes, requirements gathering is concerned with the crucial activities of gathering and defining requirements (Alhazmi & Huang, 2020). For ease of communication among stakeholders, requirements documents are typically written in natural language text by analysts working with product owners. Software requirements are generally divided into two categories: functional and non-functional requirements (Shreda & Hanani, 2021).

Requirements take diverse forms and representations, serving as pivotal artefacts that guide the development process (Kusumo et al., 2022). One prevalent representation is the user story, which is central to agile methodologies. User stories provide a concise and user-centric description of a feature, focusing on the "who," "what," and "why" aspects of a requirement (see Fig.3). They are designed to articulate the value a feature brings to the user, stating clearly who the feature is for, what it should do, and why it is needed. The "how" aspect, which details the specific implementation or technical requirements necessary to realise the user story, is

typically captured within the acceptance criteria rather than in the user story itself (Nasiri et al., 2023).

Figure 3: An Example of a User Story.

Acceptance criteria are essential components of user stories, defining the conditions under which a user story is considered complete, and the functionality is deemed acceptable. They provide a detailed and objective checklist that must be met for the story to be closed (Ferreira et al., 2022).

Contrastingly, traditional methodologies like waterfall often rely on comprehensive requirements specification documents. These documents are detailed and structured, offering an exhaustive overview of system functionalities, constraints, and specifications. They provide a static and thorough representation of requirements upfront, aligning with the sequential nature of the waterfall (Nur Fathin Najwa Binti Mustaffa et al., 2021). Use case diagrams provide another perspective on types of requirements, specifically a visual representation of functional requirements. It is used in software and system design to capture the functional interactions between the system and its external entities, such as users or other systems. Use case diagrams are part of the unified modelling language (UML), which is a standard notation for modelling software and system architectures (Hamza & Hammad., 2019).

Beyond these, prototypes and wireframes serve as visual representations, offering stakeholders a tangible preview of the system's interface and functionalities facilitating early feedback and understanding (Deininger et al., 2019). Prototypes, varying from basic outlines to interactive models, allow for the exploration and testing of design concepts and user interactions while wireframes focus on the structural layout, detailing the placement of elements like buttons and menus (Stoeva, 2021). This indicates that recognising these diverse representations of

requirements contributes to comprehending the challenges and strategies in handling requirements across different software development contexts.

2.3.1 Functional and Non-Functional Requirements

Functional requirements (FR) as stated by Kumar & Singh (2022), play a pivotal role in shaping the trajectory of the development process and serve as detailed specifications that articulate the functionalities and features expected from the software system. These requirements serve as a roadmap, guiding the development team in the creation of a system that aligns precisely with user expectations (Bowers et al., 2020). In the traditional waterfall model, functional requirements are often rigidly defined upfront during the planning phase, providing a comprehensive blueprint for the entire project while in agile methodologies, the treatment of FR is more dynamic and iterative (Justin Joseph John & Shailesh Satish Sharma, 2024).

Justin Joseph John & Shailesh Satish Sharma (2024) acknowledges that user needs and project priorities can evolve, emphasising continuous collaboration and flexibility, rather than having an exhaustive set of requirements predetermined. It allows for the gradual development and refinement of functional requirements throughout the project's lifecycle. This approach enables the development team to respond promptly to changing user needs and market dynamics. The significance of functional requirements lies in their ability to bridge the gap between the conceptualisation of the software system and its tangible implementation. By clearly defining what the system is expected to achieve and how it should behave, FRs provide a shared understanding among stakeholders, including developers, testers, and end-users. This shared understanding becomes especially crucial where frequent collaboration and feedback loops are integral to the development process (Kumar & Singh, 2022).

Kumar & Singh (2022) also stated that Non-Functional Requirements (NFR) on the other hand, shape the overall quality and operation of a system. NFRs are distinct from their FRs as they

don't dictate what the system should do but rather outline constraints, qualities, or attributes that set criteria for the system's operation (Rahy & Bass, 2022). These requirements address aspects beyond mere functionalities, encompassing critical elements like performance, security, usability, and maintainability (Shreda & Hanani, 2021).

According to Bowers et al. (2020), the primary focus of NFR is on how well the system performs its functions rather than detailing the functions themselves. Unlike FR, which predominantly addresses "what" the system should do, NFR delves into the "how" by providing criteria for evaluating the system's operational characteristics ensuring that the software not only delivers the expected functionalities but also excels in terms of performance, security, user experience, and long-term maintainability. This indicates that balancing both FR and NFR is crucial for developing a robust software system that not only meets users' explicit needs but also satisfies implicit expectations related to its overall performance and quality. However, Anwer et al. (2019), highlight that 40% of software development challenges stem from issues within the requirement process. One significant challenge lies in the inherent nature of NFRs, which often demand a different approach compared to the more tangible and straightforward FRs.

2.4 Challenges Faced in Non-functional Requirement Management

Stakeholders encounter numerous challenges in the effective management of NFRs, primarily due to difficulties in their early identification, visualisation, and prioritisation. A significant issue is the communication gap between technical teams and non-technical stakeholders over the importance and impact of NFRs. This gap often leads to the underestimation and deprioritisation of NFRs, with stakeholders failing to recognise their critical contribution to the software's value and functionality (Kumar & Singh, 2022).

Viviani et al. (2023) discuss the late identification anti-pattern, which involves ignoring NFRs until production or after identifying failures or vulnerabilities. This can lead to significant challenges, such as architectural changes or code refactoring, often at a high cost. Agile methodology focuses on software development, involving customers throughout the development life cycle, and mainly on functional requirements and associated changes. This leads to the ignorance of NFRs, as both client and developer stakeholders do not feel responsible for defining and implementing NFRs. Additionally, there is no rigorous requirement engineering process to elicit, specify, and prioritise requirements, further causing NFRs to be flawed (Muhammad, 2023).

Jarzebowicz & Sitko (2020), stated that the highest business value first approach is commonly used in agile practices which means that features, enhancements, or user stories believed to contribute the most significant benefit to the business or end-users are developed and implemented before those with less critical impact. However, risks associated with NFRs, such as those related to performance or security, might not be effectively accommodated within a prioritisation framework that predominantly focuses on business value (Behutiye et al., 2020). Also, Behutiye et al. (2020) stated that challenges in managing NFRs have a profound impact on software quality resulting in a product that is challenging to use, prone to failure or vulnerable to security breaches, code refactoring, escalating costs, and delaying project completion. Consequently, the overall quality of the software product is compromised, affecting its competitiveness in the market and long-term sustainability. Therefore, reconciling the prioritisation of NFRs with the project objectives becomes a multifaceted challenge. (Alhaizaey and Al-Mashari, 2023). These challenges underscore the need for tailored approaches that specifically account for the distinct characteristics of NFRs, ensuring that they are not treated as an afterthought in the development process which is crucial to enhance the efficiency of incorporating NFRs, ultimately leading to the successful delivery of software that

not only meets functional expectations but also excels in terms of performance, security, and overall quality.

Research by Kumar & Singh (2022) also recognised the pivotal role of shared understanding in requirements engineering (RE) plays in the success of software projects by investigating in the context of NFRs within small agile organisations practicing continuous software engineering (CSE) and utilising cloud-based platforms. The study identifies factors contributing to the lack of shared understanding, including the fast pace of change, insufficient domain knowledge, and inadequate communication. The research reveals that CSE, often known for its iterative customer feedback cycles, may adversely impact shared understanding of NFRs, leading to rework. However, practices for building shared understanding, shared development standards and effective communication and documentation are highlighted which indicates practical implications for CSE organisations. While providing valuable insights, the study also faces limitations, including relatively small sample size and the focus on specific organisational contexts.

2.5 Related Work Reviewed

This section of the study presents a review of the related work on the phenomena of the study to gain a wider perspective of the subject matter in traceability in software development and the requirement for software development to make a viable contribution to the body of knowledge on software development.

To start with, Rempel and Mader (2018) used a regression technique, the" Multi-level Poisson" regression in their study to examine how requirement traceability completeness affects software quality. The authors revealed that software quality (defect rate) is highly impacted by the level of traceability coherence in three selected studies. It was also discovered that a higher degree of traceability reduces the anticipated rate of defects in the generated program.

Conclusively, the research findings indicate that a robust influence of traceability inclusiveness on the defect rate holds practical significance for software development projects of all stripes, even in the absence of standard or regulatory requirements requiring it.

In another study, Zhang, et.al., (2022) adopted an Information Retrieval (IR) model based on the Vector space model to design and develop an automated software requirements traceability tool. It was established that the program can assess trace linkages in addition to automatically generating them. Furthermore, the experimental results demonstrate that the tool can enhance the effectiveness of requirements traceability link generation and provide better support for software development activities.

Still, on the traceability in software development, Haider, et.al. (2019), adopted an Artificial intelligence (AI) framework to improve requirements prioritization and traceability process for global software development (GSD). Their findings showed that the suggested framework greatly reduced GSD problems by improving requirements prioritization and traceability with minimal human intervention.

Perera (2019) in another study evaluated the existing techniques regarding goal modelling and goal–centric traceability and their applicability towards human values traceability in software. The scholars found both techniques, goal modelling and goal-centric traceability as promising techniques to initiate human values traceability in software. From the case study, it was established that the goal modelling technique potentially improved to trace human values.

Furthermore, Li et al. (2022) proposed STPChain, which is a blockchain-based CSE technique for software traceability and privacy. The authors described the CSE process flow and used smart contracts that were tailored for software development to implement it to preserve software traceability. The use of smart contracts ensures that every input leaves tamper-proof records automatically. Software traceability links are made feasible with the help of the records.

Also, Nurbojatmiko and Wibowo (2021) proposed the use of FR characters to detect NFR attributes. The close relations between NFR and FR are acquired by expanding the NFR measurement to the programming code level and then applying generalization to obtain the FR character. The end product is the ISO/IEC 25023-based NFR characteristics identification method employing FR characters. During the requirements stage, the analyst as well as the programmer can use the FR character to determine the NFR properties from the FR. The study results in a validated NFR Identification Method after testing on several programmers and professionals.

In addition, Sapuri and Lee (2021) adopted a goal-scenario-based approach and a Hierarchical approach to look at software sustainability requirements to improve the requirement process for software development. The scholars established that the approaches provided a practical methodology that was needed to meet the required sustainability indicator and baseline.

In a different study, El-Nemr and Elzanfaly (2018) developed a paradigm for advanced change effect analysis in software development (SD) adopting graph databases as a medium for data storage. From their analysis, it was revealed that graph databases are powerful in analysing the nexus among entities and have also proven to be extremely scalable, which enables it to store all details related to the software model.

Furthermore, Priyadi et al. (2019) proposed a Requirement Dependency Graph (RDG) for software requirements description. Tokenization, stopword removal, and stemming are steps in the extraction process for text preprocessing. Additionally, there is a method called WordNet Similarity for Java (WS4J) that may be used to measure semantic similarity.

Lastly, Behutiye, et al. (2017) utilised ASD (Agile Software Development) to propose procedures for enhancing NFRs documentation. To determine the features, user stories, epics, and acceptance criteria, a case study was used, and practitioners from four different companies were interviewed. It was revealed that the constraints of NFR documentation methods, the loss

of NFR traceability, and the difficulty new developers have understanding NFRs when they join the team are the obstacles in ASD. The proposed suggestions reflect the miscellany and variety of the NFRs in terms of various representation artefacts when documenting and proposing them, depending on their extent and details. In further research by Sherif et al. (2023), the MANoR (Managing Non-functional Requirements) framework was proposed to address the challenges associated with handling NFRs in agile software development. The framework systematically guides the project through key phases, offering a structured approach to recommendation, elicitation, analysis, documentation, and validation of NFRs. MANoR's conflict resolution mechanism, facilitated by a detailed conflict matrix, stands out as a noteworthy contribution, assisting stakeholders in comprehending the impacts of conflicting requirements and fostering negotiation for consensus. The framework emphasises the importance of documentation, particularly in agile environments where a tendency exists to minimise documentation efforts. MANoR ensures that both functional and non-functional requirements are meticulously documented in the product backlog as user stories, enhancing transparency and reducing the likelihood of misinterpretations. Furthermore, the validation phase involving client reviews aligns expectations with documented requirements, minimising discrepancies and reducing the likelihood of rework.

