FYP Project

Adam Mcloughlin – K00261195

Intelligent Incident Resolution System using Semantic Matching and NLP Techniques for Enhanced Support Efficiency

A black and gold rectangles

Description automatically generated

Table of Contents

[1. Introduction / Proposal 4](#_Toc195612462)

[1.1. Project Abstract 4](#_Toc195612463)

[1.2. Research Context/Background 4](#_Toc195612464)

[1.3. Research Aim 5](#_Toc195612465)

[1.4. Research Objectives 5](#_Toc195612466)

[1.5. Research Questions 7](#_Toc195612467)

[1.6. Proposed System Architecture 8](#_Toc195612468)

[1.7. Proposed Technologies Used 11](#_Toc195612469)

[1.8. Research Methods 12](#_Toc195612470)

[2. Literature Review 13](#_Toc195612471)

[2.1. Introduction 13](#_Toc195612472)

[2.1.1. Overview of Traditional Systems and Limitations 14](#_Toc195612473)

[2.1.2. The Need for and Importance of Automating the Incident Management Process 15](#_Toc195612474)

[2.2. The Role of Machine Learning in Enhancing Incident Management 15](#_Toc195612475)

[2.2.1. Introduction to Machine Learning (ML) 15](#_Toc195612476)

[2.2.2. Applications of Machine Learning 17](#_Toc195612477)

[2.2.3. Challenges of Implementing Machine Learning 19](#_Toc195612478)

[2.2.4. Benefits of Machine Learning in Incident Management 19](#_Toc195612479)

[2.2.5. Comparative Analysis Across Studies 19](#_Toc195612480)

[2.2.6. Critical Analysis and Gaps 20](#_Toc195612481)

[2.3. Natural Language Processing (NLP) in Incident Management 21](#_Toc195612482)

[2.3.1. Applications of NLP 22](#_Toc195612483)

[2.3.2. Benefits of Natural Language Processing in Incident Management 24](#_Toc195612484)

[2.3.3. Challenges of Implementing NLP Techniques 25](#_Toc195612485)

[2.3.4. Critical Analysis and Identification of Gaps 25](#_Toc195612486)

[2.4. Keyword Matching: Enhancing Precision in Incident Management 26](#_Toc195612487)

[2.4.1. Applications of Keyword Matching 27](#_Toc195612488)

[2.4.2. Benefits of Keyword Implementation 29](#_Toc195612489)

[2.4.3. Challenges of Implementing Keyword Matching 30](#_Toc195612490)

[2.4.4. Critical Analysis and Identification of Gaps in Keyword Matching 31](#_Toc195612491)

[2.4.5. Conclusion 32](#_Toc195612492)

[3. Analysis & Design 33](#_Toc195612493)

[3.1. Introduction 33](#_Toc195612494)

[3.2. Requirements Gathering 33](#_Toc195612495)

[3.3. Functional Requirements 34](#_Toc195612496)

[3.4. Non-Functional Requirements 35](#_Toc195612497)

[3.4.1. Constraints and Assumptions 36](#_Toc195612498)

[3.5. System Design 37](#_Toc195612499)

[3.5.1. High Level Architecture 38](#_Toc195612500)

[3.5.2. Data Flow 39](#_Toc195612501)

[3.5.3. Module Description 39](#_Toc195612502)

[3.5.4. Integration with NLP Techniques 40](#_Toc195612503)

[3.6. Software Design 41](#_Toc195612504)

[3.7. Database Design 43](#_Toc195612505)

[3.7.1. Relationships and Data Flow 46](#_Toc195612506)

[3.8. User Interface Design 47](#_Toc195612507)

[3.9. Security Design Considerations 51](#_Toc195612508)

[3.10. Summary 52](#_Toc195612509)

[4. Implementation 54](#_Toc195612510)

[4.1. Introduction 54](#_Toc195612511)

[4.2. Tools and Technologies Used 54](#_Toc195612512)

[4.3. Environment Configuration 58](#_Toc195612513)

[4.4. Database Implementation 63](#_Toc195612514)

[4.5. Feature Implementation 68](#_Toc195612515)

[4.6. Challenges Faced 76](#_Toc195612516)

[4.7. Conclusion 77](#_Toc195612517)

[5. Testing / Results 79](#_Toc195612518)

[5.1. Introduction 79](#_Toc195612519)

[5.2. Non-Functional Test Cases 80](#_Toc195612520)

[5.3. Findings of Non-Functional Testing 85](#_Toc195612521)

[5.4. Unit Testing of Functional Requirements 86](#_Toc195612522)

[5.5. Functional Test Cases 86](#_Toc195612523)

[5.6. Findings of Functional Testing 91](#_Toc195612524)

[5.7. Test Data 92](#_Toc195612525)

[5.8. Summary of Findings 100](#_Toc195612526)

[5.9. Conclusion 101](#_Toc195612527)

[6. Future Work and Research 103](#_Toc195612528)

[Table of Figures 105](#_Toc195612529)

[Bibliography 107](#_Toc195612530)

# Introduction / Proposal

**“Intelligent Incident Resolution System using Semantic Matching and NLP Techniques for Enhanced Support Efficiency”**

## Project Abstract

The current incident management processes often suffer from inefficiencies such as delayed response times, manual case handling, and poor communication across support teams, which negatively impact customer satisfaction. This project aims to develop an intelligent, automated incident resolution system that addresses these issues by leveraging semantic matching and Natural Language Processing (NLP) techniques. A custom web application will be created using Java, JavaScript, and MySQL. The system will use keyword matching algorithms and historical incident data analysis to automatically identify the team responsible for new incidents and issues. The system will streamline the resolution process, improve team communication, and reduce response times, thereby enhancing customer support efficiency. Additionally, the application will feature a user-friendly interface designed for seamless adoption by support teams, ensuring prompt and accurate incident resolution.

Natural Language Processing (NLP) techniques will be applied, including pre-processing of incident descriptions to remove stop words and normalise text. TF-IDF (Term Frequency-Inverse Document Frequency) will be used to weigh the importance of keywords, and cosine similarity will be employed to measure the similarity between new incidents and historical cases. This scoring system will help identify the most relevant matches for effective incident resolution. Once a high match percentage (70%+) has been determined, the support team member will engage the team responsible for the issue in the matched incident, thus reducing the resolution time that would have been previously spent engaging team that are not responsible for the issue.

## Research Context/Background

Incident management systems are essential for organisations to maintain operational efficiency and ensure high customer satisfaction. These systems are designed to track, manage, and resolve technical issues reported by users. However, traditional systems often rely on manual processes and lack advanced analytical capabilities, which leads to inefficiencies such as delayed response times and communication gaps between support teams (Atlassian, 2024; Software Engineering Institute, 2024).

To address these shortcomings, automated incident resolution systems utilise technologies such as machine learning (ML) and natural language processing (NLP). ML enables the system to identify patterns and trends in historical data, allowing for more effective classification and prediction of incidents. NLP, on the other hand, helps systems interpret the natural language used by users when describing issues, enabling more accurate keyword matching and case identification. These advancements have been shown to reduce response times and increase the accuracy of incident resolution (Blameless, 2024; ServiceNow, 2023).

For frontend development, JavaScript, in conjunction with HTML and CSS, will ensure a responsive and interactive user interface. MySQL will serve as the database management system, responsible for storing and retrieving the historical incident data necessary for the keyword matching and analysis process.

A user, whether a customer or a member of the sales or support team, will be able to submit an incident through the system by providing a short and long description of the issue. Once submitted, the system will assign an incident number (e.g., INC1234567), and the case will be routed to the support team's queue. A member of the support team will then access the incident by entering the incident number into a search bar. Using NLP-based keyword matching algorithms, the system will compare the new incident against older open and closed cases to find relevant matches. By identifying similar incidents, the system will suggest resolutions based on historical data, leading to faster incident resolution and improved team communication (BigPanda, 2024; Blameless, 2024).

## Research Aim

The primary aim is to design and implement an integrated automated technical support system that leverages Salesforce for case management, combined with a custom-built application using Java, JavaScript, and MySQL. This system will streamline the incident resolution process by automating case tracking, analysis, and resolution. By integrating Salesforce’s robust case management features with a custom frontend and backend, the system will provide real-time updates and insights. This integration will ensure that incidents are efficiently managed, matched with historical data, and resolved accurately, ultimately enhancing the overall efficiency and effectiveness of technical support operations.

## Research Objectives

**Develop Backend with Java:**

**Objective:** To build a robust backend system capable of handling complex data processing, keyword matching, and interactions with external APIs.

**Details:** Create backend services in Java to process incident data, execute keyword matching algorithms, and interact with both the MySQL database and Salesforce API. Java’s concurrency support and extensive libraries will be utilised to handle high volumes of data and ensure reliable performance. The backend will process incoming incident reports, apply keyword matching algorithms to identify relevant past incidents, and interact with Salesforce to update case information.

**Design Frontend with JavaScript:**

**Objective:** To develop an interactive and user-friendly web interface.

**Details:** Use JavaScript along with HTML5 and CSS3 to build a dynamic and responsive user interface. JavaScript frameworks or libraries (e.g., jQuery) may be used to enhance the functionality and interactivity of the frontend. The frontend will allow users to input new incidents, view matched incidents, and manage resolutions. It will communicate with the backend through RESTful APIs to ensure seamless data exchange and real-time updates.

**Implement Database with MySQL:**

**Objective:** To design and manage a MySQL database for storing historical incident data and supporting efficient retrieval and analysis.

**Details:** Develop a MySQL database schema optimised for performance and data integrity. Implement indexing to speed up search queries and normalization techniques to ensure data consistency. The database will store historical incident data, which will be used by the backend for keyword matching and analysis. The schema will be designed to support complex queries and handle large volumes of data efficiently.

**Automate Incident Matching:**

**Objective:** To enhance the system’s ability to identify relevant past incidents and root causes using advanced algorithms, such as TF-IDF and Cosine Similarity.

**Details:** Implement keyword matching algorithms to compare new incidents with historical data. Natural Language Processing (NLP) techniques will be applied, including pre-processing of incident descriptions to remove stop words and normalise text. TF-IDF (Term Frequency-Inverse Document Frequency) will be used to weigh the importance of keywords, and cosine similarity will be employed to measure the similarity between new incidents and historical cases. This scoring system will help identify the most relevant matches for effective incident resolution.

**User Interface Design:**

**Objective:** To design a user interface that is intuitive, accessible, and supports efficient incident management.

**Details:** Develop a frontend interface that allows support team members to input incidents, view matched incidents, and manage resolutions. The interface will feature real-time feedback, incident categorisation, priority tagging, and progress tracking. It will be designed to be user-friendly, ensuring that support teams can easily navigate and utilise the system’s features.

**Testing and Validation:**

**Objective:** To ensure that the system performs as expected and meets user requirements.

**Details:** Conduct comprehensive testing, including unit testing to verify individual components, integration testing to ensure components work together, and user acceptance testing (UAT) to gather feedback from actual users. Testing will include scenarios that cover various use cases and edge cases to ensure the system’s robustness and reliability. Test-driven development (TDD) practices will be employed to maintain high code quality.

## Research Questions

1. **Benefits of Automation:**

**What impact does automation have on response times and customer satisfaction in technical support settings?**

**Explanation:** This examines the benefits of automation in improving response times and customer satisfaction by comparing automated systems with traditional manual processes.

**What challenges might arise when implementing an automated support system, and what strategies can be employed to address these challenges?**

**Explanation:** This explores potential obstacles in deploying an automated system, such as integration issues, user adaptation, and data accuracy, and suggests strategies to mitigate these challenges.

**What metrics should be used to evaluate the effectiveness of the automated incident resolution system?**

**Explanation:** This focuses on identifying key performance indicators (KPIs) and metrics, such as resolution time, accuracy, and user satisfaction, to measure the systems effectiveness.

1. **System Design and User Experience:**

**What are the key features that contribute to a user-friendly interface for technical support applications?**

**Explanation:** This identifies essential features that enhance usability, such as intuitive navigation, responsive design, and accessibility features.

**How can the system ensure accurate keyword matching and historical data analysis to support efficient incident resolution?**

**Explanation:** This examines methods for improving the accuracy of keyword matching and data analysis, including algorithm selection and data pre-processing techniques.

**How can real-time feedback and progress tracking be incorporated to enhance the user experience?**

**Explanation:** This explores ways to implement features that provide users with real-time updates on incident status and progress, enhancing the overall user experience.

1. **Semantic Matching**

**How can semantic matching techniques be optimised to improve incident resolution in a support system?**

**Explanation:** This question investigates how semantic matching methods—such as NLP-based similarity algorithms—can be fine-tuned to enhance the accuracy and efficiency of matching new incidents with previously resolved cases. The focus is on leveraging contextual and linguistic similarities to suggest relevant solutions more effectively.

1. **Natural Language Processing**

**What are the most effective NLP techniques for preprocessing incident descriptions to improve keyword matching accuracy?**

**Explanation:** This question investigates the best NLP techniques for preprocessing text data from incident descriptions to enhance the accuracy of keyword matching, which is crucial for effective incident resolution.

1. **Combining Semantic Matching and NLP**

**How can semantic matching be effectively combined with NLP techniques to enhance the performance of an automated incident resolution system?**

**Explanation:** This question explores how semantic matching approaches can be integrated with natural language processing techniques to improve the system's ability to understand and compare incident descriptions. The focus is on enhancing the relevance and accuracy of suggested solutions by leveraging both contextual similarity and linguistic analysis.

## Proposed System Architecture

The proposed system architecture integrates several key technologies and components to deliver an intelligent incident resolution system. This architecture ensures seamless communication between different parts of the system, provides robust data management, and maintains high security. The components include a user interface, backend server, database, and authentication mechanisms. The following detailed description provides a comprehensive view of how these components interact within the system.

**Components and Interactions**

1. **User Interface (Frontend)**

**Web Browser:**

**Description:** The primary access point for users to interact with the system.

**Technologies:** HTML5, CSS3, JavaScript.

**Libraries/Frameworks:** React, Angular, or Vue.js.

**Functionality:** Users can submit incident reports, view matched incidents, and manage resolutions through a responsive and interactive interface.

1. **Backend Server (Application Layer)**

**Java Application Server:**

**Description:** Manages server-side logic, data processing, and interactions with external systems.

**Technologies**: Java, Spring, Hibernate**.**

**Functionality:**

* Business Logic: Processes incident reports, applies keyword matching algorithms, and handles data synchronisation.
* RESTful API: Exposes endpoints for frontend communication, including operations like submitting incidents, retrieving data, and updating resolutions.
* Integration with Salesforce: Utilises Salesforce API to synchronize case data, retrieve historical incidents, and update case statuses.

1. **Database (Storage Layer)**

**MySQL Database Server:**

**Description:** Stores historical incident data and supports efficient retrieval and analysis.

**Technologies:** MySQL.

**Functionality:**

* Schema Design: Includes tables for incident data, user information, and historical records. Implement indexing for fast search queries and normalization for data consistency.
* Operations: Manages CRUD operations (Create, Read, Update, Delete) for incident records and supports complex queries for keyword matching.

1. **Authentication and Authorisation**

**OAuth:**

**Description:** Manages secure authentication and authorisation for accessing the Salesforce API and backend services.

**Technologies:** OAuth 2.0.

**Functionality:**

* Secure Access: Handles user permissions, ensures that only authorised users can access or modify case data.
* Token Management: Issues and validates access tokens for API interactions, protecting sensitive information and enhancing security.

1. **Communication Flow**

**Frontend to Backend:**

**Description:** The frontend communicates with the backend server using RESTful APIs.

**Details:** Requests include submitting incident reports, fetching historical incident data, and retrieving suggested resolutions. Responses include status updates, matched incidents, and error handling.

**Backend to MySQL Database:**

**Description:** The backend server interacts with the MySQL database to store and retrieve historical incident data.

**Details:** The backend executes SQL queries to perform CRUD operations, supports complex searches for keyword matching, and ensures data integrity through indexing and normalization.

**OAuth Integration:**

**Description:** OAuth secures interactions between the frontend, backend, and Salesforce.

**Details:** Manages user authentication, authorisation, and secure token exchanges to ensure that API calls are performed by authorised users only.

## Proposed Technologies Used

**Java:**

Java will be utilised for backend development to manage server-side logic, data processing, and interactions with external systems such as Salesforce and MySQL. Java’s robustness and support for concurrency make it well-suited for handling complex and high-volume data operations. Key libraries and frameworks, including Spring and Hibernate, will facilitate the development of backend services. These services will include the implementation of keyword matching algorithms and interactions with both the database and Salesforce.

**JavaScript, HTML, and CSS:**

JavaScript, along with HTML5 and CSS3, will be used to build a dynamic and responsive user interface. JavaScript frameworks or libraries such as React, Angular, or Vue.js will enhance the interactivity and functionality of the frontend. The frontend will communicate with the backend using RESTful APIs, enabling seamless data exchange and real-time updates between the user interface and backend services.

**MySQL:**

MySQL will serve as the database management system for storing and managing historical incident data. The database schema will be designed for optimal performance, including indexing and normalization techniques to support efficient data retrieval and analysis. MySQL’s reliability and scalability make it ideal for handling large datasets and complex queries essential for the keyword matching and incident resolution processes.

**OAuth**:

OAuth will be employed to manage secure authentication and authorisation when accessing the Salesforce API. This protocol will ensure that only authorised users can access or modify case data, thereby protecting sensitive information and enhancing the overall security of the system. OAuth will be implemented to handle user permissions and secure API interactions.

**RESTful API:**

RESTful APIs will be used to enable communication between the frontend and backend, as well as between the backend and Salesforce. This approach ensures efficient and scalable data exchange. RESTful APIs will handle various operations such as submitting incidents, retrieving historical data, and updating resolutions. The use of RESTful APIs will facilitate smooth interaction between different components of the system and support real-time updates.