The summary of the above-related works is shown in Table 2.1 for a further critical analysis to identify the grey areas and contribute to the body of knowledge on software development.

Table 1; Summary of Related Works for Critical Analysis

Authors & Years	**Nature of Study**	**Major Approach**	**Limitations**
Rempel and Mader (2018)	Examine how requirement traceability	Multi-level Poisson Regression Analysis	Only the requirement of traceability on software quality was considered.

	completeness affects Software Quality		Management of requirements is not considered.
Zhang, et.al. (2022)	Design and develop an Automated Software Requirements Traceability Tool	Information Retrieval Model based on Vector Space Model	The vector space model does not take user preferences or feedback into consideration, nor does it consider the structure, synonyms, or semantics of the terms.
Haider, et.al., (2019)	Proposed a framework to improve requirements prioritization and traceability process for global software development	Artificial Intelligent Techniques	The traceability process targeted on the global software development. Other aspects such as management of NFRs are not considered.
Perera (2019)	Evaluated the existing techniques and their applicability to trace human values in software	Goal Modeling and Goal -centric traceability	The goal has limitations because it only represents an imagined environmental scenario rather than a real one.

Li, et.al. (2022)	Proposed a framework which is a blockchain technology to provide software traceability and granular privacy	STPChain, a Crowdsource Software Engineering (CSE) method	The study is limited in context as it focused only on software traceability and did not consider the management of NFRs.
Nurbojatmiko,& Wibowo (2021)	Proposed a framework to detect the Non-functional Requirements (NFR) nexus between software quality	FR characters to identify NFR attributes	Reliance on subjective interpretations and the potential for biases could lead to inconsistencies in NFR identification, possibly affecting the alignment of software quality
Sapuri and Lee (2021)	Looked at software sustainability requirements for improving the requirement process for software development	Goal – Scenario-Based Approach & Hierarchical Approach	The goal scenario–based Approach presents several difficulties in practice. For instance, the goal process does not reflect the actual situation but an idealized environmental one. Also, the Hierarchical Approach

			cannot extremely be carried over to the SD discipline
El-Nemr and Elzanfaly (2018)	Developed a system to help software developers advance change impact analysis	Graph Database	With this approach, data inconsistencies can quickly reduce their usefulness.
Priyadi et al., (2019)	Looked at the Software Requirements Specification (SRS) documents	Requirement Dependency Graph (RDG)	The approach is limited by its singular focus on Software Requirements Specification (SRS) documents and an exclusive emphasis on traceability. This narrow scope limits its utility in agile and dynamic software development contexts.
Behutiye, et.al., (2017)	Proposed framework for enhancing NFR documentation	ASD (Agile Software Development)	The study is narrow in scope by proposing a framework for NFR documentation. The study did not consider the management of NFRs

2.6 Gaps and Critical Analysis of Related Works

As contained in Table 2.1, the review of the existing related works on the themes of the study revealed that studies abound on traceability in software development and requirements in software development. Despite the volume of the studies, three gaps were identified, which are methodological, contextual gaps and content gaps. Out of the identified gaps, the current study contributed to the body of knowledge by focusing on two gaps. First, regarding the methodological gaps, the existing studies proposed various methods and approaches to examine traceability in software development. For instance, Rempel and Mader (2018) adopted a multi-level Poisson regression analysis. In another study, Zhang, et.al., (2022) designed and developed Information Retrieval Model based on the Vector Space model. Further, Haider, et.al., (2019) proposed Artificial Intelligent Techniques. Also, Perera (2019) adopted goal modelling and goal-centric traceability. In addition, Li, et.al., (2022) adopted STPchain in their study to look at traceability.

From the above analysis, a shred of evidence revealed that there is a methodological gap in the existing studies, as none of the studies proposed or used Knowledge Graph-Based traceability of software requirement in software development. To fill the identified gap, the current study proposed and used Knowledge Graph-based traceability of software requirement in software development. With that, it is expected that the outcome of the study will provide new insights into traceability in software development.

In addition to the conceptual gap was another gap identified in the existing studies. The conceptual gaps imply that there is a considerable amount of existing research on a specific context. Evidently from the review, the existing studies on requirements in software development either considered Non-Functional Requirements using the characters of Functional Requirements (Nurnojatmiko & Wibodo, 2021); uses software requirement specification (SRS) which is peculiar to waterfall (Priyadi, et.al., 2019; Sapuri and Lee, 2021)

and how to document NFRs (Behutiye, et.al., 2017). However, it can be established that the researchers did not consider managing Non-Functional Requirements particularly in an agile environment. Therefore, the gap suggested that further study becomes necessary in that direction. To bridge the identified contextual gap, the current study explored the traceability and impact analysis of Functional and Non-Functional Requirements (NFRs) in software development in an agile environment where user stories are used. Focusing on NFRs expands the frontier of knowledge on requirements in software development enabling the efficient management of NFR.

2.7 Traceability and Impact Analysis

The traceability of requirements and their impact analysis is of paramount importance to ensure the successful development and maintenance of software systems. It establishes a clear and documented link between various phases of the software development lifecycle, from initial requirements gathering to the final implementation (Johansson & Ntoukolis, 2021). Ruiz et al. (2023) highlight that practitioners recognise the substantial benefits of traceability, which reflects an understanding of traceability's value in enhancing project management, ensuring compliance with requirements, and facilitating the maintenance and evolution of complex software systems. Impact analysis, closely tied to traceability, involves evaluating the potential effects of alterations to requirements on the overall system. By systematically tracing and analysing requirements, development teams can enhance transparency, manage project risks, and make informed decisions throughout the software development process. This contributes to improved communication, reduced errors, and increased overall project success (Aung et al., 2020).

Impact analysis and traceability are two fundamental aspects of software development that play a crucial role in maintaining the integrity and quality of software projects as they evolve. Impact

analysis is a systematic approach designed to evaluate and identify the potential consequences of proposed changes within the software system (Johansson & Ntoukolis, 2021). This process is essential for understanding how modifications to requirements, design elements, or code, can affect the overall functionality and performance of the software. Kretsou et al. (2021) emphasise the importance of thorough impact analysis in mitigating risks associated with software changes, ensuring that any alterations do not inadvertently introduce errors or degrade the system's performance. On the other hand, traceability enables effective impact analysis by establishing clear, navigable links among various software artefacts. Charalampidou et al. (2020) highlight the significance of traceability in the software development lifecycle, noting that it enables developers and project managers to track the lineage of each artefact and understand how they interrelate.

Stakeholders in software development encompass a broad range of individuals and groups with vested interests in the project's outcome, including project managers, developers, clients, end-users, and quality assurance teams (Dragos, 2021). One of the primary challenges stakeholders' faces is managing the dynamic nature of software requirements. Requirements often evolve due to changing business needs, technological advancements, or feedback from end-users. These changes can have far-reaching implications on the project, affecting everything from the software's architecture to its codebase and test plans (Haindl et al., 2020). This shows that without a clear understanding of the impact of these changes, stakeholders may find themselves facing delays, cost overruns, and a product that fails to meet the intended objectives.

Research by Johansson & Ntoukolis (2021), explores the effectiveness of different visualisations in software traceability such as graphs and matrixes, being more commonly used and perceived as advantageous. These visualisations play a crucial role in enhancing stakeholders' understanding of complex relationships and dependencies within requirements engineering. Stakeholders, including project managers and developers, benefit from intuitive

representations that facilitate the interpretation of intricate traceability information and impact analysis. This contributes to improved decision-making, communication, and collaboration among diverse project participants promoting a shared understanding and fostering adaptability in response to changing requirements, ultimately enhancing the overall efficiency of the software development process (Pimentel & Lencastre, 2020).

According to research, it's evident that the traceability of requirements in software development projects can be managed using Knowledge Graphs (KGs). Knowledge Graphs offer a dynamic and interconnected framework that captures the complex relationships between various software artefacts, including requirements, design elements, code, and test cases (Radhakrishnan et al., 2023).

2.8 Knowledge Graphs

Hao et al. (2021) stated that the knowledge graph concept aimed to refine search engine intelligence was launched by Google in 2012 and has emerged across diverse domains within software development, as demonstrated by research. Its applications span a wide spectrum, encompassing intelligent software development (Chen et al., 2023), which can provide a structured and coherent model for problem-solving using natural language processing (NLP), facilitating knowledge organisation for the reuse of expert knowledge in software development (Sun et al., 2019) (Wang et al., 2020) and as a recommender system for personal computers, demonstrating the ability to recommend components based on functional requirements (Meedeniya et al., 2019). Traditional approaches such as dependency analysis, traceability matrices, simulation and modelling, and expert judgment offer valuable insights into the consequences of modifications. However, these approaches often have limitations in capturing the complex interdependencies and semantic relationships present in software systems. In contrast KG is a graph that encodes entities and their relationships that maps out the

connections between abstract concepts and entities, thereby transforming traditional search by using semantic links for reasoning and graphically organising information for easy navigation (Lee et al., 2023).

As highlighted in Hao et al. (2021), Neo4j is a graph database that can be used for building and managing knowledge graphs to represent and query the relationships between entities. The graph model employed by Neo4j is inherently designed to map out complex networks of entities and their interrelations. This makes it well-suited for crafting knowledge graphs that require a nuanced understanding of how different data points relate to one another (Bhattacharyya & Professor, 2020). This flexibility in knowledge graph applications is vital, where the data structure can evolve, reflecting the dynamic nature of knowledge itself. To build knowledge graphs for software requirements, extracting these requirements is essential, with machine language techniques being crucial for translating complex requirements into structured forms for accurate representation and analysis (Sonbol et al., 2022).

Machine learning techniques have increasingly been used in the extraction of software requirements, offering a bridge between the often-ambiguous language of stakeholders and the precise needs of software development (Hamza & Hammad, 2019). The systematic review by Sonbol et al. (2022) highlights a pivotal shift in requirements management towards leveraging advanced machine learning techniques, particularly embedding representations like BERT, to enhance the automation of requirements. It emphasised the effectiveness of these techniques in capturing the subtle differences in requirements, thereby significantly improving the accuracy of tasks such as requirement analysis and quality assessment. In the study by Kici et al. (2021), it was leveraged through a streamlined variant of BERT for the classification of SRS. The study showcased proficiency in multi-class text classification and emphasised its potential to improve the automation of the SRS design process by accurately identifying requirement types, priorities, and severities. By processing user stories, it distils complex requirements into

structured formats that are easier for the Neo4j graph database to establish the traceability and impact analysis of the requirements (Choega,2023).

2.9 Summary

This section presents the summary of the review and reiterates the identified gaps. The chapter contained various sections and subsections to vividly explain, evaluate and synthesize the issues and opinions on software development with a focus on the traceability and impact of requirements in software development. The study's contributions to knowledge emerged from methodological gaps and contextual gaps identified. Premised on the identified gaps, it proposed and used Knowledge Graph-Based Traceability and Impact Analysis of Requirements in Software Development. In furtherance, the next chapter presents and discusses the methods used to achieve the set objectives.

2.10 Analysis of Problem/ Improvement

The comprehensive examination of the literature concerning software development reveals significant methodological and contextual gaps highlighted in section 2.5. These gaps underscore a vital need for more dynamic and technology-driven approaches, particularly in agile environments where requirements frequently change. To address these gaps, the current research integrates a Knowledge Graph-based system within a web application framework designed to enhance the traceability and management of both Functional and Non-Functional Requirements (NFRs). Earlier studies in software development traceability employed various methodologies, from statistical models to goal modelling. However, none leveraged a Knowledge Graph-based system integrated within a web application, which introduces innovative possibilities for real-time data processing and interactive user engagement.

The proposed web application utilizes a Knowledge Graph underpinned by Neo4j (Hao et al. 2021), a graph database that supports complex queries with minimal latency. This backend

architecture enables enhanced traceability of software requirements. The system allows for immediate updates and interactions, making it an excellent tool for agile settings where quick adaptation and feedback are crucial. It offers significant benefits such as real-time impact analysis, which allows users to immediately understand how changes to one requirement might affect others. Additionally, the web application's user-friendly interface facilitates direct interaction with the Knowledge Graph, enabling stakeholders to query relationships without needing advanced technical skills, effectively democratising data accessibility. This dynamic feature is essential for maintaining the integrity and functionality of evolving software.

Incorporating Knowledge Graphs enables the system to automatically establish nodes and relationships based on the classification of data inputs. As requirements are uploaded and processed, the system identifies key entities and their interdependencies, structuring them as nodes and edges within the graph. This structured representation helps visualize complex data relationships intuitively and supports more efficient querying and updating of data as software requirements evolve.

The inclusion of Knowledge Graphs in software development also aids in overcoming some of the traditional challenges associated with requirement traceability and analysis. By providing a structured yet flexible framework to manage the relationships between different requirements, Knowledge Graphs help in pinpointing dependencies and potential conflicts early in the development cycle. This proactive management helps in avoiding costly revisions and delays later in the project lifecycle, thereby enhancing overall project efficiency and stakeholder satisfaction.

In summary, the proposed web application with its Knowledge Graph-based backend represents a significant advancement in the field of software development traceability and impact analysis. It addresses the critical gaps identified in existing methodologies by providing a tool that supports dynamic requirement management and enhances decision-making

processes within agile environments. This system stands as a testament to the potential of integrating advanced data technologies like Knowledge Graphs into software development practices, promising substantial improvements in how requirements are traced, analysed, and managed across the lifecycle of a project.

CHAPTER 3. THE RESEARCH METHODS

3.1 Introduction

In continuation from the previous chapter which contained the review of the related works on the themes of the study and the identified gaps, this chapter contains the research methods chosen to achieve the set research objectives to bridge the identified gaps and make a novel contribution to software development. The methodology describes the process through which research objectives would be achieved in gathering specific and relevant information for the research. Therefore, the discussion of the methodology in the study will demonstrate the processes, procedures and parameters that will be employed in conducting the study. To develop and implement the appropriate methodology, the discussion of the methods hinged on the "Research Onions" by Saunders, Lewis and Thornhill, (2019) as contained in Figure (3.1).

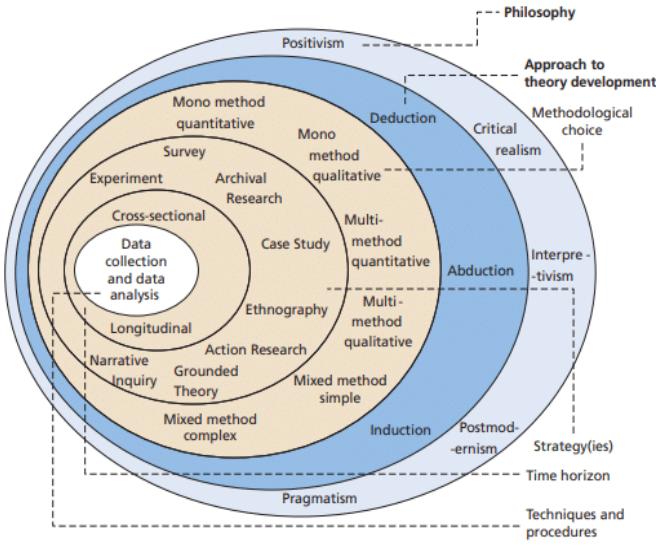


Figure 4:Research Onion (Saunders, Lewis and Thornhill, 2019, p130)

3.2 Research Philosophy

The research philosophy, a set of beliefs and assumptions about the development of knowledge in a study (Saunders, et.al., 2019) inform the process of the study in generating knowledge and the selection of other themes such as research approach, strategy, and methods. Out of the five philosophies (Critical realism, interpretivism, Positivism, postmodernism, and pragmatism) contained in the research onion, the Positivism philosophy was adopted. The choice of philosophy was premised on its relevance to the research objectives and its advantages. Firstly, the Positivism philosophy is associated with scientific research and results in the development of generalized findings from experimentation and structured observations of reality (Ragab & Arisha, 2018). It also requires a researcher to objectively obtain data, remain external in the research process, and be independent of the subject of the study. Premised on this, and given the research objectives, which are to explore the Knowledge graph-based approach in traceability and impact analysis in software development; examine the impact of a knowledge graph-based approach in traceability on stakeholders; and explore the traceability and impact analysis between FRs and NFRs in software development. Therefore, the positivism philosophy was selected as the practical philosophy to guide in the study process.

3.3 Research Approach

Another layer of the research onion that guides the study methodology is the research approach, which is an avenue through which theories are generated, evaluated, and justified. That is, influences how theories are developed and used in a study (Saunders, et.al., 2019). As indicated in the research onion (Figure 3.1), the second layer provides three approaches, which are the deductive approach, the inductive approach, and the abductive approach. Based on the choice of the research philosophy adopted, which is positivism, the appropriate research approach is the deductive approach. Research using the deductive approach begins with a theory developed from already existing literature and then designs a research strategy to test that theory

(Saunders, et.al., 2019). The choice of deductive approach was in line with the research objectives, which required the collection of existing data of requirements to design and test Knowledge graph-based traceability and impact analysis of requirements in software development.

3.4 Methodological Choice

The methodological choice suggests that it can be quantitative, qualitative, or mixed methodology. The methodological choice was further broken down into six categories in the research onions (Figure 3.1). For the study, the quantitative method is adopted. The qualitative method is a systematic investigation phenomenon in which specific and standardized data through experiments, surveys, structured observations, and structured interviews with closed questions are collected. However, the quantitative method was chosen for the study over alternative methods because it is linked to experimental undertakings like design and development and uses numerical scales as well quantitative analysis of users feedback . With that, it paves the way for repeatable steps to develop the artefact as applicable in the current study. In addition, the choice of methodology was also in tandem with the research philosophy and research approach selected to guide the study.

3.5 Design Methodology Choice

The creation of the artifact will adhere to an Agile methodology, focusing on incremental and iterative development. Throughout this research, improvement will be made to the development process until they align with the study's objectives of exploring the use of knowledge graph traceability and impact analysis.

3.6 Research Strategy

Research strategy refers to the method used to address specific research issues. It also includes the techniques and process of data collection as well as analysis. The strategies include

grounded theory, archival research, case study, experiment, survey, Ethnography, action research, and narrative inquiry as contained in the research onion. From the identified research strategy, an experimental strategy is appropriate for the current study given the research objectives, which are to explore Knowledge graph-based traceability and impact analysis of requirements in software development. Hence, an experimental strategy is applicable in this research to develop and implement Knowledge graph-based by establishing traceability and querying impact between requirements.

3.6 Time Horizon

The time horizon level in the research onion, refers to the period within which the researcher will carry out the study. Out of the two-time horizons, which are cross-sectional and longitudinal, the cross-sectional time horizon was chosen (Saunders, et.al., 2019). The choice of the cross-sectional premised on the fact that it aims to study a phenomenon at a particular point in time. Hence, given that this study was a master's degree dissertation that had a specific time frame to be completed, the cross-sectional horizon of time was followed.

3.7 Data Collection and Analysis

A textual dataset of software requirements, which includes both functional and non-functional requirements was compiled from a publicly available source, Kaggle, and includes a compiled list of user stories and acceptance criteria. A non-textual test data set of functional and non-functional was also sourced for training the machine learning model.

Commented [MA1]: Include the source

3.8 Summary

The chapter presented the research methods adopted in the study with the justifications for the appropriate options. The method contained the research philosophy, research approach, methodological choice, research strategy and time horizon. It also states the design methodology choice which is the agile methodology that will be adopted.

CHAPTER 4. DESIGN OF ARTEFACT

This section outlines the design process of the artefact, which aims to address the complexities of managing software requirements, particularly Non-Functional Requirements (NFRs). The artefact, a knowledge graph-based system, is designed to streamline the management of software requirements through an integrated approach that combines a knowledge graph database, data processing and analysis capabilities, and a user-friendly interface.

4.1.1 Design

The artefact development will proceed through several phases, adhering to agile methodology principles outlined in Section 3.5. This iterative approach ensures flexibility and adaptability, allowing for incremental improvements till the research objectives are met. To begin**,** the initial phase focuses on defining requirements and establishing the architectural framework of the knowledge graph.

4.1.1 Requirement Analysis

This sets the groundwork for the development of the artefact crucial for ensuring that the subsequent phases of the artefact are implemented. The following requirements were gathered:

4.1.1.1 Impact Analysis Metric

- **Degree of Centrality:** This is a measure in network analysis that quantifies the number of direct connections a node has in a network. It specifically looks at the number of direct connections of a node within a network, without considering the nature or strength of those connections. It helps identify nodes that are highly connected within the network, potentially indicating their importance or influence(Krnc & Škrekovski, 2020). Nodes with high degree centrality are those with many direct connections. They are often considered central or influential because they interact with many other nodes. Conversely, nodes with few

connections have low degree centrality. These nodes might be peripheral or specialized within the network's context.

- **Dependency strength:** As used in research by (Mczara et al., 2020), this metric will also be used as a criterion to understand the relationships and interdependencies between functional and non-functional requirements. It helps assess how changes to one type of requirement may affect the other based on the strength of their connections within the system. It focuses on the direct or indirect relationships between requirements, considering factors such as frequency of interactions, coupling, and impact propagation providing insights into the degree of influence one type of requirement has on the other. This is valuable for identifying dependencies that may not be immediately obvious, understanding the potential ripple effects of changes, and prioritizing requirements based on their interconnectedness.

4.1.1.2 Web Application Wireframe Design

Using Figma, a user-friendly prototype that interacts with the knowledge graph and explore software requirement relationships was designed. It provides a conversational interface for querying specific data, making it a valuable tool for impact analysis in software development environments. It comprises two main sections: an import and display area for the knowledge graph, and a chat interface for interactive user queries. See appendix for

4.1.1.3 User Story

As a user**, I want to** use a chat interface to query and analyze the impact of changes to software requirements, **so that I** can understand how alterations in one area of the application affect other connected requirements, enabling me to adjust my development strategy accordingly.

Table 2: Shows the Functional and Non-functional Relationships of the User Story in Section 4.1.1.3

Functional Requirements	**Non-Functional Requirements**
Users should be able to upload data files through the "Import Data" button, which are then processed.	The interface should be designed to be user-friendly with clear navigation and responsive design for various devices.
Users should interact with the system through a chat interface to query specific impact analysis or requirement details.	The application should have high responsiveness, with minimal delay in chat responses and quick processing of data uploads and queries.
	The system should handle a growing amount of data and user queries without degradation in performance.

4.1.1.4 Use Case

Given I am using the web application for impact analysis,

When I upload a dataset containing software requirement specifications and initiate a query about the impact of changing a specific requirement,

Then the application should analyse the dataset, update the knowledge graph, and provide an analysis of how the change affects other requirements.

4.1.1.5 Users:

- **Project Managers**: They should be able to use this application to assess how changes to one requirement might affect others, helping in planning and decision-making.
- **Developers**: To understand system requirements and dependencies for better implementation and modification strategies.

4.2 Architectural Design

In designing the artefact, the selection of the technological stack for implementing the artefact is guided by several key criteria, including scalability, performance, compatibility with knowledge graph databases, ease of integration. With these considerations in mind, the chosen tools, frameworks, and technologies encompass a combination of frontend, backend, and database management components.