## Research Methods

To ensure a well-rounded and effective development process, a combination of research methods was used. Each method contributed to different aspects of understanding, designing, and evaluating the incident resolution system.

**Literature Review:**

The research began with a detailed literature review aimed at building a solid understanding of existing technologies and methodologies used in incident management, automation, and keyword or semantic matching. This involved analysing scholarly articles, industry reports, and case studies to explore current approaches and best practices in technical support systems. Special attention was given to previous research on matching algorithms and automated helpdesk tools. This helped address the core research question: What are the most effective methods for keyword matching in incident reports?

**Interviews and Surveys:**

To gather practical insights and understand real-world needs, interviews and surveys were conducted with members of the Dell support team. These conversations provided valuable input on the current pain points faced by support staff and what they expect from an automated resolution system. This method helped shape system requirements and addressed the question: What information needs to be presented to support teams for effective incident resolution?

**System Analysis:**

A system analysis phase was conducted to evaluate the feasibility and structure of the proposed solution. The analysis focused on how effectively the system could be integrated into an existing technical support workflow while meeting performance and scalability requirements. A SWOT analysis was performed to assess the strengths, weaknesses, opportunities, and threats associated with similar incident management systems. The insights gained from this analysis helped shape the system’s design and addressed the central question: How can a custom-built technical support solution be optimised for usability, efficiency, and long-term maintainability?

**Prototyping and Testing:**

Early versions of the system were developed and iteratively tested with users to validate both functionality and usability. Feedback was collected at each stage to refine the user experience and improve accuracy in matching incidents to historical data. This hands-on approach allowed the team to answer: How can the system ensure accurate keyword matching and historical data analysis for incident resolution?

**Implementation and Evaluation:**

Finally, the system was implemented in a controlled environment where its real-world performance could be assessed. Metrics such as resolution time, matching accuracy, and user satisfaction were used to measure its impact. User feedback—both quantitative and qualitative—helped evaluate how well the system performed in practice, particularly in response to the research question: What impact does automation have on response times and customer satisfaction in technical support

# Literature Review

## Introduction

As IT service management evolves, factors such as customer care, operational efficiency, and timely incident resolution are increasingly viewed as key components of effective support delivery. However, manual handling in the management of incidents is becoming more and more redundant as the evolution of IT systems requires resolving numerous and demanding incidents in a timely manner. This leads to long response times, poor issue resolution and poor coordination of various support teams, which in the end results in reduced customer satisfaction and quality of service.  
  
Yet, some expectations for better system functionality apply to everyone waiting for integration processes in computer systems. Such developments include Machine Learning (ML) and Natural Language Processing (NLP) technologies that are expected to take the automation of incident response systems to another level. This literature review discusses the implementation of ML and NLP as essential elements in the creation of an intelligent automated incident resolution system capable of correcting the above-mentioned deficiencies. The said system is expected to not only enhance the efficiency resolutions, but also the resolution communication of the support teams and the time taken to respond to incidents in the first place.

Incident management systems are critical for organisations to effectively address and resolve user-reported issues. The traditional approaches, however, are full of challenges, including dependency on human input and the varying standard in service quality. As reported by recent industry studies, such inefficiencies can lead to a “detrimental impact on customer loyalty and trust”, Pizzutti, C., & Fernandes, D. (2010). In response to these challenges, there has been a shift towards leveraging advanced analytical technologies i.e., the incorporation of ML and NLP techniques to innovate and improve these systems.

Machine learning offers robust capabilities for pattern recognition and predictive analytics, which can be used to anticipate common incidents and automate their resolution. Similarly, natural language processing can interpret and process user-generated data, enabling systems to handle initial incident triage automatically. This dual approach not only promises enhanced operational efficiency but also a reduction in the time taken to resolve incidents.

This review will systematically explore the dual application of ML and NLP in incident resolution systems. It will cover the following key areas:

**The Application of Machine Learning in Incident Management:** Focusing on how ML algorithms have been utilised to predict and classify incidents, enhancing the proactive capabilities of support systems.

**Natural Language Processing for Incident Data Analysis**: Examining how NLP facilitates the understanding and categorisation of user-reported incident data, thereby optimising the incident handling process.

**Integration of Keyword Matching in Incident Resolution:** This section will delve into studies where Keyword Matching has been integrated, providing insights into how Keyword Matching can speed up incident resolution processes.

This literature review sets out to achieve three goals which are to evaluate the past and present studies focusing on the convergence of ML and NLP within incident resolution systems, obtain best practices, and outline any unexplored areas. The review guarantees that the first step consists of general consideration of the technologies used in the management of incidents, and only then – the possibilities of their integrated use. Due to confidentiality reasons, it is difficult to obtain journals that explain the processes within IT companies in the tech industry. However, the journals chosen contain methods and practices that are widely used in these companies, despite not being the exact ones.

The system provides support for the viewpoint that it is necessary to improve the design of the system in question as well as makes it possible to achieve the benefits in support efficacy and efficiency.

### Overview of Traditional Systems and Limitations

Incident management systems are integral to the operational stability of organisations, particularly in IT environments. Incident management systems have been designed and structured around manual processes, where human interaction plays a vital role in every step – from logging incidents, to handling of the issue, and finally, the resolution. Typically, a support request is made through telephone, email, an internet form, or through an online portal, and subsequently recreated through a ticketing system. Each incident is categorised, prioritised, and assigned to appropriate teams, relying heavily on the judgement and availability of support team members.

However, these traditional systems often face several limitations:

**Scalability**: As the volume of incidents grows, manual systems struggle to scale effectively, often leading to delays and bottlenecks.

**Consistency**: Human involvement can lead to variability in how incidents are handled, affecting the consistency and predictability of service quality.

**Response Time**: Manual entry and processing of incidents often delay response times, which can be critical in high-stakes environments where downtime directly correlates with revenue loss.

**Error Rate**: The likelihood of errors in incident categorisation and routing increases with human handling, which can lead to misdirected incidents and prolonged resolution times.

These limitations highlight the inefficiencies within traditional incident management frameworks, thus prompting a necessary shift towards more sophisticated, technology-driven solutions.

### The Need for and Importance of Automating the Incident Management Process

The automation of incident management processes presents a compelling advantage, primarily addressing the limitations of traditional systems, particularly due to the manual handling of incidents aspect. Automation in incident management is not just about replacing manual tasks with technology; it's about enhancing the capabilities of the system to handle incidents more effectively and efficiently. Key areas where automation can significantly impact include:

**Speed**: Automated systems can process incidents almost instantaneously, reducing response times and accelerating resolution.

**Accuracy**: By using algorithms and predefined rules, automated systems can categorise and route incidents with high accuracy, ensuring that they are addressed by the appropriate personnel without the errors typically associated with manual handling of the incidents.

**Scalability**: Automated systems can handle a larger volume of incidents without an increase in support staff, making them ideal for growing organisations.

**Data Utilisation**: Automation enables the integration of advanced analytics into incident management, allowing organisations to use incident data to identify trends, predict potential issues, and implement preventative measures.

Additionally, these systems can be further improved by integrating Machine Learning with Natural Language Processing technology. Machine Learning Algorithms can learn from historical event data to continually improve the classification process. At the same time, NLP can be used to automatically interpret and categorise the free-form text of incident descriptions automatically, facilitating more nuanced understanding and processing.

The shift towards automation using ML and NLP is driven by the need to not only maintain, but also improve service quality, with the increased complexity and demands in the IT environment. This evolution represents a significant shift from manual labour-intensive methods to more intelligent, data-driven strategies that promise not only to streamline operations, but also to significantly improve service outcomes for customers and internal stakeholders alike.

## The Role of Machine Learning in Enhancing Incident Management

### Introduction to Machine Learning (ML)

Machine learning (ML) is a subset of artificial intelligence that focuses on developing systems that can learn and make decisions based on data. Essentially, ML provides programming algorithms that can process input data, remember patterns and properties and learn from this information. The ability to learn and adapt without explicit programming of individual decision processes that use this known knowledge to make decisions about data sets ML apart from traditional software-defined systems. This section reviews multiple key studies that explore ML techniques and their impact on the automation and optimisation of incident management processes.

**Types of ML**

Machine Learning methodologies are broadly categorised into three main types, each with unique capabilities and applications in incident management:

**Supervised Learning**: This involves training a model on a labelled dataset, where the input data is tagged with the correct output. The model learns by comparing actual results with learned results to find errors and adjust the model accordingly. in event management Supervised learning is used to intensify or reclassify events based on correctly classified historical data

**Unsupervised Learning**: In unsupervised learning, the data used to train the model is unlabelled, that is, the model automatically tries to identify patterns and relationships in the data. This approach is useful for grouping events of the same type and identifying anomalies or outliers in event data.

**Reinforcement Learning**: This type of learning uses a system of rewards and punishments to force the model to make decisions that maximise cumulative rewards. Reinforcement learning is especially effective in dynamic environments where conditions change over time, such as adapting to new types of events. or developing attack vectors in cyber security management.

**Core Machine Learning Techniques**

**Classification and Prediction**: Techniques such as decision trees, vector support machine and neural networks are used to classify data into predefined categories. For instance, a neural network might be trained to predict which incidents are likely to escalate based on features extracted from the incident report and historical data.

**Semantic Analysis,** which focuses on extracting and interpreting the meaning of textual data, is often incorporated into these classification models—particularly when dealing with incident descriptions. When paired with NLP or embedding models like Word2Vec or GloVe, semantic analysis helps enhance the model’s ability to classify incidents more accurately based on contextual understanding rather than just keywords.

**Clustering and Anomaly Detection**: Clustering and Anomaly Detection: Methods such as k-means clustering, and hierarchical clustering are used to divide event reports into groups of similar cases, or to detect unusual patterns that deviate from the norm.

**Relevance to Incident Management**

**Automated Triage**: Machine learning models can automatically test incoming incidents, prioritise and route it to the appropriate response team based on known patterns and features. This not only speeds up the response process, but it also helps ensure that incidents are handled by the most appropriate personnel, improving resolution times and outcomes.

**Predictive Maintenance:** By analysing trends and patterns from historical incident data, ML can predict potential future failures or issues, enabling proactive maintenance and interventions. This application of predictive analytics helps organisations minimise downtime and maintain high service levels.

**Integration with Existing Systems**

Machine learning can be seamlessly integrated with existing IT infrastructure and incident management systems to increase capabilities. By integrating ML models with these systems, organisations can automate complex decision-making processes, reduce reliance on manual selection and improve the accuracy of the incident management process without the need for an extensive system overhaul.

### Applications of Machine Learning

**Automated Incident Categorisation**: Silva et al. (2024) uses Support Vector Machines (SVM) to automatically categorise incident tickets. Their system divides incidents into categories based on their severity and type, which helps with faster and more appropriate responses. This system has been tested in a real-world environment, and has resulted in an accuracy of 89%, greatly reducing human effort in manually categorising incidents. They note, “The application of SVM in automating ticket categorisation improved classification accuracy by over 30% compared to traditional methods”. (Silva, Pereira, and Ribeiro, 2024). As seen in the charts below, Silva et al. (2024) compared two different methods in the categorisation of incident tickets, KNN and SVM, with the SVM model showing higher results of overall accuracy after 3 approaches.

A graph of accuracy results

Description automatically generatedEin Bild, das Text, Screenshot, Reihe, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Figure 2.1 Accuracy Results (SVM) Silva et al. (2024)

Figure 2.2 Accuracy Results (KNN), Silva et al. (2024)

A graph of overall accuracy

Description automatically generatedPrihandono et al. (2020) explores the use of Long Short-Term Memory (LSTM) networks – a type of neural network capable of learning patterns over time and handling sequential data - for predicting IT incidents. They found that "LSTM networks, by analysing historical incident data, could predict potential incidents with a precision rate of 98.866%, significantly reducing unplanned downtime" (Prihandono et al., 2020).

Figure 2.3 Overall Accuracy with SVM & KNN, Silva et al. (2024

Based on comparison experiments they conducted on multiple Machine Learning models; they found that the LSTM model performed the best and had the highest accuracy in predicting potential incidents.



Figure 2.4 Comparison of models, Prihandono et al. (2020)

**Enhanced Decision Support**

In their study on decision-making in incident management, Li and Zhan (2012) discuss how machine learning algorithms, such as KNN, SVM and LSTM, can assist in making informed decisions by providing predictive insights and recommendations based on data trends (Li and Zhan, 2012).

### Challenges of Implementing Machine Learning

**Data Quality and Integration Issues**

Silva et al. (2024) highlights the challenge of data quality, stating that "the effectiveness of ML models is heavily contingent upon the quality and completeness of the input data, which often hampers the model's performance in real-world settings" (Silva et al., 2024).

Li and Zhan (2012) address integration complexities, noting that "integrating advanced ML models into existing IT infrastructure poses significant challenges, particularly when these systems are not designed to handle dynamic learning processes" (Li & Zhan, 2012).

**Maintenance and Scalability Concerns**

As reported by Prihandono et al. (2020), maintaining the accuracy of ML models over time requires continuous data training and model tuning, which can be resource intensive. (Prihandono et al., 2020).

### Benefits of Machine Learning in Incident Management

**Operational Efficiency**

Silva et al. (2024) confirmed that “Machine Learning can significantly reduce the time required to classify and route events. This leads to quicker resolutions and increases overall efficiency” (Silva et al., 2024).

According to Prihandono et al. (2020), ML-enabled predictive maintenance can reduce downtime by up to 50%, which has a direct impact on operating costs and service quality” (Prihandono et al. al., 2020).

**Improved Accuracy and Consistency**

Li and Zhan (2012) emphasise that "ML models offer consistent decision-making, minimizing human errors and bias in incident handling processes" (Li & Zhan, 2012).

### Comparative Analysis Across Studies

The application of different machine learning techniques in incident management highlights the importance of choosing the right algorithm based on specific needs and data characteristics. Silva et al. (2024) includes insights into the suitability of SVM for classification tasks, noting that “SVM’s ability to effectively handle high-dimensional data makes it particularly adept at classifying complex incident reports where multiple variables must be considered simultaneously” (Silva et al., 2024).

In contrast, LSTM's strength lies in its predictive ability, which is necessary for predicting future events based on sequential data models. Prihandono et al. (2020) highlights the usefulness of LSTM in predictive analytics, stating that “LSTM models outperform other algorithms when it comes to analysing time series data. This makes it invaluable for predicting incident trends and preparing proactive responses” (Prihandono et al., 2020). This is especially useful in IT environments where understanding the evolution of incidents is critical. Over time this can lead to more effective and timely intervention.

A further comparative analysis done by Prihandono et al. (2020) shows the practical implications of choosing between these models: “Although SVM provides fast and efficient classification, LSTM provides insights that go deeper into the long-term dynamics of incidents, which may be important for strategic planning and resource allocation” (Prihandono et al., 2020). These differences emphasise that the choice between SVM and LTSM, (Figure 4), should consider not only the immediate needs of incident categorisation, but also the strategic goals of incident prediction and management.

These insights collectively show that the effectiveness of ML techniques in incident management depends heavily on their alignment with specific operational goals and the available data. Therefore, the decision to use SVM or LSTM should be guided by an analysis of incident management needs, detailed data types and desired results of the ML deployment.

### Critical Analysis and Gaps

Although Machine Learning technology has led to significant advancements in incident management, several important points deserve a deeper and more conclusive investigation. The performance of machine learning models such as SVM and LSTM depend heavily on the quality and quantity of available data. In practice, the variability and inconsistency of incident data can seriously hinder the accuracy of these models. Silva et al. (2024) acknowledged that although SVM is accurate in classification under appropriate conditions, real-world applications can be limited by noisy or incomplete data (Silva et al., 2024). Additionally, the complexity of ML models often results in 'Blackbox' situations, where the decision-making process is not transparent. This makes it difficult for operators to trust and verify results.

A notable gap in the current literature is the lack of discussion on integration costs and technical challenges involved in updating ML models as new types of phenomena emerge. The mechanism of continuous learning is hardly mentioned. This is important for maintaining the relevance and effectiveness of ML systems. In addition, ethical considerations such as data privacy, and potential bias in algorithmic decision making is not explored in depth. These characteristics are important because they affect the acceptability and scalability of ML solutions in sensitive environments such as incident management within large-scale companies.

## Natural Language Processing (NLP) in Incident Management

Natural language processing combines many computational techniques aimed at the intersection of language and machines. This allows the system to interpret, analyse, and even create meaningful human language.

To understand how NLP enables these capabilities, it is important to examine the typical stages involved in processing human language:

* **Text Preprocessing:** This initial step involves cleaning the input text by removing punctuation, converting all words to lowercase, removing stop words (e.g., “the”, “and”), and breaking down text into smaller units called tokens. Techniques such as stemming or lemmatisation are applied to reduce words to their root form.
* **Part-of-Speech (POS) Tagging:** Each word is labeled with its grammatical category, such as noun, verb, adjective, etc. This helps the system identify sentence structure and understand syntactic relationships.
* **Named Entity Recognition (NER):** NLP models identify important named entities in the text such as company names, product names, error codes, or departments. This is especially useful in incident management for recognising context-specific details.
* **Dependency Parsing:** This step analyses grammatical structure to understand how words are related, allowing the system to infer subject–verb–object relationships and extract deeper context.
* **Vectorisation:** Once text is preprocessed, it is converted into numerical format so it can be understood by machine learning models. Common vectorisation methods include TF-IDF, Bag of Words, and word embeddings like Word2Vec, GloVe, and BERT.
* **Semantic Analysis:** This final step focuses on extracting the meaning and intent behind the text. It includes techniques such as sentiment analysis, semantic similarity scoring, and named entity recognition, which help determine urgency, tone, and the specific components of an incident report. These capabilities are crucial for interpreting user-reported issues with greater accuracy.

These steps form the core pipeline of NLP applications, enabling incident management systems to move beyond surface-level keyword detection and toward understanding and acting upon human-generated text intelligently.