- **Frontend:** JavaScript framework, React.js and Vite are used for their component-based architecture, flexibility, and robust ecosystem. The chosen frameworks prioritize usability, interactivity, and visualization capabilities and enables the creation of dynamic, responsive user interfaces for uploading data. To facilitate efficient communication with the backend, the frontend leverages API for asynchronous data interactions. This enables the React components to efficiently retrieve, display, and send data to the server.
- **Backend:** Python has been selected as the primary technology due to its versatility, extensive libraries, and ease of integration with knowledge graph databases. Python offers a robust ecosystem for training model. With Python frameworks like Flask providing essential features for building APIs, handling HTTP requests, and managing the application logic is known for its simplicity and flexibility, making it well-suited for developing lightweight backend services tailored to the requirements of the artefact. Additionally, Python's compatibility with Neo4j's enables seamless interaction with the knowledge graph database, for efficient data retrieval, manipulation, and storage. This choice aligns with ensuring scalability, performance

optimization, and smooth integration with the knowledge graph-based system. To enhance the backend capabilities, an integration with open API service from chatgpt will be used to facilitate interaction with the designed model and the model provided by open API service. This allows for robust chat functionality within the web application, enabling users to interact dynamically with the system. Furthermore, WebSocket technology will be integrated to enable real-time communication between the web and the backend server. This is crucial for maintaining a live and interactive user experience, especially in scenarios where users expect immediate responses from the system. These integrations are pivotal in delivering a seamless and dynamic user experience, leveraging the open API for data access and WebSocket for real-time interactivity, which are essential for the functionality and scalability of the chat services within the web app.

- **Graph Database:** This centred around knowledge graph databases that support efficient storage, querying, and traversal of interconnected data. Neo4j, a graph database management system, stands out for its native support for graph structures, query language (Cypher), and scalability features. Neo4j's ability to represent complex relationships between user stories makes it well-suited for the artefact's knowledge graph-based approach to requirement management. This selection is driven by the need for a cohesive, scalable, and efficient solution that leverages the strengths of each component to realize the artefact's objectives effectively. Through careful consideration of selection criteria and compatibility with knowledge graph databases, the chosen tools, frameworks, and technologies lay the foundation for developing a knowledge graph-based system for software requirement management in agile development environments.

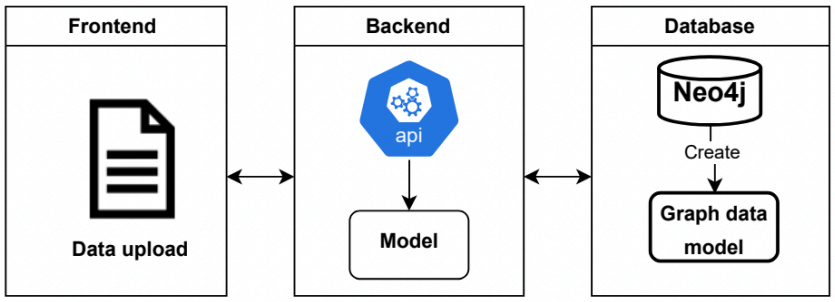


Figure 5: Architectural Design

4.1.2 System Design

The system design illustrated in the sequence diagram in Fig.7 provides a comprehensive overview of how data flows and is processed within an integrated software environment, emphasizing real-time data interaction and efficient data management. The process begins when a user interacts with the web application, initiating data upload through a defined user interface. This interface is designed to be user-friendly, allowing for easy navigation. Once the data is uploaded, it triggers the upload data API within the web application. This function is crucial as it serves as the entry point for data into the system, setting the stage for subsequent processing. After the data is uploaded, it is forwarded to the trained model. This plays a pivotal role in data classification and analyses the data based on predefined criteria and algorithms, classifying it into various categories as needed for further processing. The results of this classification are then sent back to the web application, which takes the processed data to the next step.

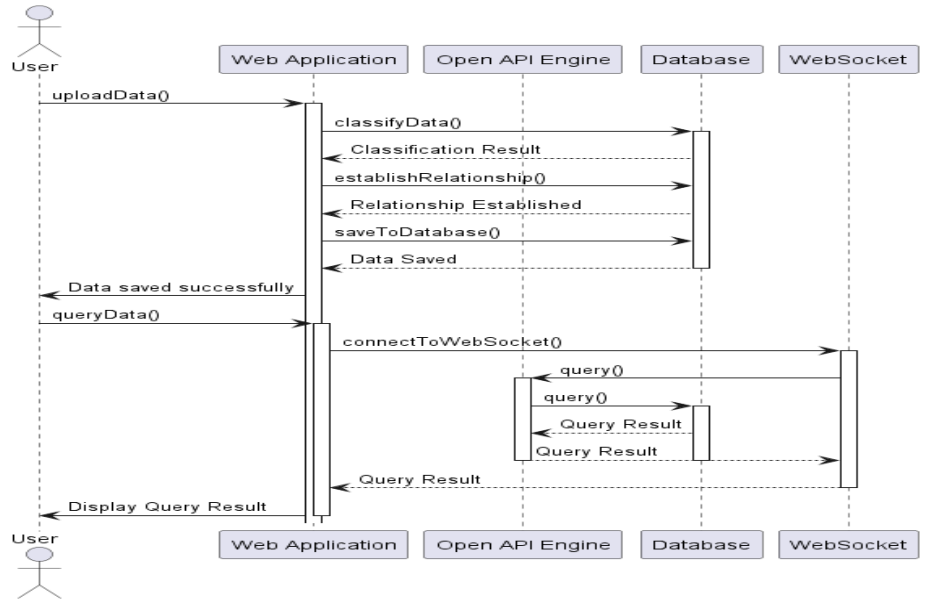


Figure 6: Sequence Diagram

Following data classification, the model establishes relationships between different data points based on the classification results. By defining these relationships, the system can better understand the data interdependencies and interactions, which is essential for comprehensive analysis and decision-making. Once relationships are established, the classified and related data using the is saved into Neo4j database. This step is crucial for data persistence, ensuring that all processed information is securely stored and retrievable. The database confirms the successful saving of data, signalling to the web application that the process has been completed successfully. With the data saved, the user can proceed to query which connects the web application to the database via a WebSocket. The use of WebSocket is significant as it facilitates real-time communication between the client (web application) and the server (database), allowing for instantaneous data querying and retrieval. The open AI service will enable an interaction with the trained model by converting the user query into a cypher query the data base could understand.

Using it capabilities (Open AI) the query results is being sent back through the WebSocket to the web application. The web application then displays these results to the user, completing the cycle from data upload to data retrieval. Figure 8 futhers illustrates how the system interacts with one another.

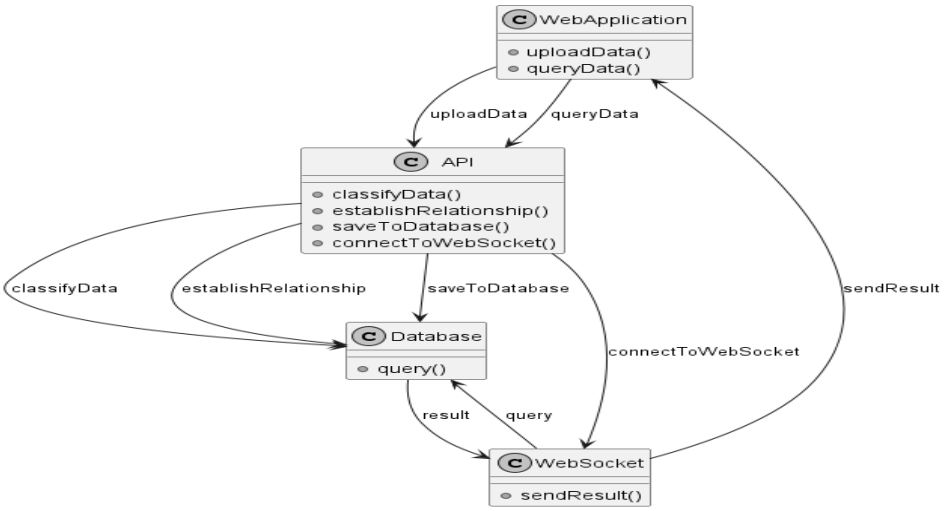


Figure 7: Class Diagram

4.4. Implementation

4.4.1 Training Machine Learning Model

This research trained a machine learning model using the BERT tokenizer, tokenizer from the 'transformers' library, Pandas for data handling and TensorFlow for binary classification of software requirements. The dataset was processed and split into training and testing sets and trained over three epochs with the Adam optimizer. The evaluated model achieved a training loss: 0.1824, training accuracy 93.41% (0.9341), validation loss: 0.1506 and validation accuracy: 91.84% (0.9184).

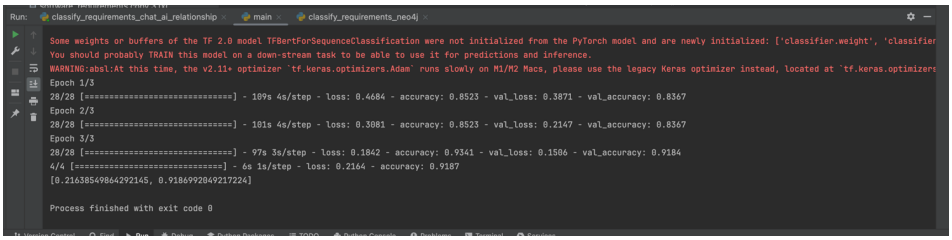


Figure 8: Trained Model

4.4.2 Requirements Classification

Utilizing the pre-trained BERT-based model, the software requirements was extracted by classifying the requirements into "Functional" or "Non-functional" classes. The classification results were represented in a structured JSON format which can be seamlessly integrated into knowledge graph-based system. This enables the incorporation of categorized requirements into the graph database for further analysis and visualisation.

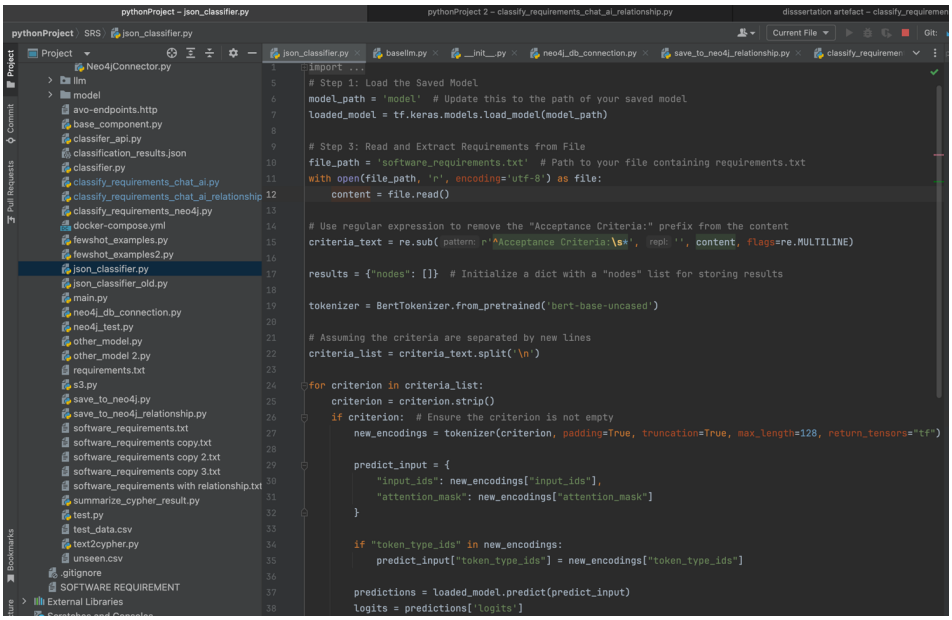


Figure 9: Requirement Classification

The JSON output in Fig.7, details the classification of requirements into functional and non-functional categories. For example, requirement id "3" about the accuracy of account balance is classified as " Non-functional," emphasizing constraints or conditions the system must meet. Conversely, requirement id "2," about displaying the balance upon login, is classified as "Functional," indicating it describes a specific system functionality or feature.