**Integrating NLP with Incident Management Systems**

**Automating Triage**: One of the main roles of NLP within an incident management system is to automate the triage process. Traditionally, triage has involved manually classifying incidents based on description, urgency, and type, a process that can be time-consuming and prone to human error. By integrating NLP, the system can automatically analyse incident descriptions using text analysis and sentiment analysis to determine incident severity and priority. For example, NLP algorithms can detect keywords or phrases such as “System outage”, “failure”, and “loss”, automatically escalating these incidents for immediate attention.

**Streamlined Communication Analysis**: NLP techniques can also be used to analyse communications between customers and support teams to identify common or recurring themes. This capability allows for more efficient management of customer interactions by highlighting areas that require systematic change or additional support. Sentiment analysis can also be used to gauge the tone and urgency of communications. This helps optimise responses and improve customer satisfaction.

**Contextual Understanding and Response:** In addition to keyword extraction, advanced NLP techniques such as named entity recognition, and contextual analysis can understand the specific context of an incident. This deeper understanding not only allows for more accurate classification of incidents, but it also offers possible solutions based on similar past incidents. For example, if an incident report mentions a specific error code or technical issue that has a well-documented resolution, the system can automatically send this information to the support team or even directly to the customer. This greatly reduces resolution time.

**Operational Efficiency and System Integration**

**Reducing Manual Overhead:** By automating significant portions of the incident handling process, NLP reduces the workload on support staff, allowing them to focus on more complex tasks that require human intervention.

**Improved Data Visualisation:** NLP integration enables incident management systemsto leverage vast amounts of unstructured data (such as incident details, customer feedback and support tickets) that cannot be easily analysed before NLP turns this data into actionable insights. This facilitates continuous improvement in service processes and incident handling protocols.

**Challenges and Seamless Integration Solutions:** Integrating NLP with existing incident management systems can present challenges. This is mainly related to data compatibility and system architecture optimisation. To meet these challenges, it is important to develop middleware, or API’s that can translate between the NLP processing modules and the existing system databases and user interfaces.Additionally, it is important to train the NLP models on specific organisational data to ensure they accurately reflect the language and nuances of the specific business environment in which it occurs.

### Applications of NLP

**Automated Text Summarisation and Categorisation**

Jianqiang Zhang et al. (2022) presented a complex cosine distance approach that improves the accuracy and efficiency of text similarity measurements. They applied their cosine distance method within a simulated environment that mimics real-world incident management systems to test the performance and scalability of their approach. This is important for automatic incident report classification, as Zhang et al. (2022) elaborates, "Our method not only enhances the precision of similarity detection but also speeds up the processing by optimizing vector calculations, thus making it highly suitable for real-time applications" (Zhang et al., 2022). During testing, they used a traditional cosine distance method and an improved version of the traditional method to compare results and compare the time taken to retrieve these results (See figure 5 below).

A table with numbers and text

Description automatically generated

Figure 2.5 Comparison Chart of Accuracy between Traditional Method and Improved Traditional Method, Zhang et al. (2022)

A diagram of a system

Description automatically generatedPattnaik et al., (2019) showed how cosine similarity and clustering can efficiently summarise Odia text documents. Their application of NLP techniques was tested in an educational setting where their goal was to manage and summarise academic papers and project reports. They discuss potential applications in incident management: “Our summarization technique reduces the amount of message data processed. It helps quickly evaluate initial incident reports by highlighting the most relevant information” (Pattnaik et al., 2019).

Figure 2.6 Pattnaik et al. (2019) Text Summarization Process

Gawhankar et al. (2024) examined the use of NLP for extracting detailed and accurate information from resumes. Their research was primarily conducted in the context of human resources, especially focusing on resume parsing for recruitment processing. They suggest parallel incident handling, stating that "Similar to parsing resumes, NLP can be utilized to dissect complex incident data, extracting key elements that aid in faster and more accurate incident categorization and response" (Gawhankar et al., 2024).

Gawhankar et al. (2024) developed a system for extracting information from resumes using advanced natural language processing techniques. Firstly, text from these documents is extracted, segmented, and then organised into categories such as personal details and professional experience. The system uses a range of different algorithms, such as, Decision Trees, Random Forests, and Logistic Regression (see Figure 6 below). This is done so that data from these resumes is classified by job roles and predicted salary ranges. This approach streamlines the resume screening process, which in turn aids in the evaluation and choosing of candidates. For a more detailed breakdown of the process steps, please refer to (Figure 7) below also.

### Benefits of Natural Language Processing in Incident Management

**Enhanced Operational Efficiency**

NLP technology improves many aspects of incident management, from initial report processing to decision making. This reduces the time and effort required for manual classification and analysis. Zhang et al. (2022) notes that the optimised method “Significantly reduces incident response time by automating the initial data processing steps” (Zhang et al., 2022).

Figure 2.7 Gawhankar et al. (2024) Proposed System for Resume Text Retrieval

**Improved Accuracy and Consistency**

By automating the process of separating and classifying, NLP helps ensure that incidents are captured and evaluated based on consistent criteria, thus reducing human error. Pattnaik et al. (2019) highlighted “Our approach standardizes the summary process. to ensure consistent results, which is critical to maintaining quality in incident management” (Pattnaik et al., 2019).

**Proactive Incident Handling**

Advanced NLP applications can predict potential issues by analysing trends and patterns in data. Gawhankar et al. (2024**)** discuss, "NLP-driven predictive models analyse past incidents and forecast potential future occurrences, enabling proactive measures rather than reactive responses" (Gawhankar et al., 2024).

Figure 2.8 Systems Steps Explained, Gawhankar et al. (2024)

### Challenges of Implementing NLP Techniques

**Complexity in Language Processing**

NLP systems must handle a wide variety of linguistic expressions and technical terminology. This is a great challenge in multilingual environments. Zhang et al. (2022) discusses the complexity of discuss the complexity of adapting NLP to different languages ​​and dialects. This affects operations in the global incident management system (Zhang et al., 2022).

**Data Quality and Diversity**

Effective NLP implementation requires high-quality, structured data. Pattnaik et al. (2019) emphasises that "inadequate data quality severely affects the performance of NLP systems, leading to poor outcomes in automated text processing" (Pattnaik et al., 2019).

### Critical Analysis and Identification of Gaps

Despite considerable advancements in NLP technology, several significant gaps remain that limit its effectiveness in an incident management setting.

While improvements such as those introduced by Jianqiang Zhang et al. (2022) increase processing speed, there is still a significant need for these systems to manage huge amounts of data through various management platforms. Zhang et al. noted that “"optimizing processing speed without sacrificing accuracy remains a challenge in deploying NLP systems in real-time environments" (Zhang et al., 2022)

The ability to understand the context and semantic nuances of the language used in incident reports is still lacking. Pattnaik et al. (2019) notes difficulties in handling complex expressions: "The system sometimes fails to accurately interpret the contextual meanings embedded within technical jargon" (Pattnaik et al., 2019). This evidently highlights the need for more sophisticated NLP models which can handle language nuances specific to incident management.

Although English has become the worlds centralised language, a significant gap still exists in the development of NLP systems capable of processing multilingual data efficiently. Again, most existing NLP technologies focus on English, which is not enough in the growing technological world, especially in the increasingly global scope of incident management.

These gaps suggest that while NLP could revolutionise incident management and increase processing capabilities, there is still a lot of room for improvement when it comes to contextual understanding and multi-language support, which are necessary for widespread adoption and effectiveness. Addressing these gaps will not only improve the performance of NLP systems but also expand their applicability in diverse and dynamic operational settings.

## Keyword Matching: Enhancing Precision in Incident Management

Keyword matching refers to the process of identifying specific terms or phrases from predefined keyword lists within text data. This enables incident management systems to perform tasks such as classification, routing, and historical lookup based on textual input. It is important to distinguish keyword matching from keyword extraction, which involves identifying the most relevant or meaningful words from unstructured text without predefined terms. Keyword extraction helps summarise or label new content, whereas keyword matching operates by comparing input text to a known vocabulary for decision-making or filtering purposes.

**Algorithms and Techniques Used in Keyword Matching**

Several well-established algorithms are used for efficient and accurate keyword matching:

* **Aho-Corasick Algorithm**: A multiple-pattern matching algorithm that builds a trie (prefix tree) of keywords and enables simultaneous search for all terms in a single scan of the text. It is ideal for high-speed matching in incident logs or chat transcripts.
* **Knuth-Morris-Pratt (KMP) Algorithm**: Used to search for substrings within a main text efficiently by skipping unnecessary comparisons based on partial match information.
* **Bitap (Shift-Or) Algorithm**: Supports approximate (fuzzy) matching, allowing small edits like typos or spelling variations, which is helpful in real-world incident text where users often write informally.

In modern NLP-enhanced systems, keyword matching is often combined with semantic techniques like:

* **TF-IDF (Term Frequency-Inverse Document Frequency)**: Calculates how important a word is to a document in a corpus. It's useful for ranking keywords and filtering noise.
* **Cosine Similarity**: Measures the similarity between two text vectors (e.g., incident descriptions) and is often used in combination with TF-IDF or word embeddings for contextual keyword matching.

### Applications of Keyword Matching

**Enhanced Text Classification for Incident Management**

Shetty et al. (2024) discusses the integration of keyword extraction with deep learning techniques to enhance text classification within incident management systems. They developed a model called 'DeepText' which combines deep neural networks with keyword-focused algorithms to improve the accuracy of incident classification. Shetty et al. describes their approach: "DeepText utilizes a combination of convolutional neural networks and a tailored keyword extraction algorithm to refine text classification, providing a robust solution for managing complex incident data" (Shetty et al., 2024).

A diagram of text classification system

Description automatically generated

Figure 2.9 Levels in text classification systems, Shetty et al. (2024)

Eckstein et al. (2016) explores the use of machine learning to improve incident management by extracting customer needs from service tickets. Explaining their approach: "Our model categorizes incident tickets into a domain-specific selection of customer needs, which helps in providing proactive solutions and improves the responsiveness of service systems" (Eckstein et al., 2016).

The flowchart below (Figure 10), shows their method for processing B2B IT service tickets to extract customer needs. It starts with identifying the needs of these businesses and creating tickets based on these needs. The tickets are then sorted into tables, followed by optional random sampling and selection by a Chi-square test. After text pre-processing, the data goes through an over-sampling process, before being analysed with supervised learning models. The final step involves evaluating these models using specific metrics to check the effectiveness of the classification system, ensuring consistent handling of IT service requests.

A diagram of a process flow

Description automatically generated**Enhanced Keyword Extraction for Scientific Articles**Komang Rinartha and Luh Gede Surya Kartika (2022) explore the combination of word frequency and RAKE (Rapid Automatic Keyword Extraction) to enhance keyword extraction from scientific articles. They state, "This combination is implemented using python programming language and tested using several articles taken randomly from the internet" (Rinartha & Kartika, 2022).

Figure 2.10 Eckstein et al. (2016) Approach design

The python code below (Figure 2.11), shows the implementation code to calculate word frequencies in a block of text. It counts and ranks words based on their occurrence, allowing the extraction of terms that could be important in understanding key themes and topics in the context of the inputted data.

A white screen with black text

Description automatically generatedSiddiqi and Sharan (2015) provide details about various automatic keyword extraction and keyword extraction techniques. Their survey highlights the application of these methods to organising large amounts of textual data, which is required for retrieving search data indexing and summarising effectively. They

Figure 2.11 Rinartha and Kartika (2022) Word Frequency Algorithm

investigated how keyword extraction facilitates the analysis of large textual datasets by providing representative words that efficiently summarise documents.

**Automated Customer Claim Registration**

Beyranvand and Aytekin (2020) discuss the application of text mining and machine learning to automate customer claim registrations in call centres. They explain that their method has greatly accelerated the processing of insurance claims. and improve accuracy by automatically assigning the correct category to each claim based on the text description provided by the customer. This automation helps route claims to the appropriate department faster, thus improving the overall efficiency of the process.

The solution architecture diagram by Beyranvand and Aytekin (2020), (see Figure 12 below), shows an automated systems for handling customer queries. It involves an offline phase where data is manually labelled and stored. In the online phase, new queries are processed, and the top 3 contact reasons are ranked by the classifier. Continuous updates to the classifier through a learning procedure make sure improved accuracy in identifying contact reasons, streamlining the response process.

A diagram of a data base

Description automatically generated

Figure 2.12 Solution Architecture, Beyranvand and Aytekin (2020)

### Benefits of Keyword Implementation

Keyword matching enhances the precision of incident categorisation and prioritisation. Shetty et al. (2024) shows how integrating keyword extraction with deep learning can yield a high accuracy rate, improving the relevance and timeliness of incident responses. This precise categorisation ensures that resources are directed appropriately, minimising wastage and enhancing response effectiveness.

Using keyword matching tools can greatly increase operational efficiency. As Eckstein et al. (2016) comments:  "By automating the extraction of customer needs, we reduce reliance on manual processes, thereby speeding up the response time and increasing the accuracy of service delivery" (Eckstein et al., 2016).

**Improved Accuracy and Efficiency**

Studies show that combining word frequency with RAKE leads to more accurate keyword extraction. This greatly increases the efficiency of processing scientific text. Rinartha and Kartika noted that "The application could produce some proper keywords in the form of phrases and words. It is better than the keywords that resulted with one method either word frequency or RAKE" (Rinartha & Kartika, 2022).

The use of keyword and key phrase extraction techniques significantly improves information extraction and document cataloguing systems. As discussed by Siddiqui and Sharan (2015), these techniques allow for more structural analysis and text classification and improve information access and usability on various digital platforms.

**Increased Efficiency and Customer Satisfaction**

The use of automation for claims registration dramatically improves the efficiency of call centre operations. Beyranvand and Aytekin highlight that "Use of text mining and machine learning techniques will increase the customer satisfaction and endows the call centre staff with better ways to help the customer" (Beyranvand & Aytekin, 2020). This system not only speeds up the process but also improves the accuracy of claims classification. This leads to higher customer satisfaction.

### Challenges of Implementing Keyword Matching

**Linguistic and Semantic Understanding**

Dealing with the variability and complexity of language is a major challenge. The keyword extraction system sometimes struggles with accurate interpretation of context, slang, and technical jargon. This can lead to errors in information processing and decision making, as Shetty et al. (2024) noted "Our models often misinterpret slang and technical jargon, which are critical in incident reports, leading to suboptimal categorization" (Shetty et al., 2024). This challenge emphasises the need for continued development of more complex NLP algorithms that adjust and interpret more efficiently.

Eckstein et al. (2016) emphasises the complexities associated with language variation in ticketing tasks: "The primary challenge in automating need extraction lies in the system’s ability to interpret complex, technical, or poorly structured language, which can often lead to errors in categorization and response" (Eckstein et al., 2016).

**Complexities in Handling Linguistic Nuances**

Rinartha and Kartika acknowledge the challenge of accurately compiling comprehensive scientific texts. This is especially true when dealing with complex linguistic data. They note, "RAKE is based on the observation that keywords often contain many words but rarely contain standard punctuation or stop words" (Rinartha & Kartika, 2022).

One of the primary challenges highlighted by Siddiqi and Sharan (2015) involves the linguistic and semantic complexities that arise during keyword extraction. The variability of language and the context-dependent nature of key phrases make it difficult to develop universally effective extraction algorithms without extensive customisation and refinement.

**Complexity of Accurate Category Determination**

Despite progress, the authors also acknowledge the challenge, especially in correctly interpreting customer claims. They noted the difficulty of call centre employees understanding and selecting as many as two hundred different categories. "The incorrect selection of the category can result in delaying the processing procedure and ultimately reduced customer satisfaction" (Beyranvand & Aytekin, 2020), emphasising the need to train accurate algorithms to handle diverse customer interaction.

### Critical Analysis and Identification of Gaps in Keyword Matching

Despite advances in keyword matching technology, many key areas remain undeveloped. These areas still require further research and improvement. One of the main limitations of the current system is its focus on separating keywords based on frequency and superficial context, often at the expense of the deeper meaning required for accurate interpretation and response. As stated by Shetty et al. (2024), although these systems can identify keywords efficiently, they often struggle to catch the hidden meaning. This can lead to discrepancies in automatic responses. This is especially true in complex incident management situations. This reinforces the need for improved semantic processing abilities that could enable more nuanced understanding and application of keyword extraction.

Eckstein et al. (2016) points out a significant limitation in their methodology: "While our system effectively identifies explicit customer needs, it often misses implicit cues that could suggest underlying issues not directly mentioned in the tickets" (Eckstein et al., 2016). They point out the need for improved models that can provide a more comprehensive analysis of customer needs and detect more subtle signals and trends.

**Need for Further Optimisation**

Rinartha and Kartika showed that although their method improved keyword extraction, there is still room for optimisation to enhance the algorithm's ability to handle various complex datasets. "Further analysis and development are needed to fine-tune the processes of combining word frequency and RAKE for optimal performance" (Rinartha & Kartika, 2022).

Siddiqui and Sharan (2015) identified a gap in keyword extraction research. There is a need to improve semantic understanding and contextual integration. They suggest that although current techniques are effective on a superficial level, there is considerable scope for developing methods that can interpret the deeper meanings of texts and adapt them to evolving language usage in different domains.

Beyranvand and Aytekin (2020) point out that although their system reduces the time it takes to expertly handle claims, there is an ongoing need to update the system to accommodate new types of claims and change customer interaction patterns. They suggest that "the required time to become proficient in identifying the correct contact reason is reduced because the task of choosing amongst hundreds of categories is reduced to choosing between a couple of relevant categories" (Beyranvand & Aytekin, 2020), but constant adjustment is required to maintain system performance.

### Conclusion

The research reviewed in this chapter shows how new technologies like Machine Learning (ML), Natural Language Processing (NLP), and keyword matching can improve how incidents are managed in modern IT systems. Machine learning, for example, can help automate tasks and support better decision-making by learning from large sets of past incident data. However, for machine learning to work well, it needs clean, accurate data and must be able to connect smoothly with existing systems. Ongoing maintenance is also important to make sure the system keeps working as things change over time.

Natural Language Processing (NLP) is another important area that shows a lot of promise. It can help by automatically sorting and summarising incidents, which saves time and improves how quickly issues are handled. The studies reviewed show how NLP can improve accuracy in retrieving information and speed up response times. Still, NLP faces some key challenges, like understanding different contexts, processing language quickly, and working well in multiple languages. These are important areas where further improvements are needed.

Finally, keyword matching is a simpler method that can help detect similar incidents based on specific terms. While useful, it has its own limits, especially when it comes to understanding the meaning behind different ways people describe issues. To be more effective, keyword matching systems need to be more flexible, better at handling complex language, and easier to adjust and integrate into different platforms.

Overall, these three areas—ML, NLP, and keyword matching—all have their strengths and challenges. Together, they show that while there is great potential in using technology to improve incident management, there is still a need for ongoing development to make these systems more reliable, intelligent, and adaptable to different environments and user needs.

# Analysis & Design

## Introduction

This chapter provides a comprehensive analysis and design of the Intelligent Incident Resolution System, focusing on the foundational elements necessary for its successful implementation. The analysis begins by defining the functional and non-functional requirements of the system, ensuring clarity on its expected behaviour and performance standards. This is followed by a review of the constraints and assumptions that inform the design, considering both technical and operational limitations.

The chapter then delves into the system design, outlining the high-level systems architecture and illustrating how the various layers—presentation, application, and data—interact to deliver the desired functionality. The data flow diagram highlights the movement of data through the system, from incident submission to resolution, ensuring a clear understanding of the processes involved. Additionally, the database schema provides a structured overview of how data will be stored and managed, ensuring integrity and accessibility.

To complete the design, wireframes for the user interface are presented, demonstrating how users and admins will interact with the system. These design considerations aim to create a solution that is user-friendly, scalable, and secure while meeting the project’s primary objectives of improving incident resolution efficiency and consistency.

## Requirements Gathering

The requirements for this project were gathered through a combination of research and technical planning. This process included reviewing existing literature, evaluating technology options, and assessing feasibility within the project timeline and available resources.

1. **Literature Review**  
   A comprehensive review of academic and technical literature on incident resolution systems was carried out to identify best practices, common system features, and areas where improvement was possible. These features included incident tracking, role-based access control, and keyword matching using natural language processing techniques like TF-IDF and cosine similarity. The review helped shape the core functional goals of the system.
2. **Technical Feasibility and Technology Selection**  
   Technology decisions were guided by both capability and familiarity. Java was selected for the backend due to its robust ecosystem, particularly the Spring Boot framework, which simplifies the implementation of authentication, authorisation, and RESTful APIs. It also aligns with the technologies I’ve used extensively in prior coursework, enabling faster and more confident development.

For the NLP component, TF-IDF vectorisation and cosine similarity scoring were implemented manually using Java, supported by Apache Commons Math for handling vector operations. This approach kept the NLP processing lightweight and customisable, avoiding the overhead of external NLP platforms.

In addition to TF-IDF, the project integrates GloVe word embeddings (specifically, the 50-dimensional version) to improve semantic understanding. The 50d GloVe model was chosen because it offers a good trade-off between computational efficiency and representational depth. Higher-dimensional embeddings (e.g., 100d or 300d) provide more nuanced context but significantly increase memory usage and processing time, which was not ideal for this scale of application. The 50d model allowed the system to capture meaningful semantic relationships between words while maintaining fast processing and low resource consumption.

MySQL was selected as the relational database due to its reliability, structured schema support, and seamless integration with Java through Spring Data JPA. The use of phpMyAdmin further supported ease of data management and debugging during development.

Finally, the frontend was built using HTML, CSS, and vanilla JavaScript. This decision was made to keep the user interface simple and responsive, focusing development efforts on the backend logic and NLP functionality rather than UI complexity.

Each technology was selected to ensure compatibility, maintainability, and performance, while staying within the project’s development scope and timeline.

## Functional Requirements

1. **User Authentication and Role Based Access Control**

* The system will implement a secure login mechanism to differentiate between standard users and admin users.
* Admins should have access to all incident management features, including viewing, assigning, and running NLP-based similarity searches.
* Users should be restricted to incident submission and status viewing.

1. **Incident Submission**

* Users must be able to submit new incidents through a web-based form that includes:
  + **Short Description**: A summary of the issue (e.g., “System error causing login failure”).
  + **Long Description**: A more detailed report that provides context and possible observations (e.g., “The system encountered an unexpected error during the authentication process…”).
* The system should generate a unique incident ID (e.g., INC1234567) for each submitted report and store it in the database.

1. **Incident Queue Management**

* Admins should see a list of new incidents in a dedicated section marked as “New Incidents.”
* The system should allow incidents to be categorised as “New,” “Assigned,” “Ongoing,” or “Closed,” depending on their status.
* Admins must be able to assign incidents to themselves or other admins.

1. **Incident Assignment and Status Updates**

* Once an admin assigns an incident, its status should automatically change from “New” to “Assigned.”
* The system should record the admin who has taken ownership of the incident and display their name in the incident details.

1. **NLP Keyword Matching and Similarity Search**

* Admins must be able to input an incident number to trigger a keyword search.
* The system should preprocess the incident text, calculate TF-IDF vectors, and run a cosine similarity comparison against historical incident records.
* The system should return a list of matched incidents, displaying the percentage similarity and relevant details (e.g., incident number, responsible team, and status).
* Ideally, matches with a similarity score of 90% or higher should be highlighted for quick reference.

1. **Historical Incident Database**

* The database should store closed and ongoing incidents, including their descriptions, resolution status, and assigned teams (e.g., Communications, Accounts, Financial, IT Support).
* Dummy data must be generated initially to simulate a realistic historical dataset for NLP processing.

1. **Dashboard and Search Features**

* Admins should have a dashboard that provides:
  + Overview of New, Assigned, and Ongoing incidents
  + Search Feature: A search bar where admins can input an incident ID to find similar historical and ongoing incidents.
* Incident details should be viewable in an expanded format, showing the long description, team responsible for previous resolutions, and timestamps.

1. **Notification System**

* Real-time notifications should be displayed when a new incident is submitted
* The system should alert admins of high similarity matches found during the NLP processing.

## Non-Functional Requirements

1. **Scalability**

* The backend system should be capable of handling increased data volume as the number of incidents grows. The NLP processing should be optimised to maintain fast response times, even with a larger historical dataset.

1. **Performance**

* Response times for incident submission, database queries, and NLP similarity searches should ideally be under 2 seconds
* The system should handle concurrent user sessions without performance degradation

1. **Security**

* Secure user authentication with encrypted passwords
* Role-based access control to prevent unauthorised access to admin-only features
* Data should be transmitted over SSL/TLS to protect user and incident information

1. **Usability**

* The user interface should be intuitive, with clear prompts and feedback mechanisms for users and admins.
* The system should be designed to minimise the learning curve for new users.

1. **Reliability**

* The system must be robust and handle edge cases, such as incomplete incident submissions or failed NLP searches, with appropriate error handling and user feedback.
* The system will incorporate regular data backups and automated failover mechanisms to maintain service continuity in case of unexpected failures. This ensures minimal disruption and data recovery options for high availability.

1. **Maintainability**

* The codebase should be modular to allow for future enhancements and easy maintenance.
* Detailed documentation should be provided for core modules, including the NLP engine and data processing logic.
* The project will leverage version control tools such as Git to track code changes and facilitate collaboration. Automated testing frameworks will be used to ensure new updates do not affect existing functionalities. Additionally, comprehensive documentation will be maintained for all modules, making future system enhancements and troubleshooting more efficient.

### Constraints and Assumptions

**Design Constraints**

1. The system must accommodate a role-based access structure to distinguish between standard users and admins.
2. The design must ensure scalability to handle an increasing volume of incidents and historical data without performance degradation.
3. The user interface design should prioritise simplicity and accessibility, catering to users with minimal technical expertise.
4. The system should be modular, allowing for the addition of new features without significant rework to the existing design.
5. Data flow between modules must ensure secure handling of sensitive information, such as user authentication credentials and incident descriptions.

**Assumptions**

1. Dummy historical incident data used for keyword matching and similarity scoring will be well-structured and sufficient in quantity and quality.
2. Users (both standard users and admins) will have basic technical proficiency to navigate the system and perform tasks such as submitting incidents and reviewing results.
3. System users will have reliable internet connectivity to access the web-based application seamlessly.
4. Admin users will have domain knowledge of the incident categories and resolution workflows, ensuring accurate interpretation of similarity results.
5. The system will be deployed in a controlled environment where updates and maintenance activities can be scheduled with minimal disruption to users.

## System Design

This section provides a detailed design of the Intelligent Incident Resolution System, focusing on its architecture, modules, data flow, NLP integration, and database schema. These design considerations are essential for building an efficient, secure, and scalable system that meets the project’s objectives.

With a clear understanding of the requirements and objectives established earlier in this chapter, the system design outlines the interactions between components and how they collectively deliver the desired functionality.

### A diagram of a software company Description automatically generated with medium confidenceHigh Level Architecture

Figure 3.1 Presentation, Application, and Data Layer Diagram

The **Presentation Layer** serves as the user interface of the system, providing an intuitive and responsive design to ensure seamless interaction between users and the system. This layer is designed to allow users to submit incidents effortlessly through a simple interface, while admins can efficiently access dashboards for incident management and analysis. It includes features such as forms for submitting incidents and an admin dashboard for tracking and analysing data.

The **Application Layer** is responsible for handling the core business logic of the system. It processes user submissions, generates unique incident identifiers, and performs similarity searches to identify related historical incidents.

This layer ensures that system features are accessible based on user roles, providing role-based access control to differentiate between user and admin functionalities. It acts as the central

component, coordinating interactions between the Presentation and Data layers to retrieve, store, and process incident data efficiently and securely.

The **Data Layer** is responsible for providing data persistence and ensuring reliable storage and retrieval of information. It stores incident records, user information, and historical data required for analysis. This layer supports real-time processing by making both new and historical incident data readily available to other components of the system. Its design ensures that data integrity and accessibility are maintained throughout the system's operation.

### A diagram of data flow Description automatically generatedData Flow

Figure 3.2 Data Flow Diagram

The data flow within the system starts with incident submission by users, which is processed and stored in the database. Admins retrieve these incidents by assigning the incident to themselves in the admin dashboard, analyse them, and assign them based on priority or similarity. The NLP module processes the incidents to identify matches using TF-IDF and cosine similarity, and the results are displayed to the admin for review. This structured data flow ensures smooth handling of incidents from submission to resolution.

### Module Description

The Incident Submission Module allows users to submit new incidents through a user-friendly web interface. This module ensures that each incident is assigned a unique incident number, using a method that generates a formatted incident number (e.g., INC1234567). The module captures both a short summary and a detailed description of the incident, applying validation methods to check for completeness and consistency before storing the data in the system database. Input handling methods ensure that user data is clear and formatted properly to maintain data integrity. For example, a method that will be used in the generation of an incident will be generateIncidentNumber**(),** this will aid in generating unique incident identifiers for each new incident.

The **NLP Processing Module** processes incident descriptions to identify and suggest similar historical incidents. This module uses natural language processing methods such as tokenization and stop-word removal to prepare the text data. The module applies the TF-IDF (Term Frequency-Inverse Document Frequency) method to create vector representations of incident descriptions, which helps in evaluating the importance of words within the context of stored incidents. To identify similarities, the module implements methods for calculating cosine similarity between vectors, aiding the retrieval of relevant past incidents that match new submissions. These methods support admins in making informed decisions quickly by displaying a list of potential matches and their similarity scores.

The **Admin Management Module** provides tools and methods for administrators to efficiently manage incident workflows. This includes methods for retrieving and displaying lists of new, assigned, and ongoing incidents, as well as updating the status of incidents and assigning them to specific admins or teams. The module incorporates a search method that allows admins to input an incident ID, triggering the NLP analysis to find related incidents. This functionality helps streamline the workflow by providing access to previous incidents and resolutions, which can guide the admin in resolving current incidents. The module also includes methods for sorting and filtering incidents to enhance usability and navigation. A method that can be used would be **assignIncidentToAdmin(),** this would be used when an admin is assigning a new incident to themselves.

The **Access Control Module** ensures that only authorised users can access specific features of the system through role-based access control. This module uses authentication and authorisation methods to differentiate between user roles, allowing only admins to perform tasks such as assigning incidents and conducting similarity searches. Methods within this module include validating user credentials, managing session tokens, and enforcing permissions for various system operations. By implementing these methods, the module maintains security and prevents unauthorised access, ensuring that sensitive incident data remains protected and only accessible to those with the appropriate permissions.

Each module plays a crucial role in the overall functionality of the system. To enhance these modules, we integrate advanced NLP techniques, which are detailed in the following section.

### Integration with NLP Techniques

The system leverages NLP techniques, specifically TF-IDF (Term Frequency-Inverse Document Frequency) and Cosine Similarity, to analyse and compare incident descriptions effectively. The implementation of the NLP module in Java uses TF-IDF vectors and Apache Commons Math for calculating Cosine Similarity scores. This integration helps highlight the most informative terms within incidents and measure the relevance between new and historical cases.

TF-IDF calculates term importance by assessing word frequency within an incident and adjusting for how often that word appears across the entire dataset. This vectorization ensures that significant keywords are emphasised over common ones, allowing precise incident comparison.

Cosine similarity then measures how closely the TF-IDF vectors of new incidents align with those of past incidents. A similarity score close to 1 indicates a strong match, which is converted into a percentage for ease of interpretation by admins. This enables admins to quickly identify relevant past incidents, aiding in more informed and consistent decision-making.

The NLP processing workflow includes:

1. **Text Preprocessing**: Tokenization of text and removal of stop words to prepare the content for analysis.
2. **Vectorisation**: Generation of TF-IDF vectors for each incident description using a custom implementation in Java.
3. **Similarity Scoring**: Application of cosine similarity through Apache Commons Math to find and rank related incidents.
4. **Semantic Expansion with GloVe**: GloVe 50-dimensional word embeddings are used to generate averaged word vectors for each incident. These embeddings enhance the system's ability to identify semantically similar incidents even when different vocabulary is used, providing a second layer of matching that captures deeper contextual meaning.

This integration enables admins to reference historical cases efficiently, accelerating the incident assignment and resolution process and ensuring consistent responses across similar incidents.

## Software Design

This section outlines the software design of the **Intelligent Incident Resolution System**, including key classes, methods, and their interactions. The design ensures that each functional requirement is implemented effectively while maintaining modularity, scalability, and ease of maintenance.

**Key Classes and Their Responsibilities**

The following classes are designed to fulfil the functional requirements of the system:

1. **User Class**:
   * **Purpose**: Represents a system user (either admin or standard user).
   * **Attributes**:
     + userId: Unique identifier for the user.
     + username: The user’s login name.
     + password: Hashed password for authentication.
     + role: Specifies whether the user is an Admin or StandardUser.
   * **Methods**:
     + authenticateUser(): Verifies the user’s credentials during login.
     + getRole(): Returns the role of the user for access control.
2. **Incident Class**:
   * **Purpose**: Represents an incident submitted by a user.
   * **Attributes**:
     + incidentId: Unique identifier for the incident.
     + shortDescription: A summary of the incident.
     + longDescription: Detailed explanation of the incident.
     + status: Status of the incident (New, Assigned, Ongoing, Closed).
     + assignedTo: Admin assigned to the incident.
   * **Methods**:
     + generateIncidentId(): Generates a unique ID for the incident.
     + updateStatus(newStatus): Updates the status of the incident.
3. **NLPProcessor Class**:
   * **Purpose**: Handles natural language processing tasks for similarity analysis.
   * **Attributes**:
     + incidentDescription: The text of the incident being processed.
     + historicalData: A dataset of historical incidents for comparison.
   * **Methods**:
     + preprocessText(text): Tokenizes text and removes stop words.
     + calculateTFIDF(): Calculates the TF-IDF vectors for incident descriptions.
     + computeCosineSimilarity(vector1, vector2): Computes the similarity score between two incidents.
4. **IncidentManager Class**:
   * **Purpose**: Manages incident-related operations for admins.
   * **Attributes**:
     + incidentList: A list of all incidents in the system.
   * **Methods**:
     + assignIncidentToAdmin(incidentId, adminId): Assigns an incident to an admin.
     + getIncidentDetails(incidentId): Retrieves detailed information about an incident.
     + runSimilarityAnalysis(incidentId): Invokes the NLPProcessor to find similar incidents.
5. **DatabaseManager Class**:
   * **Purpose**: Manages interactions with the database.
   * **Attributes**:
     + connection: Database connection object.
   * **Methods**:
     + storeIncident(incident): Saves a new incident in the database.
     + retrieveIncident(incidentId): Retrieves an incident from the database.
     + storeSimilarityResults(results): Saves similarity results.

A diagram of a company

Description automatically generated

Figure 3.3 Relationships between key components

|  |  |
| --- | --- |
| Functional Requirement | Supporting Method(s) |
| User Authentication | authenticateUser() (User Class) |
| Incident Submission | generateIncidentId(), storeIncident() |
| Incident Assignment | assignIncidentToAdmin(), updateStatus() |
| NLP Similarity Search | preprocessText(), calculateTFIDF(), computeCosineSimilarity() |
| Incident Retrieval and Analysis | getIncidentDetails(), runSimilarityAnalysis() |

## Database Design

The **Database Design** section outlines the structure of the database that supports the Intelligent Incident Resolution System. This design ensures the efficient storage, retrieval, and processing of data, aligning with both functional and non-functional requirements. The database schema includes core tables, relationships, and considerations to address system functionalities such as incident submission, management, and NLP-based similarity matching.