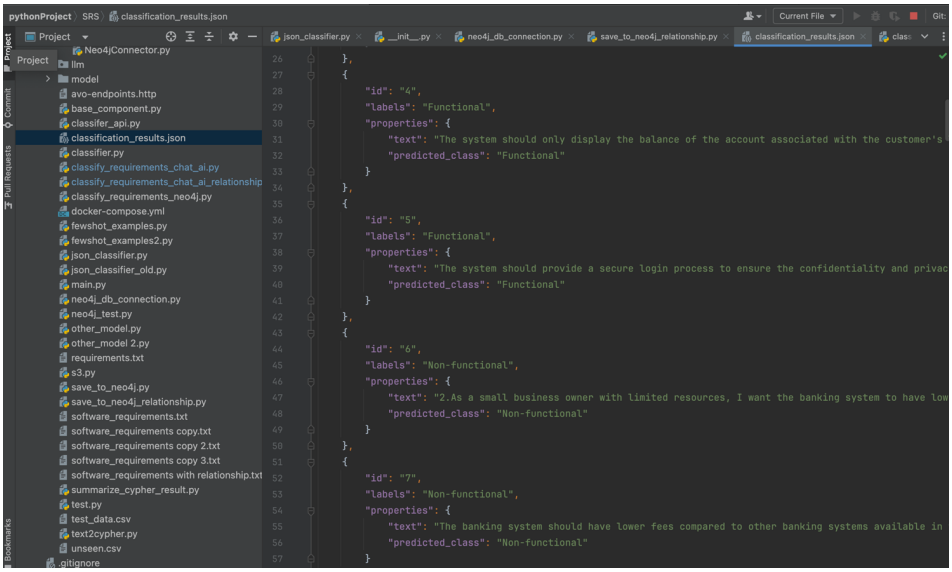


Figure 10: Result of Classified Requirement

Prior to selecting BERT, the SVM model were used for text classification, achieving accuracies of 92.68% and 84.55%, respectively (see Fig.10). However, these models' limitation with nuanced linguistic patterns and contextual dependencies critical in software requirements, BERT excels due to its capability to process bidirectional contextual information from large text corpora, pre-trained on extensive datasets to grasp subtle language nuances and generalize across different domains. This proficiency makes BERT especially effective for the complex language involved in software requirements (S. Liu, 2019).

Figure 11: Model Evaluation

4.4.3 Backend Integration

A connection to a Neo4j graph database via the Neo4j Python library is configured with the database URI, username, and password to facilitate a stable link to the database. To enhance user interaction with data, an API was developed to manage data uploads and queries efficiently. Key functionalities include handling file uploads through the `/classify/` and `/classifyV1/` endpoints, which process software requirements by classifying content using a pre-trained BERT model (see section 4.4.1) for sequence classification tasks. Subsequently, the classified data is parsed and structured into nodes and relationships within the Neo4j database using the `create_nodes_and_relationships` function. The `Neo4j` plays a critical role in managing these database interactions, ensuring that nodes and relationships are accurately stored and retrieved according to the classification results. Moreover, a WebSocket was integrated into the system, utilizing the OpenAI API to leverage large language models for generating dynamic responses based on user inputs.

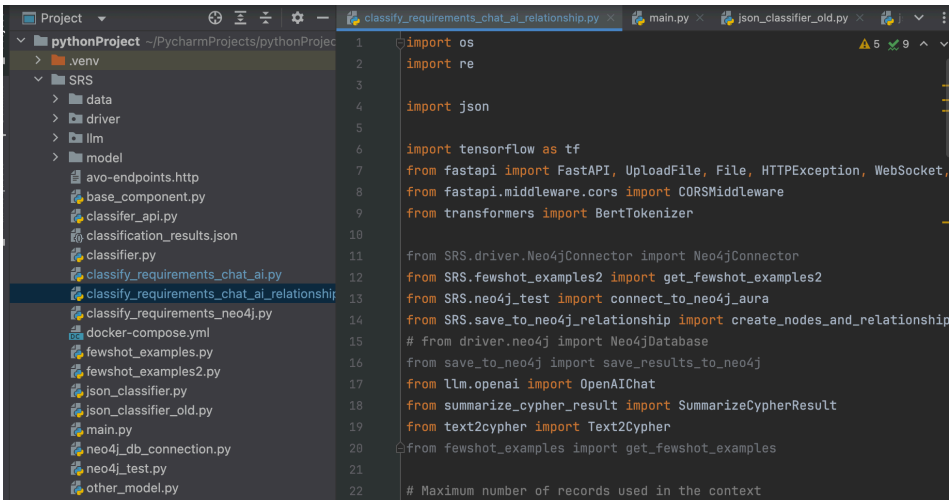


Figure 12: Show the Backend Integration

4.4.4 Graph Data Model

Prior to storing the data in Neo4j, graph data model was developed to intuitively manage, store, and query relationships between data points, especially suited for datasets where relationships are critical. The model visualizes nodes as circles and relationships as connecting lines as seen in Fig.9, aiding in understanding how various requirements are interlinked for impact analysis.

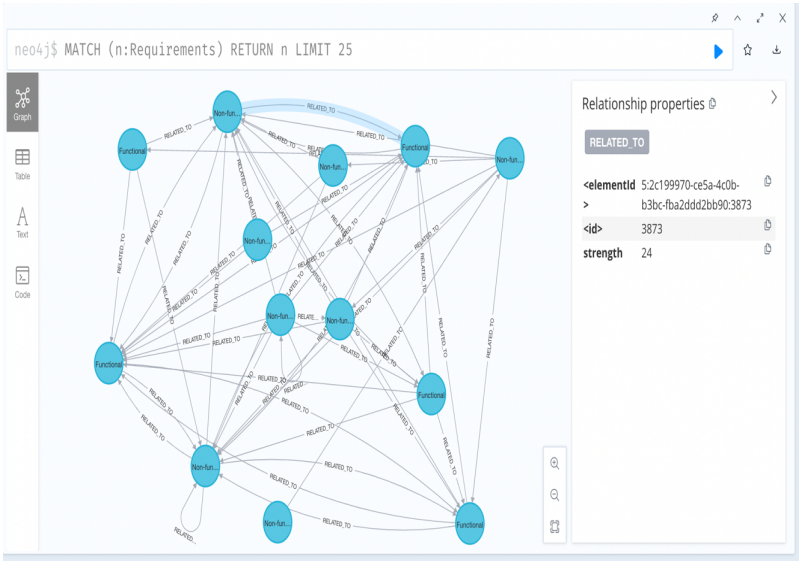


Figure 13: Graph Data Model

The graph schema detailed in Table 2 formalizes the structure and constraints of the database, clarifying how data is organized and its significance within the model.

Table 3: Shows the Structure of the Graph Model

Nodes	**Description**
Node labels	Each node is labeled as "Requirement", indicating that each node represents a specific requirement.
Node properties	Each node has properties such as elementId, id, class, description, etc. These properties store information about the requirement:
elementId	A unique identifier for the node
Id	A numeric identifier for the requirement
class	The type of requirement (e.g., "Non-functional").
description	A text description of what the requirement entails. For example, the highlighted node talks about a frequent traveller needing competitive exchange rates in multiple currencies from their banking system.
Edges	

Relationships	The edges are labelled as "RELATED_TO", suggesting that each connection represents a relationship where one requirement is related to another. This signifies dependencies between requirements.

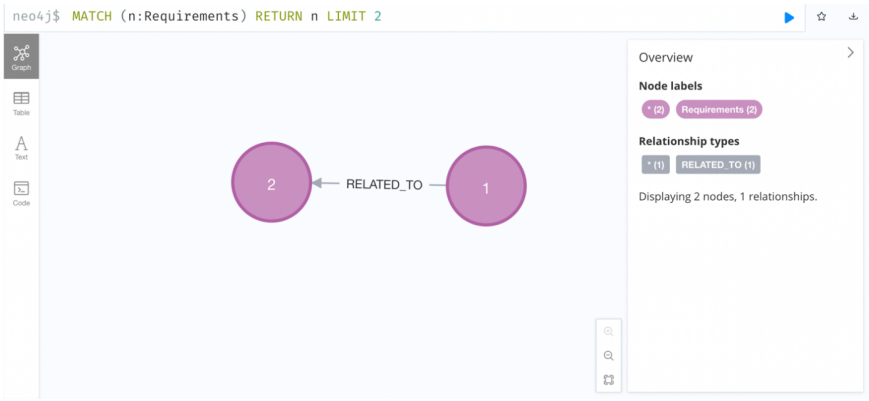


Figure 14:Graph Model showing the relationship between nodes.

For instance, the nodes labelled "1" and "2" in Fig 10 are connected by the edge "RELATED_TO", representing a dependency or sequence in which one requirement must be met for the other to be viable, thereby clarifying how requirements interact and affect each other within the system (see Table 3).

Table 4: Shows how node 1 and 2 interacts.

Dependency	Requirement 2 depends on the completion or existence of Requirement 1
Sequence	Requirement 2 should follow Requirement 1 in terms of workflow or implementation sequence.
Influence	Changes in Requirement 2 might impact Requirement 1, or it could be that Requirement 2 is derived or influenced by Requirement 1.

4.4.6 Impact Analysis Query

To perform impact analysis, traceability between requirements was utilised to implement degree of centrality and dependency strength. Fig. 11 shows that the degree of centrality was

calculated using a cypher query that assigns a property key to each node indicating the number of direct connections it has, identifying central nodes with numerous relationships which are crucial and likely to change significantly if altered.

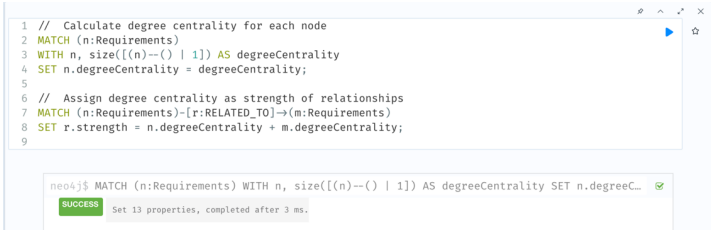


Figure 15: Degree of Centrality Query

Also, a property key for dependency strength on the edges between nodes, as illustrated in the Fig.12 was set. This metric quantifies the dependency between requirements, considering interaction frequency and the impact of potential changes, and is stored directly on the relationships within the graph database.

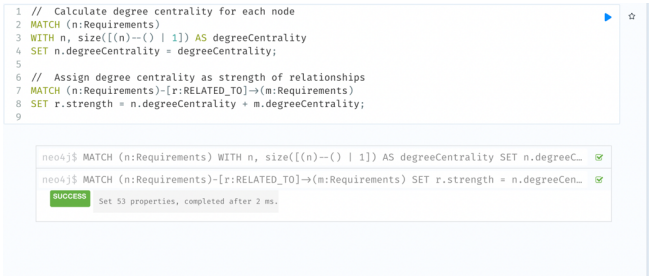


Figure 16: Dependency Strength Query

By integrating these two metrics into the graph model, it will enable a comprehensive form of impact analysis informing an approach to managing dependencies of requirements and ensuring that non-functional requirements are effectively managed.

4.4.7 Web Application Implementation

Using the wireframe as a guide (see Fig.5), web application featuring a chat box was developed powered by API that allows document uploads, text classification, and result queries. The front end, built with React, employs CSS for styling, Flexbox for responsive design, react-icons for

interactivity. The web app uses Axios library for performing Rest API communication for uploading text file through a post Rest API command and uses React web socket library roaming on OpenAPI for open and close communication and querying Neo4j database.