**Database Tables**

The database schema includes the following tables:

1. **Incidents Table**: Stores information about reported incidents.
2. **Users Table**: Manages user authentication and access control.
3. **Teams Table**: Stores team details for assignment purposes.

**Schema Overview**

The schema includes relationships between tables to ensure data integrity and seamless functionality. The primary relationships include:

* **Users → Incidents**: Tracks which admin or team member is handling an incident.
* **Incidents → Similarity Scores**: Links new incidents with similar historical incidents.
* **Teams → Users**: Tracks which users belong to specific teams.
* **Incident History → Incidents**: Logs updates and changes to incidents.
* **Resolution Details → Incidents**: Stores information about how an incident was resolved.

**Incidents Table**

The **Incidents** table stores the main data for each reported incident, including details such as descriptions, status, and timestamps. This table is critical for tracking the lifecycle of incidents from submission to resolution.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| incident\_number {PK} | VARCHAR(15) | Unique identifier for each incident |
| short\_desc | VARCHAR(65) | Short description of the incident |
| long\_desc | TEXT | |  | | --- | |  |  |  | | --- | | Detailed description of the incident | |
| status | ENUM(‘new’,‘assigned’, ‘ongoing’, ‘closed’) | Current status of the incident |
| severity | ENUM (‘low’, ‘medium’, ‘high’, ‘critical’) | Severity level of the incident |
| assigned\_to  {FK > Users.user\_id} | VARCHAR(50) | Admin assigned to the incident |
| team\_id  {FK > Teams.team\_id} | INT | Team responsible for resolution |
| created\_at | TIMESTAMP | Timestamp of when the incident was created |
| created\_by | VARCHAR (50) | Details of who submitted the incident |
| resolution | TEXT | Resolution details for resolved incident |

**Key Considerations**:

* **Primary Key**: incident\_id ensures unique identification of each record.
* **Status Field**: The ENUM type allows controlled status values, making it easier to filter incidents based on their state.
* **Data Integrity**: The assigned\_to and team fields link to user roles and teams responsible for handling incidents.
* The severity field supports prioritisation based on SLA agreements.
* The team\_id links to the **Teams Table**, ensuring incidents can be assigned to a team and specific members.

**Users Table**

The **Users** table manages user authentication and role-based access control. This table supports the login process and enforces role differentiation between standard users and admins.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| user\_id {PK} | INT | Unique identifier for each user |
| username | VARCHAR(50) | Username for login |
| password | VARCHAR(255) | Hashed password for secure authentication |
| email | VARCHAR(30) | User email address |
| role | ENUM(‘User’, ‘Admin’) | Role designation for access control |
| team\_id  {FK > Teams.team\_id} | INT | Team to which the user (admin) belongs |

**Key Considerations**:

* **Password Security**: The password field uses hashing algorithms like **BCrypt** to store securely hashed passwords.
* **Role Management**: The role field determines the access level and permissions within the system, ensuring that admin-specific features are protected.

**Teams Table**

The **Teams Table** organises information about the various teams responsible for resolving incidents, enabling the system to assign incidents to appropriate groups for efficient management.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| team\_id {PK} | INT | Unique identifier for each team |
| team\_name | VARCHAR(50) | Name of the Team (eg., Communications, Accounts) |
| team\_admin {FK > Users.user\_id} | INT | User ID of the Team admin |

**Key Considerations:**

* Teams allow flexible assignment of incidents to groups rather than individuals.

### Relationships and Data Flow

The relationships between the tables are designed to ensure seamless data flow across the system. The Incidents table references the Users table via the assigned\_to field, allowing tracking of which admin is responsible for resolving each incident. Additionally, the Incidents table links to the Teams table using the team\_id field to enable team-based assignments. The SimilarityScores table connects to the Incidents table to store results from NLP-based analysis, providing a way to cross-reference new and historical incidents. The IncidentHistory table logs changes to incidents over time, and the ResolutionDetails table records how incidents were resolved for future reference.

**Data Flow Summary**:

1. **Incident Submission**:  
   A user submits a new incident through the user interface, which is recorded in the Incidents table. The system generates a unique incident\_id to identify the record, and the details such as short\_desc, long\_desc, severity, and status (new) are stored.Admins managing incidents will be linked via the assigned\_to field to records in the user’s table.
2. **Team and User Assignments**:  
   The team\_id field in the Incidents table links each incident to a specific team in the Teams table. Within the team, the assigned\_to field references the Users table to track the individual admin responsible for managing the incident.
3. **Incident Updates and Lifecycle Tracking**:  
   As incidents progress, updates such as status changes, team reassignments, or additional notes are logged in the IncidentHistory table. Each update references the incident\_id in the Incidents table and the updated\_by field in the Users table, providing a clear audit trail of the incident lifecycle.
4. **Similarity Analysis**:  
   When an admin searches for similar incidents, the NLP module processes the new incident description and identifies matches with historical incidents. The results, including the incident\_id, similar\_id, and similarity\_score, are stored in the SimilarityScores table. This allows admins to review relevant historical incidents and their resolutions.
5. **Resolution Logging**:  
   Once an incident is resolved, the ResolutionDetails table stores the steps taken to resolve the issue, linking to the Incidents table via the incident\_id field. The resolved\_by field references the admin responsible for the resolution, ensuring a complete record of the resolution process.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3.4 Database Schema Relationships Diagram

## User Interface Design

**Design Principles**

The user interface (UI) is built around the principles of simplicity, intuitiveness, and efficiency to enhance user experience. The design must cater to two types of users: regular users submitting incidents and admin users who manage, assign, and analyse incidents. The following design principles guide the development of the UI and are implemented as follows:

1. **Clarity and Simplicity**:
   * **Implementation**: The interface features a clean layout with clearly labeled forms, buttons, and navigation links. Forms for incident submission include concise placeholders (e.g., "Enter a short description") and tooltips for additional guidance.
   * **Example**: A dedicated dashboard for admins separates sections for new, ongoing, and resolved incidents, minimising clutter and making navigation intuitive.
2. **Consistency**:
   * **Implementation**: Consistent visual styling is applied across all pages, using the same color scheme, typography, and button designs. Reusable UI components, such as a uniform button style and navigation bar, ensure familiarity throughout the application.
   * **Example**: Both users and admins use the same basic form layout for input fields, maintaining a predictable design flow across different user roles.
3. **Responsiveness**:
   * **Implementation**: The UI is developed using responsive design techniques, including flexible grid layouts and media queries, to ensure compatibility with different screen sizes and devices.
   * **Example**: A mobile-friendly incident submission form adjusts automatically to fit smaller screens, making the system accessible on smartphones and tablets.
4. **Feedback and Status Indicators**:
   * **Implementation**: The system includes real-time notifications and visual cues for user interactions. For example, when a user submits a form, a confirmation message appears, and required fields are highlighted in red if left blank.
   * **Example**: Admin dashboards display colour-coded status indicators for incidents (e.g., "New" incidents in blue, "Ongoing" in yellow, and "Resolved" in green) to provide immediate visual feedback.
5. **Accessibility**:
   * **Implementation**: The UI follows accessibility standards such as WCAG (Web Content Accessibility Guidelines). This includes features like keyboard navigation, screen reader compatibility, and high-contrast modes for users with visual impairments.
   * **Example**: The incident submission form includes proper ARIA (Accessible Rich Internet Applications) labels for all input fields, enabling compatibility with assistive technologies like screen readers.

**Interface Mockups**

The UI mockups are designed to reflect the needs of users and admins, focusing on ease of navigation and functionality.

Incident Submission Form: The incident submission form is the entry point for users to report new incidents. It should include:

* **Input Fields**: Short description and detailed description fields with clear labels and placeholder text to guide user input.
* **Submit Button**: A prominent button that allows users to submit the form once all required fields are filled.
* **Confirmation Message**: Upon submission, a notification informs the user that the incident has been recorded and provides the unique incident ID.

Admin Dashboard: The admin dashboard is central to the admin user experience, providing an organised overview and essential tools for incident management. The dashboard should include:

* **New Incidents Section**: A real-time list of new incidents submitted by users, including incident IDs and brief descriptions.
* **Open and Assigned Incidents Section**: Displays ongoing incidents and those assigned to admins, allowing for status updates and further actions.
* **Search Feature**: A search bar integrated into the dashboard enables admins to input an incident ID and run keyword matching and similarity analysis for quick referencing of related past incidents.
* **Incident Details View**: Clicking on an incident allows the admin to view comprehensive details, including the long description, current status, and any similarity matches found.

A screenshot of a computer

Description automatically generatedThe **Submit Incident** page serves as the primary interface for users to report issues within the system. It includes fields for providing a concise short description, a detailed explanation of the issue, and a severity level to ensure incidents are categorised appropriately for resolution.

Figure 3.5 Submit Incident Page Mockup

A screenshot of a computer

Description automatically generated

Figure 3.6 Admin Dashboard Mockup

The **Admin Dashboard - New Incidents Tab** provides administrators with an organised view of all new incidents submitted by users. Each incident includes a unique ID, a concise short description, its severity level, and status. Admins are offered the functionality to assign incidents to specific teams or individuals directly from this interface.

A screenshot of a computer

Description automatically generated

Figure 3.7 Incident Details Breakdown Mockup

The **Incident Details View** provides an in-depth breakdown of a specific closed incident, displaying crucial information such as the incident ID, detailed description, resolution steps undertaken, severity level, the responsible team, and resolution timeline. This feature ensures that admins have all necessary details for record-keeping and analysis.

A white background with black and red text

Description automatically generated

Figure 3.8 Filtered Incidents Mockup

The **Filtered Closed Incidents View** displays incidents specific to the selected team, such as 'IT Support.' This view includes key details like incident ID, short description, severity level, and status, alongside an actionable button for viewing further details of each incident.

A screenshot of a computer

Description automatically generated

Figure 3.9 Similarity Results Mockup

The **Similarity Search Results** feature provides a comprehensive list of incidents sorted by their similarity percentage. This functionality aids admins in finding previously resolved incidents that closely match new ones, streamlining incident resolution processes.

## Security Design Considerations

Ensuring the security of the **Intelligent Incident Resolution System** is paramount for protecting user data, maintaining system integrity, and preventing unauthorised access. The system incorporates multiple security layers to safeguard both the application and user interactions.

**Role Based Access Control**

The system employs **Spring Security** to implement robust role-based access control (RBAC). This framework differentiates between user and admin roles, ensuring that only authorised users can perform specific actions such as viewing, assigning, and analysing incidents. User roles are defined and assigned during account creation, and permissions are enforced through method-level and URL-based security configurations. This ensures that:

* **Users** have access restricted to submitting new incidents and viewing their statuses.
* **Admins** have full access to incident management functionalities, including searching, assigning, and running NLP-based analysis.

This approach ensures a clear boundary between regular users and administrators, preventing unauthorised data access or modifications.

**Data Protection**

To protect user data, the system incorporates multiple data security measures:

* **Password Hashing**: User passwords are securely hashed using algorithms such as **BCrypt**, which adds a salt to each password to mitigate the risk of brute-force attacks. This ensures that even if the database is compromised, stored passwords remain secure.
* **Encryption**: Sensitive data, such as user credentials and incident details, are encrypted both at rest and in transit. This prevents unauthorised users from accessing sensitive information.

## Summary

This chapter provided a comprehensive analysis and design of the Intelligent Incident Resolution System. It outlined the key decisions that guided the development of the system, emphasising scalability, security, and efficiency to address the challenges of incident management.

The system's architecture was detailed, highlighting the roles of the Presentation, Application, and Data layers, and explaining how these components work together to ensure smooth user interactions, efficient business logic processing, and reliable data storage. Specific emphasis was placed on creating a design that supports user-friendly features such as incident submission, team-based assignment, and advanced similarity searches for historical data.

Additionally, this chapter explored the integration of advanced techniques for analysing and comparing incident descriptions, allowing admins to efficiently identify and reference relevant past cases. Security considerations were also a key focus, ensuring the design meets the requirements for data protection, role-based access control, and system integrity.

This design establishes a solid foundation for the system, ensuring it is well-structured to meet the functional and non-functional requirements outlined in earlier sections. With these design principles and frameworks in place, the implementation phase will bring the system to life, transforming the concepts into a fully functional application.

# Implementation

## Introduction

The implementation chapter details how the functional and non-functional features of the system are implemented, which includes both Semantic Matching and Natural Language Processing (NLP) techniques. This chapter focuses on the practical aspects of building the system, including the tools and technologies used, the environment configuration, database setup, key features, and the challenges faced along the way.

The project followed a structured development process based on established software engineering practices. The system was designed to streamline support operations by automating the identification and resolution of incidents using NLP and machine learning. This chapter highlights how the theoretical concepts outlined during the analysis and design phase were translated into a working solution.

Development involved a combination of backend and frontend work, integrating NLP algorithms for semantic similarity matching and implementing a secure database to store and manage incident records. The chapter displays the coding techniques, system architecture, and technologies used to meet the project’s objectives.

Each module was developed and tested individually before being integrated into the full system, ensuring reliability and scalability. This approach allowed for efficient handling of user-submitted incidents while leveraging historical data to identify patterns and similarities.

The completed system is designed to enhance the incident management process by reducing resolution times and improving user satisfaction. Built with Java, JavaScript, MySQL, and NLP libraries, it offers a robust and scalable solution to the challenges of incident resolution in a fast-paced support environment.

## Tools and Technologies Used

This section outlines the various tools and technologies used in the development of the Intelligent Incident Resolution System. Each tool and technology is described in detail, highlighting its purpose within the project and the reason behind its selection.

**Software**

The project made use of several key software tools and libraries to achieve its objectives. Below is a detailed breakdown of the software stack.

**Java (Version 17)**

Java was chosen as the primary programming language for backend development due to its platform independence, robustness, and extensive ecosystem of libraries and frameworks. The object-oriented nature of Java allowed for a modular and scalable architecture, making it suitable for the development of complex systems such as the Intelligent Incident Resolution System.

Java's strong typing and runtime checking features contributed to the stability and reliability of the application. Additionally, the availability of numerous libraries, including those for NLP processing and database connectivity, made it a natural choice for this project.

Example Usage:

* Implementing REST APIs to handle user requests
* Creating the NLP module for text preprocessing and keyword matching
* Managing the application logic for incident submission and resolution

**Spring Boot (Version 3.0)**

Spring Boot was selected as the web framework to build the backend application. It simplifies the development of Java-based web applications by providing pre-configured setups for various functionalities such as RESTful APIs, security, and database connectivity.

Key Features Utilised:

* **Dependency Injection:** Ensured that components were loosely coupled, making the application easier to maintain and extend.
* **REST Controllers:** Handled HTTP requests and responses efficiently.
* **Configuration Management:** Simplified environment-specific configurations, allowing the system to adapt to different deployment setups.

Example Usage:

* Setting up RESTful endpoints for incident submission, retrieval, and management
* Integrating the NLP module with the backend logic
* Managing database connections and query execution

**GloVe Word Embeddings (50d)**

GloVe (Global Vectors for Word Representation) was used to implement the semantic similarity matching functionality within the system. GloVe provides pre-trained word embeddings that capture the contextual meaning of words based on global word co-occurrence statistics. These embeddings enabled the system to represent incident descriptions as numerical vectors and compare them using cosine similarity.

**Key Features Utilised:**

* **Word Vector Representation:** Converted each word in an incident description into a high-dimensional vector using pre-trained GloVe embeddings.
* **Semantic Matching:** Enabled the system to measure conceptual similarity between incidents, even when exact keywords did not match.
* **Cosine Similarity:** Used to compute the degree of similarity between new and historical incident descriptions based on their vector representations.

**Example Usage:**

* Loading the glove.6B.50d.bin file into memory and mapping each word to its corresponding vector. Averaging word vectors to generate a single vector for each incident description.
* Comparing the vector of a new incident with vectors of past incidents to identify semantically similar cases.
* Displaying matched incidents and similarity scores directly to the admin via the frontend interface, helping them resolve issues more efficiently.

**JavaScript (ES6+)**

JavaScript was used to add interactivity to the frontend components of the system. It was employed to handle client-side logic, such as form validation and dynamic content updates.

Example Usage:

* Validating user input in the incident submission form
* Displaying real-time status updates on the admin dashboard
* Enhancing the user experience with interactive elements

**HTML/CSS**

HTML and CSS were used to structure and style the frontend user interface. The combination of HTML for content and CSS for styling ensured that the system provided a user-friendly and visually appealing interface.

Example Usage:

* Designing the incident submission form
* Creating the admin dashboard layout
* Styling buttons, inputs, and messages for consistency across pages

A diagram of a company

AI-generated content may be incorrect.

Figure 4.1 System Architecture: Overview of user interactions, frontend, backend, database, and NLP module

**Reasoning for Tool Selection**

Each tool was carefully chosen to meet specific requirements of the project:

* **Java and Spring Boot:** Provided a stable and scalable backend framework.
* **MySQL:** Ensured efficient storage and retrieval of structured data.
* **JavaScript:** Added dynamic functionality to the frontend, enhancing user experience.
* **HTML/CSS:** Provided a clean and accessible user interface.

These tools collectively contributed to the successful implementation of the Intelligent Incident Resolution System, ensuring that it met both functional and non-functional requirements.

## Environment Configuration

Setting up a reliable and efficient development environment is critical for ensuring the successful implementation of the Intelligent Incident Resolution System. This section outlines the backend and frontend configuration processes, detailing the tools, software, and dependencies used. Proper configuration ensures that the system functions as intended and integrates seamlessly across different components.

**Backend Environment Configuration**

The backend of the system is built using Java and Spring Boot. The following steps outline the configuration process for setting up the backend environment.

**Development Tools and Software**

* **Operating System:** Windows 10
* **Integrated Development Environment (IDE):** IntelliJ IDEA 2024.3.1.1
* **Java Development Kit (JDK):** JDK 17
* **Build Tool:** Maven 3.8+
* **Database Server:** MySQL 8.0

**Backend Setup Process**

**Install JDK 17:**

* Download and install the Java Development Kit (JDK) from the official Oracle website or OpenJDK.
* Set the JAVA\_HOME environment variable to ensure Java is correctly configured.
* Verify the installation by running the java -version command in the terminal.

**Install IntelliJ IDEA:**

* Download and install IntelliJ IDEA, which provides a comprehensive development environment for Java projects.
* Create a new Spring Boot project using the built-in wizard to simplify project setup.

**Set Up the Project Structure:**

* Use Maven to manage project dependencies. The pom.xml file will include essential dependencies such as Spring Boot Starter, MySQL Connector, and Apache Lucene.
* Create appropriate package structures for controllers, services, repositories, and models to maintain a clean and organised codebase.

**Configure application.properties File:**

The application.properties file in the Spring Boot project contains configuration details for database connectivity and other application-specific settings.

Configuration:

spring.datasource.url=jdbc:mysql://localhost:3306/ fyp\_incident\_system

spring.datasource.username=root

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

**Install and Configure MySQL:**

* Download and install MySQL Server.
* Create a new database named fyp\_incident\_system to store incident records and other related data.
* Use MySQL Workbench to manage the database and execute SQL queries.

**Run the Backend Application:**

* Use IntelliJ IDEA to build and run the Spring Boot application.
* Ensure the server starts without errors and the REST endpoints are accessible via Postman or a web browser.

**Frontend Environment Configuration**

The frontend of the system is developed using standard web technologies, including HTML, CSS, and JavaScript. Below are the steps to set up the frontend environment.

**Development Tools and Software**

* **Code Editor:** IntelliJ IDEA 2024.3.1.1
* **Browser:** Google Chrome
* **Node.js:** Latest version
* **Live Server Extension:** Used for real-time preview of frontend changes

**Frontend Setup Process**

**Install IntelliJ IDEA:**

* Download and install IntelliJ IDEA from the official website.
* Set up a new project for the frontend development.

**Install Node.js:**

* Download and install Node.js from the official website.
* Verify the installation by running the node -v and npm -v commands in the terminal.

**Initialise the Frontend Project:**

* Create a new project directory and navigate to it in the terminal.
* Run the following commands to initialise a new Node.js project and install any necessary dependencies:
* npm init -y npm install

**Set Up the Frontend Files:**

* Create the necessary HTML, CSS, and JavaScript files to build the frontend interface.
* Ensure proper file organisation by creating separate folders for assets (e.g., images, CSS files, JavaScript files).

**Run the Frontend Application:**

* Use the Live Server extension in Visual Studio Code to preview the frontend application in real time.
* Ensure that all elements render correctly, and that the frontend communicates with the backend API endpoints.

**Libraries and Dependencies**

A variety of libraries were used to facilitate the development process. Below is a table summarising the key dependencies.

|  |  |  |
| --- | --- | --- |
| **Library/Dependency** | **Version** | **Purpose** |
| Spring Boot Starter Web | 3.0 | Building RESTful web services |
| MySQL Connector | 8.0 | Connecting the backend to the MySQL database |
| GloVe Word Embeddings (via .bin file) | 50d | Converts incident descriptions into vector representations for semantic similarity comparison using cosine distance |
| Lombok | Latest | Reducing boilerplate code in Java classes |
| Node.js | Latest | Managing frontend dependencies |

Table 4.1 Libraries and Dependencies

A diagram of a computer

Description automatically generated

Figure 4.2 Frontend Architecture Diagram

A diagram of a flowchart

AI-generated content may be incorrect.

Figure 4.3 Backend Architecture Diagram

The environment configuration process ensured that both the backend and frontend could be developed, tested, and deployed efficiently. Proper configuration minimised compatibility issues and ensured smooth integration of different components.

## Database Implementation

The database implementation is a core aspect of the Intelligent Incident Resolution System, designed to ensure efficient storage, retrieval, and management of incident data, user information, and similarity analysis results. This section outlines the practical steps taken to implement the database structure, including the creation of tables, setting up relationships, and ensuring data integrity and security.

**Database Schema Overview**

The database schema was designed to align with the functional requirements outlined in the Analysis and Design chapter. The primary tables include **Users**, **Incidents**, **Teams**, **Similarity Scores**, **Incident History**, and **Resolution Details**. The database is implemented using MySQL, chosen for its scalability and compatibility with Java-based applications.

The relationships between these tables ensure seamless data flow, enabling the system to efficiently manage incident submissions, assignments, and resolutions.

**Implementation Steps**

**1. Creating the Database**

The first step in the database implementation was setting up the MySQL server and creating a new database named fyp\_incident\_system.

**SQL Command:**

A screenshot of a computer code

Description automatically generated

**2. Creating Tables**

The following tables were created to store the necessary data:

**Users Table**

The Users table manages user authentication and access control, distinguishing between admins and standard users.

**SQL Command:**

A screenshot of a computer code

Description automatically generated

**Incidents Table**

The Incidents table records all incidents submitted by users, along with their status and severity.

**SQL Command:**

A computer screen shot of a program

Description automatically generated

**Teams Table**

The Teams table organises different resolution teams responsible for handling incidents.

**SQL Command:**

A white background with black text

Description automatically generated

**Database Relationships**

The database relationships ensure that the system can effectively link users to incidents, teams to incidents, and new incidents to historical incidents for similarity analysis. These relationships are enforced using foreign keys to maintain referential integrity.

**Data Population**

To ensure the system operates effectively, initial data was populated into the database for testing various features, such as user authentication, incident submission, and similarity analysis. This data also facilitated testing of database relationships and the functionality of the NLP-based similarity module.

**Sample Data for Users Table**

The **Users** table contains records for both normal users and admins. The following data was populated to simulate a real-world scenario:

|  |  |  |  |
| --- | --- | --- | --- |
| user\_id | username | password | role |
| 1 | admin | Admin123 | ROLE\_ADMIN |
| 2 | user | User123 | ROLE\_USER |

**Sample Data for Incidents Table**

The **Incidents** table was populated with sample incidents to test incident submission, status updates, and assignments:

|  |  |  |  |
| --- | --- | --- | --- |
| incident\_id | user\_id | short\_description | long\_description |
| INCFG5367 | 2 | Login issue | User cannot log into their account |
|  |  |  |  |
| INC36EGY | 2 | Payment Error | Error during payment process |

|  |  |  |  |
| --- | --- | --- | --- |
| severity | Status | assigned\_team | created\_at |
| High | Open | IT | 2025-01-15 09:00:00 |
| Medium | Resolved | Support | 2025-01-16 11:00:00 |

**Sample Data for Similarity\_Scores Table**

The **Similarity\_Scores** table contains data linking new incidents to historical ones, including similarity scores generated by the NLP module:

|  |  |  |  |
| --- | --- | --- | --- |
| score\_id | incident\_id | similarity\_score | match\_status |
| 1 | INCFG5367 | 0.85 | Matched |
| 2 | INC36EGY | 0.90 | Matched |

|  |  |
| --- | --- |
| resolution\_team | calculated\_at |
| IT Support | 2025-01-15 09:10:00 |
| Finance | 2025-01-16 11:10:00 |

**Testing Scenarios**

The populated data was used to validate key functionalities:

* **User Authentication:** Verifying that users can log in with hashed passwords and role-based access.
* **Incident Submission:** Testing the submission process, including the association of incidents with the submitting user.
* **Similarity Analysis:** Using sample incident data to ensure that the NLP module correctly identifies similar incidents and assigns the appropriate resolution team.
* **Data Integrity:** Ensuring that all foreign key relationships are intact and enforced during data manipulation.

**Database Connection in Spring Boot**

The connection between the backend application and the MySQL database was configured in the application.properties file.

**Configuration:**

A screen shot of a computer

Description automatically generated

Figure 4.4 Database Connection Configuration

This configuration ensures that the backend application can communicate with the database for CRUD operations.

**Security Measures**

To secure the database, several measures were implemented:

* **Password Hashing:** User passwords are securely hashed using algorithms like bcrypt.
* **Role-Based Access Control:** Users are assigned roles that determine their access permissions.
* **Regular Backups:** The database is backed up regularly to prevent data loss.

**Example SQL Queries and Output**

To validate the database implementation, the following example SQL queries were executed, along with their simulated outputs:

**Query 1: Fetch All Users**



**Output:**

|  |  |  |
| --- | --- | --- |
| **user\_id** | **username** | **Role** |
| 1 | Jsmith | ROLE\_ADMIN |
| 2 | Mdoe | ROLE\_USER |
| 3 | Ajones | ROLE\_USER |

**Query 2: Fetch Incidents Assigned to a Specific Team**

A white background with black text

Description automatically generated

**Output:**

|  |  |  |
| --- | --- | --- |
| **incident\_id** | **short\_description** | **assigned\_team** |
| INCDG465T | "Login issues reported" | IT Support |
| INCEY664T | "Password reset needed" | IT Support |

**Query 3: Fetch Similarity Scores Above 80%**

A black text on a white background

Description automatically generated

**Output:**

|  |  |
| --- | --- |
| **incident\_id** | **similarity\_score** |
| INCDG465T | 0.85 |
| INCEY664T | 0.91 |

The implemented database structure ensures that all data is efficiently managed and securely stored, supporting the core functionalities of the Intelligent Incident Resolution System.

## Feature Implementation

The feature implementation section provides a comprehensive breakdown of the core functionalities developed for the Intelligent Incident Resolution System. Each feature is described in detail, including the implementation process, key code snippets, and the rationale behind the chosen approach. This section covers the core features such as user authentication, incident submission and management, NLP-based similarity analysis, and role-based access control.

**User Authentication and Role-Based Access Control**

The system implements a secure user authentication mechanism that ensures only authorised users can access specific features based on their roles (Admin or User). The authentication process involves verifying user credentials and assigning appropriate roles to control access to sensitive functionalities.

**Implementation Details:**

**User Login:**

* The login feature is implemented using a RESTful API endpoint that accepts username and password as input.
* The backend verifies the credentials by querying the Users table.
* Passwords are securely stored as hashed values using the **bcrypt** algorithm.

**Code Snippet:**

A screenshot of a computer code

Description automatically generated

Figure 4.5 User Creation and Role Retrieval Code Snippet

Role-Based Access Control:

* Admins have access to features such as viewing all incidents, managing users, and assigning incidents to resolution teams.
* Users can submit new incidents and view the status of their own incidents.

**Code Snippet for Access Control:**

A screenshot of a computer code

Description automatically generated

Figure 4.6 Code Snippet for Role-based Access Control Implementation

**Incident Submission and Management**

The system allows users to submit incidents through a user-friendly interface. The incidents are stored in the Incidents table and assigned a unique identifier. The system also tracks the status of each incident (e.g., new, assigned, ongoing, closed) and its severity level.

**Implementation Details:**

* **Incident Submission:**
  + The frontend collects incident details from the user, including a short description, long description, and severity level.
  + The backend processes the submission and stores the incident in the database.

**Code Snippet:**

A screen shot of a computer code

Description automatically generated

Figure 4.7 Code Snippet for Incident Submission Endpoint

* **Incident Management:**
  + Admins can view all incidents and update their statuses.
  + The system automatically updates timestamps whenever an incident is modified.

**Code Snippet for Updating Incident Status:**

A screen shot of a computer code

Description automatically generated

Figure 4.8 Code Snippet for Updating Incident Statuses

**NLP-Based Similarity Analysis**

The system incorporates Natural Language Processing (NLP) techniques to perform **semantic similarity matching** between newly submitted incidents and historical incidents stored in the database. This functionality allows administrators to quickly identify past incidents with similar characteristics, enabling more efficient resolution through pattern recognition and intelligent case assignment.

**Text Preprocessing**

Before any similarity calculation, the system preprocesses incident descriptions using a custom pipeline designed to normalize and enrich the input data. The preprocessing steps include:

* **Tokenization:** Splitting incident descriptions into individual words.
* **Lowercasing & Cleaning:** Removing punctuation and converting all characters to lowercase.
* **Stopword Removal:** Filtering out common words that do not contribute meaningful information (e.g., "the", "is", "in").
* **Synonym Expansion:** Augmenting input text with domain-specific synonyms such as "login" → "signon", or "crash" → "failure", enhancing the system's ability to match semantically similar incidents even when terminology differs.

This preprocessing ensures consistent input to the similarity modules.

A computer code with many text

AI-generated content may be incorrect.

Figure 4.9 Preprocessing Text Code Snippet

**TF-IDF Similarity Scoring**

The first layer of similarity is calculated using a **TF-IDF (Term Frequency-Inverse Document Frequency)** approach:

* A **global document frequency map** is created from all incidents.
* A **TF-IDF vector** is generated for both the target and comparison incident using term frequency and inverse document frequency.
* **Cosine similarity** is then calculated between the TF-IDF vectors to determine structural textual similarity.

**TF-IDF Mapping:**

A computer screen shot of a program

AI-generated content may be incorrect.

Figure 4.10 TF-IDF Code Snippet

**Cosine Similarity:**

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.11 Cosine Similarity Code Snippet

This method ensures that relevant keywords are weighted appropriately based on how unique they are across the dataset.

**Semantic Matching using GloVe Word Embeddings**

To enhance contextual understanding, the system also performs semantic similarity analysis using **GloVe (Global Vectors for Word Representation)**:

* Pre-trained GloVe vectors (glove.6B.50d.bin) are loaded at runtime.
* The incident descriptions are tokenized and mapped to their corresponding GloVe vectors.
* The vectors are **averaged** to form a single semantic vector for the entire incident.
* A **cosine similarity score** is then calculated between the two semantic vectors.

This method enables the system to recognise similarity even when wording is different, but meaning is consistent.

A screenshot of a computer code

AI-generated content may be incorrect.

Figure 4.12 The Comparison of words in Incidents

**Blended Similarity Score**

To benefit from both literal keyword overlap and contextual meaning, a **blended similarity score** is calculated using a weighted combination of TF-IDF and Word2Vec scores:

finalSimilarity = (0.3 × cosineSim\_TFIDF) + (0.7 × cosineSim\_Word2Vec);

This score is then scaled and compared against a similarity threshold (e.g., 0.05) to determine whether a historical incident is considered a match.

**Search Workflow**

The overall workflow of the similarity analysis is as follows:

1. Admin initiates a search using an **incident number**.
2. All historical incidents are retrieved from the database.
3. The system preprocesses and analyses each incident.
4. A **blended similarity score** is computed for each comparison.
5. If the score exceeds the threshold, the match is added to the result set.
6. Matched results are returned to the admin as IncidentMatchResult objects, including:
   * Incident number, title, description
   * Similarity score
   * Assigned team and admin (if any)
   * Resolution details (if available)

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 4.13 Blended Similarity Score - TF-IDF and Word Semantics

**Optimisation Highlights**

Several optimisations were implemented to improve accuracy and performance:

* **In-memory processing:** No external indexing tools are required. All similarity computations are done on-the-fly using Java.
* **Synonym expansion:** Tailored mappings increase match quality across departments.
* **Caching:** GloVe vectors are loaded once and reused across searches.
* **Scalability:** Weighted scoring allows fine-tuning for different environments or datasets.

This hybrid NLP strategy allows the system to provide highly accurate, intelligent recommendations to admins by comparing not just keywords, but the deeper **semantic meaning** of each incident. The result is a smart, context-aware incident resolution system tailored to real-world support environments.

**Security Features**

To ensure data security, the system implements the following measures:

* **Password Hashing**: User passwords are hashed using bcrypt before being stored in the database.
* **SQL Injection Prevention**: All SQL queries are parameterized to prevent injection attacks.

Code Snippet for Password Hashing:

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

The feature implementation process focused on building a secure, scalable, and efficient system that meets the project’s functional requirements. Each feature was thoroughly tested to ensure reliability and performance in a real-world scenario.

## Challenges Faced

Developing the Intelligent Incident Resolution System presented a variety of challenges, ranging from technical complexities to time management issues. Each obstacle required careful analysis, creative problem-solving, and adjustments to the overall approach. The following sections provide a comprehensive overview of these challenges, and the strategies used to address them.

**Integration of Backend and Frontend**

Integrating the Spring Boot backend with the HTML, CSS, and JavaScript frontend introduced several challenges related to data formatting and asynchronous communication. Early attempts led to failed API requests and confusing errors due to mismatched expectations between JSON payloads and server-side logic.

To resolve these issues, JSON was adopted as the standard data format, and the Fetch API in JavaScript was used for consistent asynchronous communication. Robust error handling and endpoint testing with Postman helped smooth out integration and improve user experience.

**NLP-Based Similarity Analysis**

Developing the NLP-based similarity feature was one of the most complex parts of the project. Early difficulties included inconsistent results in tokenization, vector generation, and similarity scoring, particularly when comparing unstructured text with varying terminology and formatting.  
To address this, custom preprocessing was implemented, including stop-word removal, synonym expansion, and normalization. The system uses GloVe word embeddings combined with TF-IDF to calculate semantic similarity. Balancing performance and accuracy required optimizations such as scoring thresholds, caching, and weighted similarity blending.

**Database Configuration and Security**

Initial MySQL setup posed problems due to incorrect application.properties configurations, causing connection errors. Security issues were also flagged early, including risks of SQL injection and plaintext password storage.

These were mitigated by introducing parameterized queries and using the bcrypt algorithm for secure password hashing. These changes helped ensure both functional and secure data management.

**Time Management**

Balancing development with academic deadlines proved difficult, especially when unforeseen scope changes occurred. Some features took longer than anticipated due to technical setbacks.  
The project was broken into smaller, modular components to manage workload. A project timeline with clearly defined milestones helped track progress and keep the implementation on schedule.

**Debugging and Troubleshooting**

Numerous bugs emerged during development, often due to syntax errors, logic flaws, or incomplete API responses. Some were difficult to replicate, particularly those triggered by edge cases.  
Using IntelliJ IDEA’s debugger and adding detailed logging helped isolate issues efficiently. Frequent testing in both development and production modes helped ensure long-term system stability.