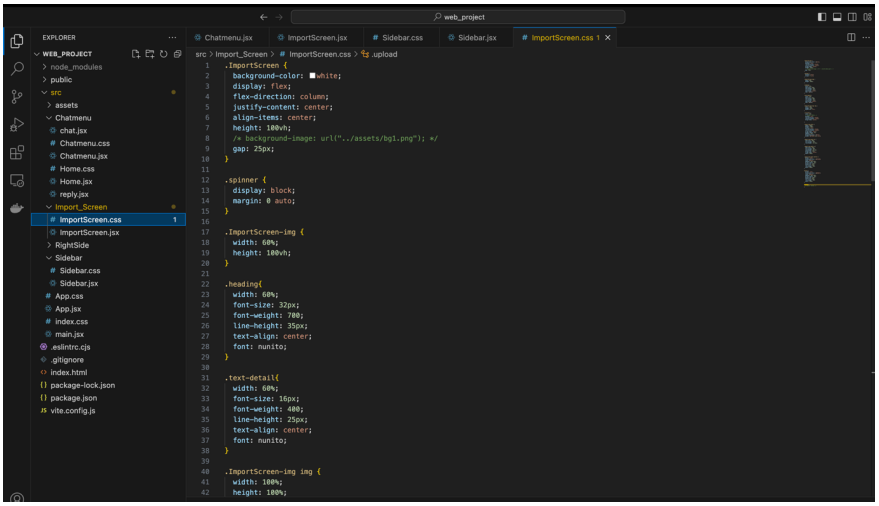


Figure 17: Web App Implementation

4.4.8 Versioning and Deployment

To carry out the validation by industry experts for this implementation, the deployment of the application was done using a combination of GitHub, AWS, Netlify and Render. The application's codebase is managed and version-controlled on GitHub, enabling automatic updates and seamless integration for continuous deployment workflows. For the backend, an AWS S3 bucket, named `dissertationartefact`, securely stores the machine learning model, making it accessible across the cloud. Security is reinforced through an IAM user with `AmazonS3FullAccess` permissions, with credentials managed via environment variables to avoid exposure while Render, which detects repository updates and automates the deployment process, ensuring the application is consistently synchronized with the latest changes was used to deploy the FastAPI, providing a reliable endpoint for user interactions.

However, the frontend was deployed on Netlify by also integrating GitHub repository essential for leveraging Netlify's continuous deployment feature, allowing automatic syncing of code changes. It manages SSL certifications for secure HTTPS connections and provides a dashboard for real-time monitoring and easy rollbacks.

4.5 Testing and Validation

The testing and validation phase of for the Knowledge Graph-based web application represents a critical stage in this research, where the effectiveness and reliability of the implementations was assessed to ensure alignment with the research objectives, and capable of addressing identified research gaps. A combination of automated tests and expert user evaluations was done to thoroughly examine the application capabilities.

Table 5: Shows the various testing carried out:

Testing	**Objective**	**Description**
Functional	Verify the functionality of system.	Postman was used to test the API functionality to ensure the data files imported are accurately represented within the knowledge graph, reflecting all edges and nodes. Also testing the chat interface to correctly processes and responds to user queries based on the knowledge graph data.
Performance	Measure the system's efficiency and responsiveness.	A test on web socket was performed to ensure that queries can handle escalating amounts of data efficiently and to assess response times.

Usability	Measure the system's efficiency and responsiveness.	Collected and analysed expert feedback on the functionality layout to determine the intuitiveness of the design.

4.5.1 Traceability and Impact Analysis between FR and NFR

Fig. 18 validates that it is possible to establish relationship between FR and NFR using knowledge graph. It shows a clear traceability path from general user needs (viewing account balances online) to specific functional execution (display on login) and quality assurance (accuracy and timeliness).

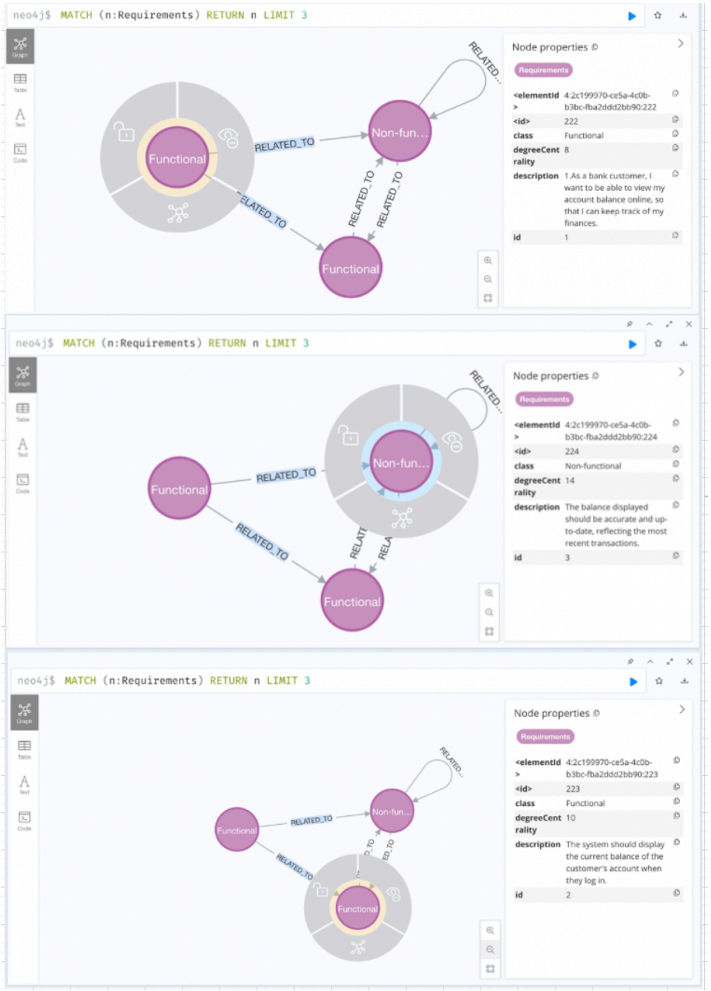


Figure 18:Traceability between FR and NFR

Analysis of Relationships

- FR to FR: As seen in Fig. The second requirement (ID 2) is a direct extension of the first (ID 1), specifying when and how the balance should be displayed based on the customer's action of logging in.

- FR to NFR: The non-functional requirement (ID 3) is linked to the second functional requirement (ID 2). This link indicates that the functional requirement to display the account balance must meet the non-functional standards of accuracy and timeliness at login, modifications to display or update methods must also ensure compliance with these accuracy standards.

4.5.2 Impact Analysis Based on Degree of Centrality

As discussed in section 4.1.1 the degree of centrality as seen in Fig.14 indicates that nodes with high degree centrality (like nodes "3", "5", and "13" with degrees of 14) are likely critical in the network. Changes to these nodes or the requirements they represent could have significant ripple effects affecting multiple other nodes.

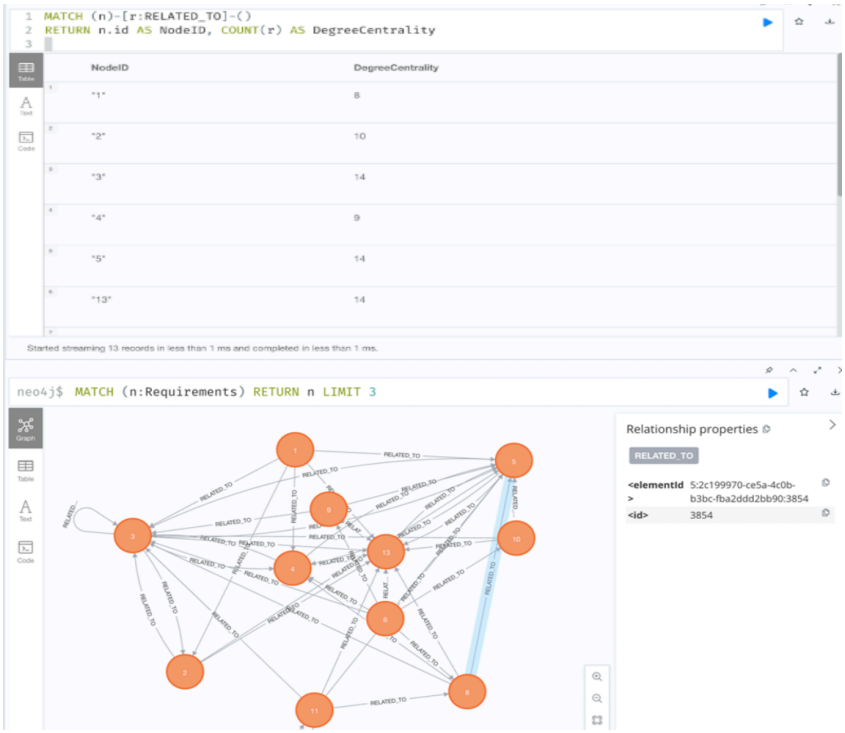


Figure 19: Degree Centrality between FR and NFR

Analysis

Node 3's high degree centrality indicates it is a major hub within the network. This centrality suggests that Node 3 plays a crucial role in the network, likely influencing or being influenced by many other nodes. In practical terms, this could mean:

- Node 3 serves as a crucial central requirement in the network, essential for the functionality of many other nodes.
- Any failure or modification in Node 3 could directly impact 14 other nodes, potentially causing widespread disruption across the network.
- Due to its critical role, Node 3 requires prioritized resource allocation, rigorous testing, and careful monitoring to maintain network stability.

4.5.3 Impact Analysis Based on Dependency Strength

The query (see Fig.15) retrieves pairs of nodes (n and m) that represent requirements and the strength of their relationship (r. strength).

Interpretation of Results

- High-Strength Relationships: The strongest relationships, all with a strength of 28, include connections between nodes 3 and 5, 5 and 13, and 13 and 5. These connections suggest significant dependencies or interactions between these requirements.

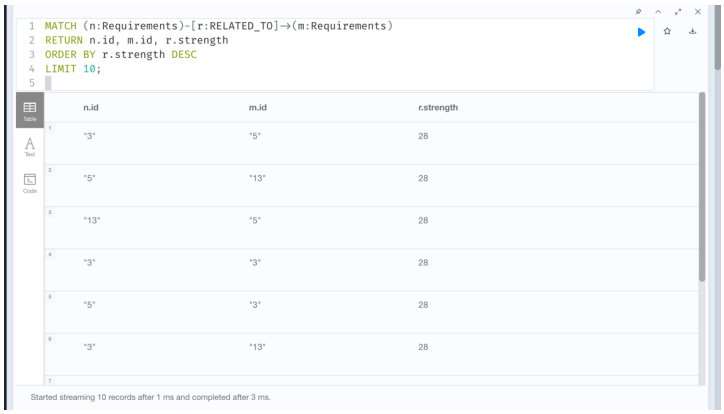


Figure 20: Dependency Strength between FR and NFR

- Symmetry: Some relationships appear twice with reversed node orders (n.id and m.id), such as 3-5 and 5-3, or 13-5 and 5-13. This indicates that the relationships are bidirectional, or their representation considers the influence or dependency to be mutual (see Fig. 20).

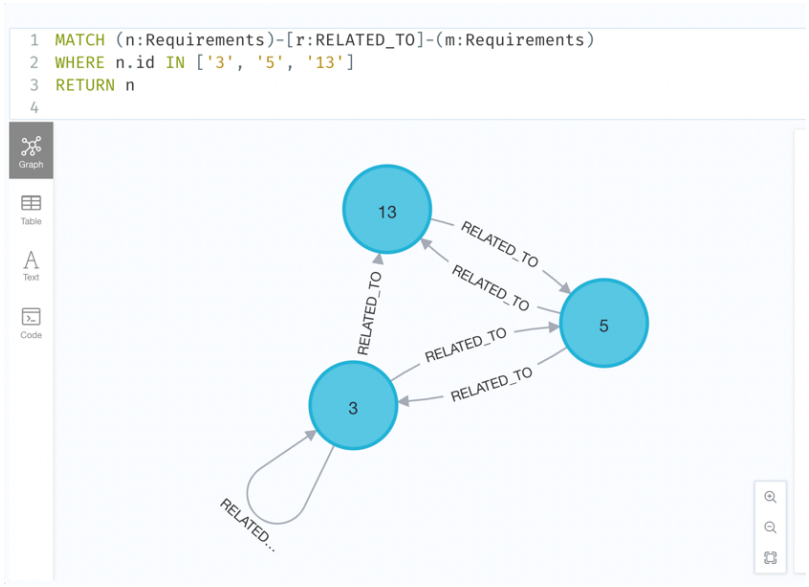


Figure 21: Graph Model Showing Dependency Strength between FR and NFR

The highest values of **r.strength** suggest that these connections are critical within the network. Changes or issues within one node in these pairs could significantly impact the other, reflecting

a high degree of interdependence. Identifying the strongest relationships might also guide where to focus resources, such as more rigorous testing, more frequent reviews, or higher development priority, to ensure the stability and performance.