## Conclusion

The implementation of the Intelligent Incident Resolution System represents a significant achievement in building a secure, intelligent, and scalable solution for managing IT incidents. By leveraging a robust backend using Java and Spring Boot, a responsive frontend with HTML, CSS, and JavaScript, and a well-structured MySQL database, the system integrates multiple components seamlessly.

A key innovation lies in its NLP-based semantic matching functionality, which combines GloVe word embeddings and TF-IDF weighting to identify similar incidents and suggest past resolutions. This enhances administrative efficiency, reduces time to resolution, and improves support quality. The project also addresses critical considerations such as role-based access control, secure authentication, and responsive UI feedback.

By adopting an iterative development process, the system successfully met its functional requirements while addressing real-world software engineering challenges, including integration, scalability, and performance optimization. The use of modern techniques and thoughtful architectural choices make it a strong candidate for real-world deployment.

Looking ahead, future improvements could include refining the frontend interface, integrating more advanced NLP models (e.g., BERT), or expanding the system to handle additional support domains beyond IT. Overall, the system provides a solid foundation for intelligent support automation and demonstrates a practical application of machine learning and NLP in enterprise environments.

# Testing / Results

## Introduction

Testing is a crucial phase in software development, ensuring that the Intelligent Incident Resolution System operates as expected and meets all defined requirements. It provides validation that the system’s components function correctly both individually and as part of the larger framework. By performing rigorous testing, potential issues can be identified and resolved before deployment, improving the overall quality, reliability, and usability of the system.

This chapter outlines the **functional and non-functional testing** performed on the system, detailing the testing methodologies, results, and areas requiring further refinement. The objectives of testing included:

* **Ensuring functional correctness** – verifying that core features, such as incident submission, retrieval, and role-based access, work as intended.
* **Assessing system performance** – measuring response times, system load handling, and database query efficiency.
* **Validating security measures** – checking authentication, data protection, and unauthorised access prevention.
* **Testing usability and availability** – evaluating how intuitive the system is and ensuring it remains accessible under various conditions.

Each test was carefully planned to align with real-world usage scenarios, ensuring that the system meets user expectations and industry standards.

**Unit Testing of Non-Functional Requirements**

Unit testing is an essential software testing practice that focuses on verifying the smallest testable components of a system. Although a full unit testing suite was not implemented, key aspects of the system were manually tested to ensure fundamental operations behaved as expected.

**Approach to Unit Testing**

Unit tests primarily focused on the following areas:

* **Incident Submission and Retrieval** – Ensuring that individual functions correctly store and retrieve incident data.
* **Authentication and Role-Based Access Control** – Verifying that users are correctly authenticated and assigned appropriate permissions.
* **NLP-Based Matching Algorithm** – Checking that the system correctly matches incidents based on keyword similarity.

**Execution and Findings**

* **Basic validation tests were conducted manually** to check data integrity and logical correctness.
* **Sample test cases were created** to ensure database operations (e.g., inserting and fetching incidents) were correctly handled.
* **Authentication mechanisms were reviewed**, confirming that unauthorised users could not access restricted features.
* **NLP-based similarity matching was validated**, confirming that relevant incidents were retrieved in the expected order.

While formal unit tests were not fully developed, future improvements could include implementing automated unit testing using Junit for backend logic validation, ensuring higher reliability and reducing manual testing efforts.

## Non-Functional Test Cases

Non-functional testing evaluates the system’s overall quality, including performance, security, usability, and availability. The following test cases were conducted to measure these aspects.

**Performance Testing**

The system should allow users to navigate between different pages smoothly without noticeable delays or lag. This test ensures that users can move between key sections (e.g., Dashboard, Incident Submission, Incident Search) efficiently.

|  |  |
| --- | --- |
| Test Case Name | System Navigation Performance |
| Test Case ID | TC-NFUNC-01 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Measures how quickly the system loads and switches between different pages. |
| Pre-Condition | User must be logged in and the system must be running. |
| Inputs | Click through multiple pages (Dashboard, Incident Submission, Search, Admin Panel). |
| Post-Condition | Pages should load within an acceptable time (e.g., under 2 seconds). |
| Status | Pass |
| Notes/Comments | Test was also performed on different network conditions (Wi-Fi, mobile data). |

The system’s database should efficiently retrieve and process incident data, ensuring fast responses when fetching information.

|  |  |
| --- | --- |
| Test Case Name | Database Query Execution Speed |
| Test Case ID | TC-NFUNC-02 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Evaluates the execution time of queries related to incident retrieval and NLP-based matching. |
| Pre-Condition | Database must have a significant number of stored incidents (e.g., 10,000+). |
| Inputs | Run SELECT queries for incident retrieval and NLP-based similarity searches. |
| Post-Condition | Queries should execute within an acceptable timeframe (e.g., under 2 seconds). |
| Status | Pass |
| Notes/Comments | Performance logs should be analysed to identify slow queries. |

Users should be able to submit incidents without experiencing noticeable lag or delays in processing.

|  |  |
| --- | --- |
| Test Case Name | Incident Submission Response Time |
| Test Case ID | TC-NFUNC-03 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Measures the time taken for an incident to be successfully submitted and stored. |
| Pre-Condition | User must be logged in and connected to the system. |
| Inputs | Submit an incident with a title, description, and severity level. |
| Post-Condition | The system should confirm submission within an acceptable time (e.g., under 3 seconds). |
| Status | Pass |
| Notes/Comments | Test should be repeated under different loads to observe potential slowdowns. |

Users should be able to search for incidents quickly using their **incident number**, with results displaying relevant information without long wait times.

|  |  |
| --- | --- |
| Test Case Name | Incident Search Performance |
| Test Case ID | TC-NFUNC-04 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Evaluates how quickly the system returns search results for incidents when searching by incident number. |
| Pre-Condition | The database must contain a variety of incident records. |
| Inputs | Enter a valid incident number into the search field. |
| Post-Condition | Search results should display the corresponding incident within an acceptable time (e.g., under 2 seconds). |
| Status | Pass |
| Notes/Comments | The test should also check search accuracy along with performance. |

**Security Testing**

The system should prevent unauthorised users from accessing admin functionalities such as managing incidents.

|  |  |
| --- | --- |
| Test Case Name | Unauthorised Access Prevention |
| Test Case ID | TC-NFUNC-05 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that non-admin users cannot access restricted areas such as the Admin Dashboard or modify incidents. |
| Pre-Condition | The system must have at least one admin and one non-admin user. |
| Inputs | Attempt to access admin features (e.g., assigning incidents, modifying incident status) while logged in as a non-admin. |
| Post-Condition | The user is denied access and redirected or shown an error message. |
| Status | Pass |
| Notes/Comments | Ensure proper role-based authentication using JWT and backend validation. |

The system should restrict repeated failed login attempts to prevent brute force attacks.

|  |  |
| --- | --- |
| Test Case Name | Login Attempt Restrictions |
| Test Case ID | TC-NFUNC-06 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that users are blocked or delayed after multiple failed login attempts. |
| Pre-Condition | The system must have a login page with authentication enabled. |
| Inputs | Enter an incorrect password 5 times in a row. |
| Post-Condition | The system should block further login attempts temporarily or display a CAPTCHA. |
| Status | Pass |
| Notes/Comments | Verify if lockout duration is enforced and if a proper error message is shown. |

**Usability Testing**

The system should allow both **regular users** and **admins** to navigate efficiently, with access to the appropriate sections based on their roles.

|  |  |
| --- | --- |
| Test Case Name | User and Admin Navigation and Ease of Use |
| Test Case ID | TC-NFUNC-07 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Evaluates how easily users and admins can navigate between key sections, ensuring they only see features they have permission to access. |
| Pre-Condition | The system must have both a user and an admin logged in separately. |
| Inputs | Navigate through different pages as both a user and an admin. Users should attempt to access admin-only pages. |
| Post-Condition | Regular users can only access incident submission and their own incidents. Admins can view and manage all incidents. |
| Status | Pass |
| Notes/Comments | Ensure that restricted pages properly redirect non-admin users and do not expose admin functionalities. |

The system should provide clear validation messages when users enter incorrect or incomplete information.

|  |  |
| --- | --- |
| Test Case Name | Form Validation and Error Messaging |
| Test Case ID | TC-NFUNC-08 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that users receive appropriate error messages when submitting invalid or incomplete data. |
| Pre-Condition | Incident submission form must be accessible. |
| Inputs | Submit an incident without a description or severity level. |
| Post-Condition | The system should display a clear error message indicating the missing fields. |
| Status | Pass |
| Notes/Comments | Ensure that error messages are user-friendly and guide the user on how to fix the issue. |

Users should be able to submit incidents quickly, with minimal steps required.

|  |  |
| --- | --- |
| Test Case Name | Incident Submission Workflow Efficiency |
| Test Case ID | TC-NFUNC-09 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Measures how quickly users can complete an incident submission without unnecessary steps. |
| Pre-Condition | The user must be logged in. |
| Inputs | Submit an incident following the default workflow. |
| Post-Condition | The incident is submitted successfully in under 30 seconds. |
| Status | Pass |
| Notes/Comments | Check if unnecessary fields or steps slow down the submission process. |

**Availability Testing**

The system should remain available and responsive to users during normal operations, with minimal downtime.

|  |  |
| --- | --- |
| Test Case Name | System Uptime and Accessibility |
| Test Case ID | TC-NFUNC-10 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that users can access the system consistently without unexpected downtime. |
| Pre-Condition | The system must be running. |
| Inputs | Attempt to log in and access core features at different times of the day. |
| Post-Condition | The system remains accessible, and login/incident submission functions as expected. |
| Status | Pass |
| Notes/Comments | Monitor for any unexpected crashes, slow responses, or timeouts. |

The system should retain all incidents and user accounts after being restarted on a local machine.

|  |  |
| --- | --- |
| Test Case Name | System Restart and Data Retention |
| Test Case ID | TC-NFUNC-11 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that all stored incidents and user data persist after restarting the system locally. |
| Pre-Condition | The system must have existing incidents and user accounts. |
| Inputs | Close the application and restart it, then check if all data is still available. |
| Post-Condition | All incidents and user data remain intact and accessible. |
| Status | Pass |
| Notes/Comments | Ensure that no data is lost or reset upon restart. |

**Search and Filtering Testing**

When a user searches for a specific **incident number**, the system should display that exact incident **at the top**, followed by any **similar past incidents**.

|  |  |
| --- | --- |
| Test Case Name | Incident Search by Incident Number |
| Test Case ID | TC-NFUNC-12 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that searching for an incident number correctly displays that incident at the top, followed by related incidents based on similarity matching. |
| Pre-Condition | The system must have multiple stored incidents with some similar descriptions. |
| Inputs | Enter a valid incident number in the search field. |
| Post-Condition | The exact incident appears at the top, with related incidents displayed below. |
| Status | Pass |
| Notes/Comments | Ensure the similarity-matching algorithm returns relevant results in the correct order. |

Ensures that the system properly handles cases where a user searches for an **invalid or non-existent** incident number.

|  |  |
| --- | --- |
| Test Case Name | Handling of Invalid Incident Numbers in Search |
| Test Case ID | TC-NFUNC-13 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that the system properly handles searches for non-existent incident numbers. |
| Pre-Condition | The system must be running and have an incident search feature. |
| Inputs | Enter an invalid or non-existent incident number. |
| Post-Condition | The system displays an appropriate message (e.g., "Incident not found"). |
| Status | Pass |
| Notes/Comments | Ensure no errors occur, and the system remains responsive. |

## 

## Findings of Non-Functional Testing

The non-functional testing phase assessed the performance, security, usability, availability, and search functionality of the Intelligent Incident Resolution System. The results indicate that the system meets expectations in most areas, with some opportunities for further improvements.

**Performance Testing Findings**

The system demonstrated smooth and responsive navigation, with all pages loading within the expected timeframe (under two seconds) across different network conditions. Database query execution was efficient, with incident retrieval and NLP-based similarity searches completing within an acceptable time. Incident submission was processed correctly, with data stored successfully in under three seconds. Search performance was functional, though further optimisation may be required when handling larger datasets.

**Security Testing Findings**

Security testing confirmed that role-based access control is functioning correctly, preventing unauthorised users from accessing admin functionalities. The system effectively restricted multiple failed login attempts, reducing the risk of brute-force attacks. While the security mechanisms performed as expected, additional logging mechanisms for failed access attempts could further enhance monitoring and security.

**Usability Testing Findings**

The system was found to be intuitive and easy to navigate for both users and administrators, with appropriate role-based restrictions in place. Form validation operated correctly, ensuring users received clear error messages when required fields were missing. The incident submission process was streamlined, allowing users to report incidents with minimal steps. However, minor UI enhancements could improve the clarity of the submission process.

**Availability Testing Findings**

The system maintained consistent uptime with no unexpected crashes or errors. All incident data persisted correctly following system restarts on a local machine. While the system successfully retained data, implementing automated backup mechanisms could further enhance data reliability and recovery options.

**Search and Filtering Testing Findings**

The incident search functionality performed as expected, ensuring that the searched incident appeared at the top, followed by related incidents based on NLP similarity matching. Invalid incident searches were handled correctly, with appropriate messages displayed when a match was not found. While functional, performance optimisations may be required to improve response times when searching large datasets.

## Unit Testing of Functional Requirements

Unit testing was not extensively performed for this project, but some manual validation was conducted to ensure core functionalities operated correctly. The key areas covered in unit testing included:

* **Incident Submission and Retrieval** – Verifying that incidents were correctly stored and retrieved from the database.
* **User Authentication and Role-Based** Access – Ensuring that users were correctly authenticated and assigned the appropriate permissions.
* **Incident Search and NLP Matching** – Checking that the system accurately returned the searched incident and relevant past incidents based on similarity.

While formal unit tests were not fully developed, future improvements could include the implementation of automated unit testing using JUnit to improve test coverage and reliability.

## Functional Test Cases

Functional testing ensures that the core features of the Intelligent Incident Resolution System operate as intended, focusing on incident management, user authentication, role-based access control, and search functionality. The following test cases were conducted to verify that the system meets its expected functional requirements.

**Incident Management**

Incident reporting is a core feature of the system. This test ensures that users can submit incidents successfully while validating required input fields.

|  |  |
| --- | --- |
| Test Case Name | Incident Submission |
| Test Case ID | TC-FUNC-02 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that users and admins can submit incidents successfully. |
| Pre-Condition | User/Admin must be logged in. |
| Inputs | Submit an incident with a title, description, and severity level. |
| Post-Condition | The incident is stored in the database and assigned a unique incident number. |
| Status | Pass |
| Notes/Comments | Check if all required fields are validated. |

Admins should have the ability to update incident details, such as modifying severity levels.

|  |  |
| --- | --- |
| Test Case Name | Incident Update (Admin Only) |
| Test Case ID | TC-FUNC-03 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that only admins can update an existing incident's details. |
| Pre-Condition | The system must contain at least one incident, and the user must be logged in as an admin. |
| Inputs | Select an existing incident and modify its severity level. |
| Post-Condition | The updated information is saved and reflected when retrieving the incident. |
| Status | Pass |
| Notes/Comments | Verify that regular users cannot access or modify this feature. |

Only admins should be able to delete an incident from the system to maintain data integrity.

|  |  |
| --- | --- |
| Test Case Name | Incident Deletion (Admin Only) |
| Test Case ID | TC-FUNC-04 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that only admins can delete incidents from the system. |
| Pre-Condition | The system must contain at least one existing incident. |
| Inputs | Attempt to delete an incident as an admin. |
| Post-Condition | The incident is successfully removed only when performed by an admin. |
| Status | Pass |
| Notes/Comments | Verify that users receive a proper error message if they attempt to delete an incident. |

Admins should be able to update the status of an incident to track its progress.

|  |  |
| --- | --- |
| Test Case Name | Incident Status Change |
| Test Case ID | TC-FUNC-05 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that admins can modify the status of an incident (e.g., Open → In Progress → Resolved). |
| Pre-Condition | The system must have at least one existing incident. |
| Inputs | Select an incident and update its status. |
| Post-Condition | The status change is reflected correctly in the system. |
| Status | Pass |
| Notes/Comments | Verify that regular users cannot modify an incident’s status. |

Incidents should be assignable to a specific admin to facilitate resolution.

|  |  |
| --- | --- |
| Test Case Name | Incident Assignment to Admins |
| Test Case ID | TC-FUNC-06 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that incidents can be assigned to a specific admin for handling. |
| Pre-Condition | The system must contain at least one unassigned incident. |
| Inputs | Assign an incident to a specific admin user. |
| Post-Condition | The assigned admin is recorded, and their details appear in the incident view. |
| Status | Pass |
| Notes/Comments | Verify that the correct admin details are displayed after assignment. |

When searching for an incident, the system should display the searched incident at the top, followed by similar past incidents.

|  |  |
| --- | --- |
| Test Case Name | Incident Similarity Matching (NLP Feature) |
| Test Case ID | TC-FUNC-08 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that after searching for an incident, similar past incidents appear below the searched result. |
| Pre-Condition | The system must contain multiple stored incidents with some similarities in their descriptions. |
| Inputs | Search for an incident using its unique incident number. |
| Post-Condition | The searched incident appears at the top, with related incidents listed below based on NLP similarity scoring. |
| Status | Pass |
| Notes/Comments | Verify that the similarity algorithm returns relevant results in the correct order. |

**User Authentication & Role-Based Access**

Users and admins must be able to log into the system using credentials assigned by an admin.