4.5.4 Web App Validation

Twenty experts were invited to test the web application for traceability and impact analysis, but feedback was received from only ten. These experts demonstrated a high level of familiarity with software development processes as seen in Fig.22, lending credibility to their evaluations. The application was well-received by obtaining their consent, which most found extremely helpful in understanding the impact of changes in software requirements.

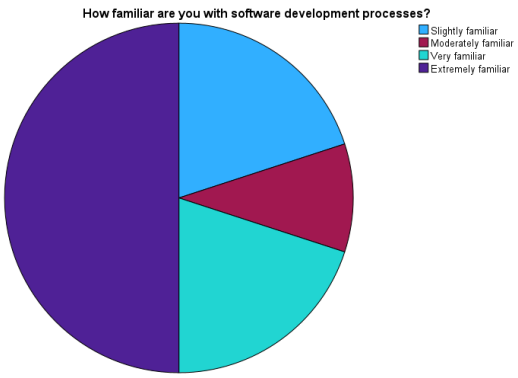


Figure 22: Descriptive Statistics

Using SPSS, the table in Fig 25 presents descriptive statistics from the survey. The feedback indicates potential for enhancement in the web application, particularly in how it delivers insights into software changes, where the mean score is 1.40, while the overall user experience and clarity of traceability links within the knowledge graph are rated higher (means of 4.10 and 4.40 respectively), the interactive chat feature, with a mean of 1.70, shows room for improvement. These results suggest that while the application has strong foundations, particularly in user

interface and functionality, further refinements are necessary to maximize its effectiveness and user engagement.

Descriptive Statistics

	N	Range	Mean		Std. Deviation	Variance
	Statistic	Statistic	Statistic	Std. Error	Statistic	Statistic
Have you used similar web application before?	10	4	2.70	.559	1.767	3.122
Based on your experience, do you feel the web application provides valuable insights into the impact of changes in software requirements?	10	2	1.40	.221	.699	.489
What are your thoughts on the interactive chat querying the changes in software requirements. Was it helpful in understanding the impact	10	3	1.70	.300	.949	.900
On a scale of 1 to 10, how would you rate your overall experience with the prototype testing session?	10	2	4.10	.233	.738	.544
How clear were the traceability links established within the Knowledge Graph? Did they effectively connect different requirements?	10	2	4.40	.221	.699	.489
How familiar are you with software development processes?	10	3	4.00	.394	1.247	1.556
Valid N (listwise)	10					

Furthermore, a correlation analysis in Fig.23 indicated a perfect positive correlation ($\rho = 1.000$) between the clarity of traceability links and the value of insights provided, affirming the application's effectiveness in delivering actionable insights. However, a notable negative correlation ($\rho = -0.497$) between how effectively different requirements were connected suggests areas for enhancement.

Nonparametric Correlations

Correlations

			On a scale of 1 to 10, how would you rate your overall experience with the prototype testing session?	What are your thoughts on the interactive chat querying the changes in software requirements. Was it helpful in understanding the impact
Spearman's rho	On a scale of 1 to 10, how would you rate your overall experience with the prototype testing session?	Correlation Coefficient	1.000	-.411
		Sig. (2-tailed)	.	.238
		N	10	10
	What are your thoughts on the interactive chat querying the changes in software requirements. Was it helpful in understanding the impact	Correlation Coefficient	-.411	1.000
		Sig. (2-tailed)	.238	.
		N	10	10

Figure 23: Correlation Analysis I

Also, Fig.24 indicates that while the application effectively supports decision-making in software development, improving the connectivity between different requirements could enhance its utility further.

Nonparametric Correlations

Correlations

			Based on your experience, do you feel the web application provides valuable insights into the impact of changes in software requirements?	How clear were the traceability links established within the Knowledge Graph? Did they effectively connect different requirements?
Spearman's rho	Based on your experience, do you feel the web application provides valuable insights into the impact of changes in software requirements?	Correlation Coefficient	1.000	-.497
		Sig. (2-tailed)	.	.144
		N	10	10
	How clear were the traceability links established within the Knowledge Graph? Did they effectively connect different requirements?	Correlation Coefficient	-.497	1.000
		Sig. (2-tailed)	.144	.
		N	10	10

Figure 24: Correlation Analysis II

4.6 Critical Evaluation

Drawing insight from the test and validation, the implementation was critically evaluated employing a SWOT analysis to examine the implementation of Knowledge Graphs (KGs) in managing software requirements, focusing on the integration of machine learning models and graph-based systems for enhancing traceability and impact analysis. It examines how effectively

this approach addresses the gaps, with a particular focus on methodological innovations, practical implementations, and the broader implications for software development practices.

4.6.1 Strengths

- **Advanced Data Structuring and Semantic Relationships:** KGs inherently facilitate the structuring of data that enhances the representation and management of complex relationships among software requirements. This capability allows for a more nuanced understanding and visualization of dependencies, which is a significant advantage in maintaining the coherence and accuracy of project data.
- **Enhancement of Traceability and Impact Analysis:** The primary strength of using KGs lies in their ability to improve traceability and impact analysis within software development processes. By dynamically linking requirements and documenting their interdependencies, KGs provide clear pathways for assessing the impact of changes, thereby supporting more informed decision-making and risk management.
- **Integration with Agile Methodologies:** It is well-suited to agile environments where requirements can change frequently and rapidly. The real-time update capability of KGs aligns with the iterative and incremental nature of agile development, facilitating continuous feedback and adjustments.
- **Use of Machine Learning for Automation:** The integration of machine learning models to classify and analyse requirements further enhances the capabilities of KGs. These models can automate the extraction and classification of data, reducing manual overhead and improving the accuracy and efficiency of processing requirements.
- **Impact on Software Development Practices:** The practical impact of KGs in software development is significant, particularly in enhancing the traceability and impact analysis within agile projects. This capability allows for better management of changes and more efficient risk

Commented [MA2]: reference

Commented [MA3]: reference

assessment, which is crucial in fast-paced development settings. Moreover, the theoretical contributions of this research enrich academic discussions on requirement management and the practical application of KGs, highlighting its role in advancing current methodologies.

- **Enhanced Accessibility through User Interface Design**: One of the notable strengths of the KG implementation is the user interface design, which is strategically developed to ensure ease of access for users with varying levels of technical expertise. This inclusivity is critical as it democratizes the use of complex KG systems, allowing more stakeholders to engage effectively with the technology without a steep learning curve. The intuitive design helps in reducing the barrier to entry, making it a valuable tool for a diverse set of users within agile project environments.

4.6.2 Weaknesses

- **Complexity in Implementation;** The advanced nature of KGs and their reliance on sophisticated technologies can be a double-edged sword. The complexity of setting up, maintaining, and updating KG systems requires significant technical expertise and resources, which could be a barrier for smaller teams or organizations with limited IT capabilities.
- **Cost:** Implementing and deploying the technologies such as machine learning models incurs considerable expenses. For example, hosting the model on an Amazon S3 bucket proved to be financially demanding. This factor can be a major barrier, especially for small to medium enterprises or projects with limited budgets.
- **Accuracy:** An inconsistency in the query result was discovered which can significantly undermine the functionality and reliability of the application.
- **Feedback Diversity**: The current feedback mechanism relies heavily on a select group of experts, which poses a risk of bias. This approach might overlook certain usability issues or user needs that are only visible from the perspective of novice users or stakeholders from

different backgrounds. By diversifying the sources of feedback, the system can be evaluated more comprehensively, ensuring that it meets a broader range of user expectations and operational requirements.

4.6.3 Opportunities

- **Expansion into Various Development Environments:** The flexibility and scalability of KGs present opportunities for expansion beyond traditional software development into areas like systems engineering, where complex dependencies are common. This expansion could open new markets and applications for KG technologies.
- **Enhancements in AI and Machine Learning Integration:** Continuous advancements in artificial intelligence and machine learning offer opportunities to further enhance the capabilities of KGs. For instance, more advanced techniques could improve the system's ability to understand and process complex requirement documents automatically.
- **Collaborative and Cross-Functional Teams:** KGs can facilitate better collaboration among diverse teams by providing a unified view of software requirements and dependencies. This transparency is crucial for cross-functional teams that include stakeholders from different domains.
- **Scalability and System Integration**: There are significant opportunities for scaling the use of KGs to accommodate larger datasets and a growing user base. Exploring specific strategies to enhance the scalability of KG systems could lead to more robust and efficient operations as project demands increase. Additionally, there is potential for integrating these systems into various software development environments, which would broaden their applicability and utility across different sectors and project types.
- **Advanced Research**: The study opens avenues for further research into the long-term sustainability and performance of KG systems in operational settings. Investigating these

aspects could lead to significant improvements in system design and functionality, potentially establishing KGs as a standard tool in software development.

4.6.4 Threats

- **Technological Obsolescence:** The rapid pace of technological change poses a threat to KG implementations, as newer and potentially more effective solutions could render current KG-based systems obsolete. Staying abreast of technological advancements and continuously updating the systems can be resource intensive.
- **Resistance to Change:** There may be resistance from within organizations, particularly from those accustomed to traditional methods of requirement management. Overcoming this resistance requires effective change management strategies to demonstrate the benefits and ease the transition to KG-based systems**.**

4.7 Summary

In this chapter of the research, it outlines the design and implementation of a knowledge graph-based system aimed at enhancing software requirement management, specifically for non-functional requirements (NFRs). It details the process of developing an artifact that integrates a knowledge graph database, data processing and analysis, and a user interface, all designed following agile methodologies. It begins with a thorough requirement analysis to identify essential functionalities and metrics such as Degree of Centrality and Dependency Strength, which assess the influence and interdependencies among requirements. The technology stack selected incorporating React.js, Python, Flask, and Neo4j supports scalability, performance, and ease of integration. The system architecture then processes user stories through a machine learning model, classifying them into functional and non-functional requirements stored in the Neo4j database for detailed impact analysis. This setup enables a comprehensive traceability and impact analysis through a web application, facilitating dynamic interaction and decision-

making for development teams and stakeholders. However, despite its effectiveness, the system faces challenges such as the complexity of maintaining the knowledge graph, high operational costs, and the dependence on high-quality data for accurate results. Strategies to overcome these challenges and potential risks will be discussed in the future work section.

CHAPTER 5. CONCLUSION AND FUTURE WORK

5.1 Conclusion

The utilisation of Knowledge Graphs (KGs) for traceability and impact analysis significantly enhances the management of software development, particularly within agile frameworks. A review of related literature emphasizes the indispensable role that both functional and non-functional requirements play in shaping software development projects. Through the strategic application of Knowledge Graphs, this study demonstrates a methodical approach that not only improves the traceability and impact analysis of these requirements but also aids in more effective decision-making and boosts the efficiency of development workflows.

The research findings indicate that Knowledge Graphs provide a sturdy infrastructure for handling the intricate web of interdependencies among software requirements. By mapping out relationships and attributes in a dynamic graph structure, KGs facilitate real-time queries and updates. This feature is particularly crucial in agile development environments where requirements are subject to frequent modifications. Such flexibility ensures that stakeholders have a precise grasp of project dynamics, enabling them to make swift, well-informed decisions

that enhance the project's overall agility and responsiveness. Moreover, the study sheds light on the integration of machine learning technologies to streamline the classification of requirements.

This integration significantly minimises manual labour and enhances the accuracy of the data input into the Knowledge Graph. This advanced automation allows for a more detailed and nuanced examination of how alterations to requirements impact the software development process. Consequently, stakeholders are equipped with comprehensive insights into the potential ramifications of these changes with the use of the web application, facilitating better strategic planning and management of project outcomes.

5.2 Future Work

Looking ahead, there are numerous potential directions for extending the research on Knowledge Graphs beyond the scope of this initial study, which concentrated primarily on agile software development environments. Future investigations might assess the utility of Knowledge Graphs in fields outside of software development, such as systems engineering and product development, where complex requirement interdependencies similarly challenge project managers and development teams.

As Knowledge Graphs become increasingly pivotal in managing critical systems, the demand to scale these solutions to accommodate larger datasets and handle more sophisticated queries without a loss in performance will grow. Future research should focus on enhancing the scalability and efficiency of the underlying graph database technologies. Developing innovative architectures that support higher levels of scalability could help in managing the growing complexity and volume of data.

Also, this research has introduced a basic web application designed for interacting with the Knowledge Graph. However, there is significant scope for developing this application further. Future improvements could include more advanced user interfaces and interaction designs that simplify navigation and enhance usability, thus making these systems more accessible to a broader range of stakeholders, including those with limited technical expertise. By improving the interface, the adoption of KGs could become more widespread, enhancing the collaborative capabilities across various project teams and effective change management strategies will be essential to mitigate the risks of obsolescence and resistance from traditional practices.

Moreover, to fully establish the effectiveness of Knowledge Graph-based systems in real-world applications, additional empirical research is necessary. Longitudinal studies, in particular, would be valuable as they could track the performance and impacts of KGs over extended periods, offering insights into their long-term viability and effectiveness. These studies would help in understanding how KGs can adapt and scale over time within dynamic project environments.

To maximise their utility, it is crucial that Knowledge Graphs integrate seamlessly with existing project management and software development tools. Future work in this area could focus on the development of APIs and plugins that facilitate easy integration with widely used software development tools and platforms, potentially making KGs a standard component of the software development toolkit. Such integration would ensure that KGs enhance, rather than disrupt, existing workflows, encouraging their adoption across the software development industry.

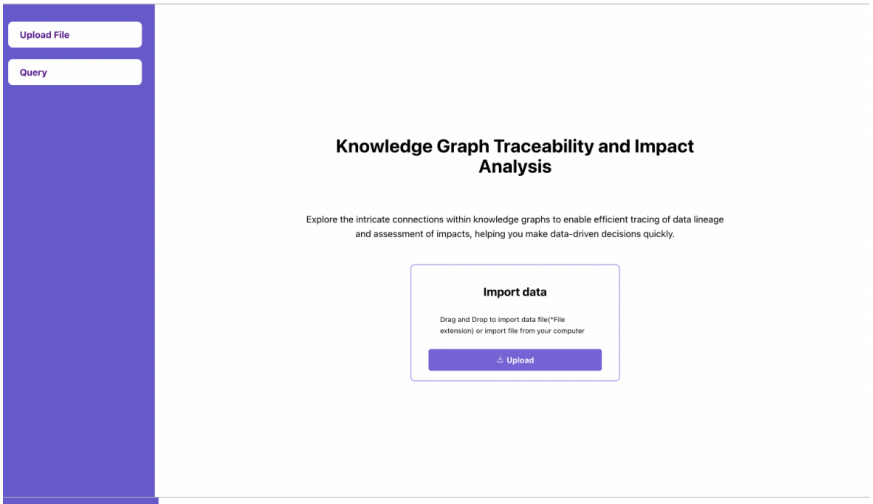
Lastly, Addressing the challenges of complexity and high costs in KG implementation will require future initiatives to develop more streamlined and economical KG frameworks. This effort should focus on designing modular components that are simple to integrate, require less

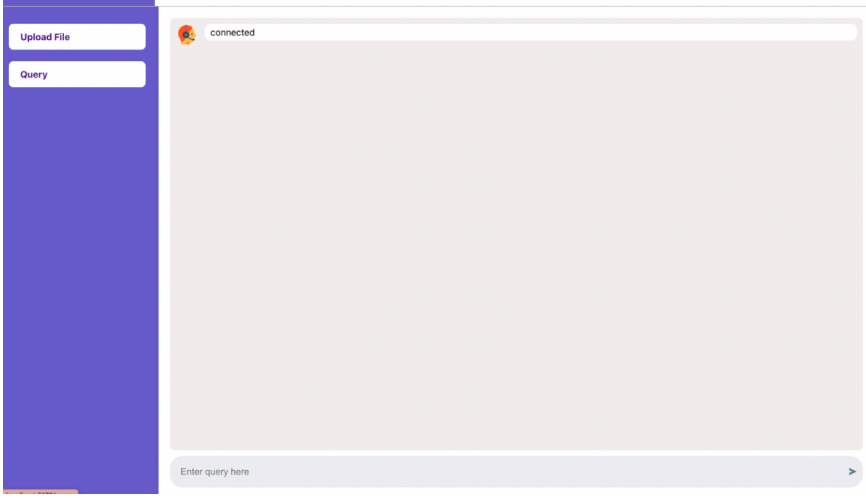
technical knowledge, and are cost-effective for small teams and organizations. Moreover, considering cloud-based options or collaborating with cloud service providers may help lower infrastructure expenses and simplify system complexity.

REFERENCE

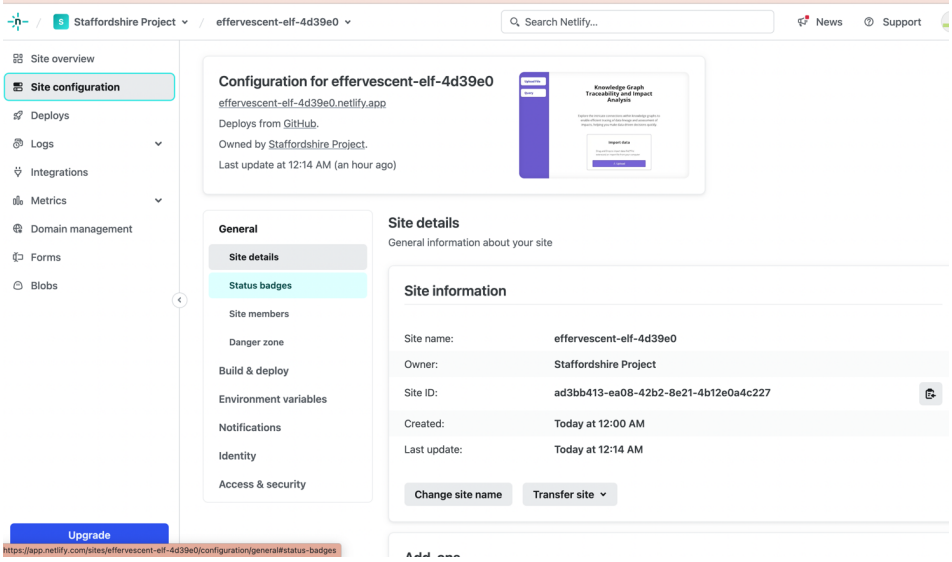
APPENDICES

Appendix A: Wireframe of the web application

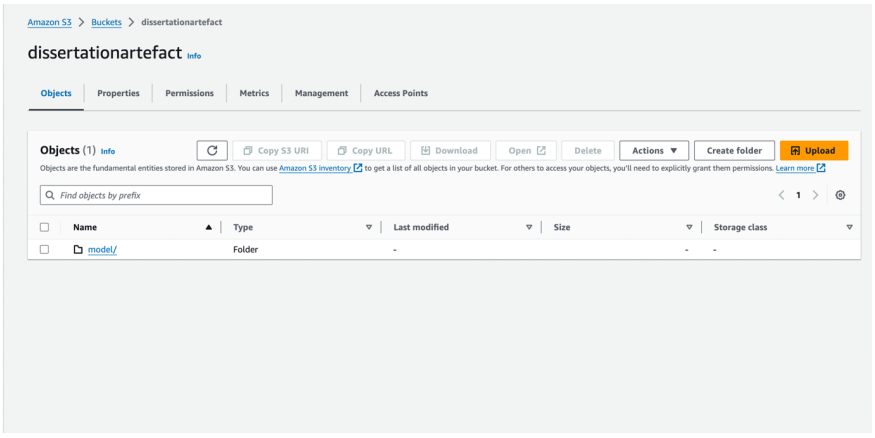




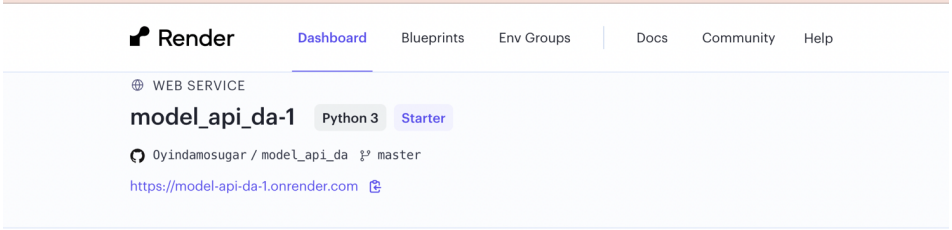
Appendix B: Web application deployment



Appendix C: AWS S3 Bucket

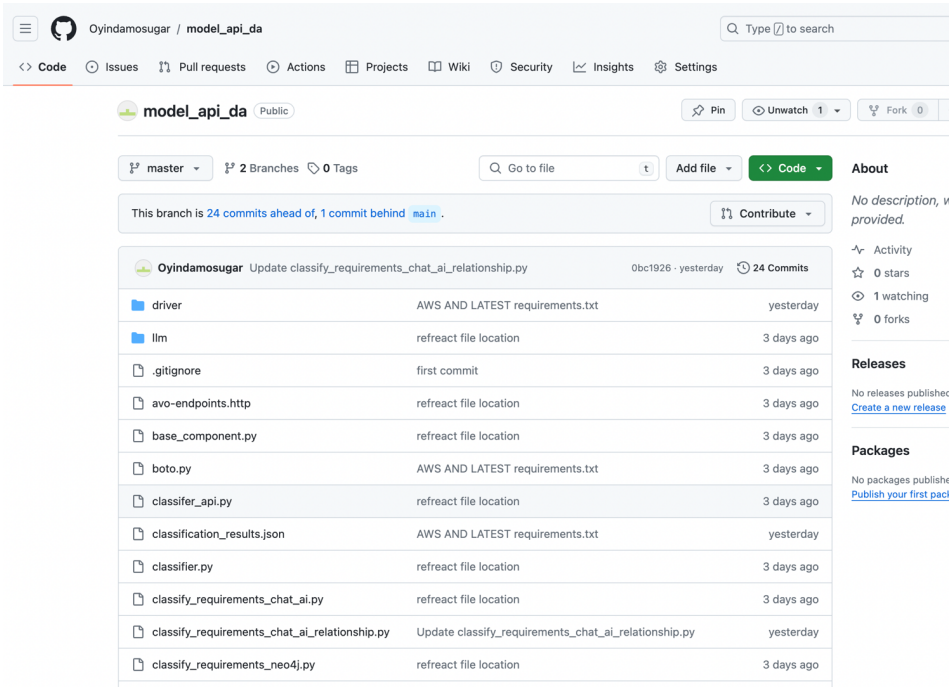


Appendix D: Backend Deployment



Appendix E: Link to Github

Backend: https://github.com/Oyindamosugar/model_api_da/tree/master



Appendix F: Consent Form

Appendix G: Questionnaire