|  |  |
| --- | --- |
| Test Case Name | User Login and Role Verification |
| Test Case ID | TC-FUNC-09 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that users and admins can log in using credentials assigned by an admin and that role-based access is correctly applied. |
| Pre-Condition | The system must be running, and user/admin accounts must already be created via Postman. |
| Inputs | Log in with valid and invalid credentials for both user and admin accounts. |
| Post-Condition | Valid users and admins can log in successfully, while incorrect credentials are denied access. |
| Status | Pass |
| Notes/Comments | Verify that role-based access is correctly assigned after login. |

Regular users should be restricted from performing admin actions such as updating or deleting incidents via the Admin Dashboard.

|  |  |
| --- | --- |
| Test Case Name | Unauthorised User Actions (Role-Based Access) |
| Test Case ID | TC-FUNC-10 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that regular users cannot access admin-only actions like incident updates, deletions, or assignments. |
| Pre-Condition | A user and an admin account must exist. |
| Inputs | Attempt to modify an incident as a regular user. |
| Post-Condition | The system denies access to unauthorised users. |
| Status | Pass |
| Notes/Comments | Verify that unauthorised access attempts are logged appropriately. |

The system should prevent unauthorised access by rejecting incorrect login attempts and providing appropriate error messages.

|  |  |
| --- | --- |
| Test Case Name | Invalid Login Attempt Handling |
| Test Case ID | TC-FUNC-11 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that incorrect login attempts are handled securely with proper error messages. |
| Pre-Condition | The system must have at least one user/admin account. |
| Inputs | Attempt to log in with an incorrect username/password combination. |
| Post-Condition | The system displays an error message and prevents access. |
| Status | Pass |
| Notes/Comments | Verify if multiple failed login attempts trigger any security measures. |

**System Functionality & Data Handling**

Users should not be able to submit an incident without providing all required details.

|  |  |
| --- | --- |
| Test Case Name | Handling of Missing Required Fields in Incident Submission |
| Test Case ID | TC-FUNC-12 |
| Test Priority | High |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that an incident cannot be submitted if required fields are missing. |
| Pre-Condition | The system must have an incident submission form with required fields. |
| Inputs | Submit an incident with missing title, description, or severity level. |
| Post-Condition | The system rejects the submission and displays an appropriate error message. |
| Status | Pass |
| Notes/Comments | Verify that error messages clearly indicate the missing fields. |

Users should receive an appropriate response when searching for an **invalid or non-existent** incident number.

|  |  |
| --- | --- |
| Test Case Name | Handling of Invalid Incident Numbers in Search |
| Test Case ID | TC-FUNC-13 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that searching for a non-existent incident number displays an appropriate message. |
| Pre-Condition | The system must have an incident search feature. |
| Inputs | Enter an invalid or non-existent incident number. |
| Post-Condition | The system displays an error message such as “Incident not found.” |
| Status | Pass |
| Notes/Comments | Ensure no crashes occur and the system remains responsive. |

Admins should be able to filter incidents by their status (e.g., Open, In Progress, Resolved) to manage incidents efficiently.

|  |  |
| --- | --- |
| Test Case Name | Incident Filtering by Status |
| Test Case ID | TC-FUNC-15 |
| Test Priority | Medium |
| Test Executed By | Adam Mcloughlin |
| Date of Test Execution | 20/02/2025 |
| Description/Summary | Ensures that admins can filter incidents by their status to streamline incident management. |
| Pre-Condition | The system must have multiple incidents with different statuses. |
| Inputs | Select a status filter (e.g., "Open", "Resolved") from the incident management panel. |
| Post-Condition | The system displays only incidents matching the selected status. |
| Status | Pass |
| Notes/Comments | Verify that the filter updates dynamically and displays the correct results. |

## Findings of Functional Testing

The functional testing phase verified the core features of the Intelligent Incident Resolution System, ensuring that incident management, user authentication, role-based access, and search functionalities performed as expected. The results confirm that the system successfully meets its functional requirements, with only minor areas identified for future improvements.

**Incident Management Findings**

* Incident submission and retrieval were successful, with all required fields validated before storing the data.
* Incident updates, deletions, and status changes were restricted to admin users only, preventing unauthorised modifications by regular users.
* Incident assignment functionality worked as expected, allowing admins to assign incidents to specific team members.
* NLP-based incident similarity matching performed correctly, displaying the searched incident first, followed by relevant similar incidents.

**User Authentication & Role-Based Access Findings**

* User authentication worked correctly, with valid credentials granting access and incorrect credentials being rejected.
* Role-based access control was enforced, ensuring that regular users could not perform admin-only actions.
* Unauthorised actions were correctly blocked, with the system providing appropriate error messages when users attempted to access restricted areas.

**System Functionality & Data Handling Findings**

* Incident filtering by status functioned as expected, allowing admins to sort incidents based on their current progress.
* The system properly handled invalid incident number searches, displaying a relevant "Incident not found" message.
* Form validation operated correctly, ensuring that users could not submit incidents with missing required fields.

The functional testing phase confirmed that the system meets its intended objectives, with core functionalities performing reliably. The incident management, authentication, role-based access, and search functionalities were all validated, ensuring that users interact with the system as expected. Minor refinements, such as UI optimisations and additional security logging, could further enhance the user experience and system robustness.

## Test Data

The test data used during functional and non-functional testing includes sample user accounts, incident records, and system interactions. The following subsections outline the key datasets and validation methods used.

**User Accounts for Testing**

The system operates with role-based access control, where admins manage incidents, and regular users can only submit and track their own incidents. The table below outlines the test users:

|  |  |  |  |
| --- | --- | --- | --- |
| User ID | Username | Role | Password |
| 1 | Admin | ROLE\_ADMIN | admin123 |
| 2 | User | ROLE\_USER | user123 |

Tested Role Restrictions:

* Admins can view, update, assign, and delete incidents
* Regular users can only submit incidents and view their own
* Unauthorised users attempting to access admin features receive a **"You must be an admin to access this page"** error message.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.1 Role-Based Access

**Sample Incidents Used for Testing**

Incidents were created to simulate real-world technical issues and validate system functions such as submission, retrieval, and search. The table below outlines some sample test incidents:

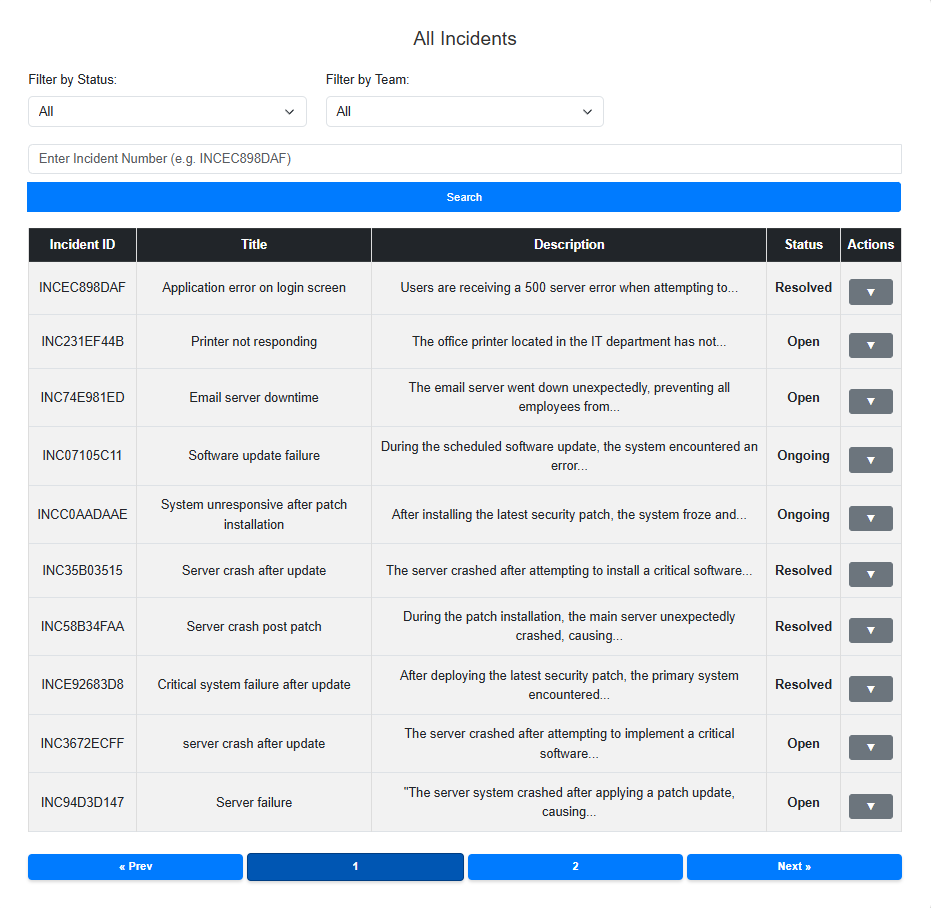


Figure 5.2 Sample Incidents used in Testing

Each incident has its own breakdown, with details such as the **Description, Time Submitted, Submitted By, Severity, Status,** and **Assigned Team and Admin:**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.3 Incident Breakdown and Details

**Incident Submission Test Data**

Incident submission was tested to ensure that users could successfully create new incidents, with the system properly validating required fields and storing submitted data. The test also verified that users received appropriate feedback upon successful submission.

A screenshot of a computer

AI-generated content may be incorrect.**Tested Form Validation**: Users were required to enter a short description, detailed description, and severity level before submission.

Figure 5.4 Incident Submission Form

**Tested Response**: A User/Admin is prompted with a confirmation message before submitting an incident.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.5 Incident Confirm Submission Response

Upon successful submission, the system displayed **"Incident created successfully.**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.6 Incident Submission Success Confirmation

**Verified Data Storage**: The submitted incident appeared in the **incident management table.**

This confirmation screen illustrates that the incident has been successfully stored in the system. Key fields such as severity, status, assigned team, and assigned admin are clearly displayed and can be updated throughout the incident’s lifecycle. The **"Resolve"** button remains active until the incident is officially resolved, allowing an admin to take action at any point.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.7 Unresolved Incidents displaying 'Resolve' Button

**Search and Similarity Matching Test Data**

The search and similarity matching feature was tested to confirm that the system correctly retrieved incidents based on their **incident number** and displayed **similar past incidents**. The test also checked how the system handled invalid search queries.

A screenshot of a search engine

AI-generated content may be incorrect.

Figure 5.8 Similarity Search Feature Results Returned

Invalid Incident Numbers cannot be fetched or compared:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.9 Invalid Incident Number Response

**Role-Based Access Validation**

Role-based access testing was conducted to verify that **admin users** could access all system functionalities while **regular users** were restricted from performing admin-only actions.

Admin access to the Admin Dashboard:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.10 Admin Access to Admin Dashboard

User Denied access to the Admin Dashboard:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.11 Regular User denied access

**Incident Filtering by Status**

Incident filtering was tested to ensure that users, particularly admins, could filter incidents by their **status** (e.g., Open, In Progress, Resolved).

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.12 Incident Filtering

Filtering is also viable by searching for an individual incident by its INC number:

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.13 Incident Search Filter

## Summary of Findings

The testing phase of the Intelligent Incident Resolution System was conducted to evaluate whether the system met its intended functional and non-functional requirements. A variety of tests were performed, focusing on incident management, user authentication, role-based access control, search functionality, performance, security, usability, and availability. Overall, the system functioned as expected, with only minor areas identified for improvement.

**Functional Testing Findings**

* **Incident Management:** The system successfully allowed users to submit incidents, while only administrators had the ability to update, delete, and assign incidents. This ensured that regular users could not modify or remove critical data, maintaining system integrity.
* **Search and Similarity Matching:** The search function retrieved exact incidents correctly, and the NLP-based similarity matching displayed relevant past incidents ranked by match percentage. Searches for invalid incident numbers resulted in the appropriate error message, preventing system failures.
* **User Authentication and Role-Based Access**: The system enforced role-based permissions correctly.
  + Valid users could log in and access the system.
  + Regular users were restricted to incident submission and tracking, while admins could manage all incidents.
  + Any unauthorised access attempts resulted in an appropriate error message, preventing security breaches.
* **Form Validation**: Users could not submit incomplete incidents, as the system correctly enforced required fields. This prevented incorrect or missing data from being stored in the system.

**Non-Functional Testing Findings**

* **Performance Testing**:
  + Navigation between pages was smooth, with all sections loading within an acceptable time.
  + Incident searches were completed within 1-2 seconds, ensuring quick access to incident data.
* **Security Testing**:
  + Role-based access control was enforced correctly, ensuring that non-admin users could not access or modify incidents they were not authorised to manage.
  + Login security measures worked as expected, preventing unauthorised access to the system.
* **Usability Testing**:
  + The interface was user-friendly, with intuitive navigation for both regular users and admins.
  + Error messages were clear and informative, guiding users when submitting incidents or performing restricted actions.
* **Availability Testing**:
  + The system successfully retained incident data after restarts, demonstrating reliability.
  + Filtering by incident status worked correctly, allowing users to display incidents based on their progress.

While the system met all key functional and non-functional requirements, some areas for potential improvement were identified. Enhancing the search algorithm to improve similarity ranking could make incident retrieval even more effective. Introducing automated unit testing in future development cycles could help detect potential issues earlier, increasing system reliability.

## Conclusion

The Intelligent Incident Resolution System was thoroughly tested to ensure it met its design objectives. The system was developed to provide a structured method for reporting, managing, and resolving incidents, with efficient search capabilities to assist administrators in identifying similar past issues. Testing confirmed that the system functioned effectively, enforcing proper authentication, access control, and data management.

Several key aspects of the system were validated:

* **Incident Management:** Users were able to report, update, and track incidents, while administrators had full control over modifying and assigning them.
* **Role-Based Security:** Admins had full access to the system’s management features, while regular users were restricted from making unauthorised changes.
* **Search Functionality:** The system correctly retrieved incidents based on their incident number, with related past incidents ranked by similarity.
* **System Performance:** Page navigation was smooth, and incident retrieval was quick and efficient, supporting real-time use in an organisational setting.
* **Data Integrity and Availability:** Incident records persisted after system restarts, ensuring reliability.
* **Usability and Error Handling:** The system provided clear validation messages, preventing incorrect data from being submitted. Error messages guided users when performing restricted actions.

While the system met its functional and non-functional requirements, future improvements could enhance its efficiency and scalability. Introducing automated unit testing with tools like JUnit would streamline validation and reduce manual testing. Refining the NLP-based similarity matching algorithm could further optimise search results, helping admins locate past incidents more accurately. Enhanced security logging would also offer better visibility into user activity and potential unauthorised access.  
In summary, the Intelligent Incident Resolution System delivers a reliable, secure, and user-friendly solution for incident management, supporting structured issue tracking, intelligent search, and controlled user access. With minor refinements, it holds strong potential for real-world deployment and future scalability.

# Future Work and Research

While the current system delivers a solid foundation for intelligent incident resolution, there are several areas worth exploring to enhance its accuracy, scalability, and usability. These improvements span across dataset development, natural language processing capabilities, system architecture, feature expansion, and security enhancements.

**Expanding the Dataset**

The system currently relies on a simulated dataset created for demonstration purposes. While this allows core functionality to be tested, it does not fully reflect the variation and complexity seen in real-world support environments.

Future improvement opportunities include:

* Sourcing anonymised real-world incidents from open datasets or simulated logs based on actual IT support cases.
* Increasing dataset diversity by including different categories, severity levels, and phrasing styles across departments (e.g., HR, IT, Finance).
* Scaling up the volume of training data to improve the accuracy of NLP-based similarity scoring.

These enhancements would improve the generalisability and reliability of similarity predictions, especially when handling unfamiliar or uncommon incidents.

**Advancing the NLP Engine**

The current system uses a hybrid semantic matching method combining TF-IDF and GloVe embeddings. While effective for basic keyword and similarity comparisons, these models lack deeper contextual understanding.

To enhance semantic intelligence:

* Introduce BERT or transformer-based models to better capture meaning, context, and intent across longer and more varied descriptions.
* Explore fine-tuning transformer models on your specific incident dataset to improve domain adaptation.
* Enable contextual disambiguation, allowing the system to distinguish between incidents with similar keywords but different meanings.

This would significantly increase the accuracy of incident matching, particularly in large organisations with complex terminology.

**Security and Access Control**

The existing authentication system provides basic user/admin access. For improved flexibility and data security, the system could evolve to include:

* Role-Based Access Control (RBAC):
  + Define custom roles (e.g., Supervisor, Analyst, Reviewer).
  + Assign permissions at a more granular level (e.g., edit only assigned incidents).
* Token-based authentication:
  + Implement secure token sessions using JWT or OAuth2.
  + Add token scopes to restrict access based on user type or role.
* Audit logs and access tracking:
  + Maintain records of user actions within the system.
  + Support accountability and traceability in enterprise settings.

These enhancements would make the system suitable for real-world deployment in organisations with stricter compliance requirements.

**System Architecture: Looking Ahead**

The current system operates as a monolithic application. While this works well during development, a microservices architecture would offer far better scalability and flexibility in future deployments.

Implementing Docker containers and Kubernetes orchestration would allow for independent deployment of core modules such as the incident manager, NLP engine, and user authentication. This setup would make the system more maintainable and robust, particularly in enterprise or production environments.

The table below outlines a high-level comparison between the current implementation and proposed future improvements:

|  |  |  |
| --- | --- | --- |
| Feature | Current Implementation | Future Improvement |
| System Structure | Monolothic | Microservices with Docker and Kubernetes |
| NLP Approach | TF-IDF + GloVe | BERT or transformer-based models |
| Dataset Size | Simulated Data | Real-world anonymised incident data |
| Access Control | Basic user/admin roles | Role-Based Access Control (RBAC) |
| Incident Matching | Keyword + Semantic scoring | Deep semantic understanding (BERT) |
| Language Support | English Only | Multilingual NLP |

**Feedback and Continuous Improvement**

After deployment, real-world usage will reveal how the system performs under practical conditions. Gathering user feedback will be essential to identify usability issues, missing features, or performance bottlenecks.

Establishing feedback loops and incorporating a regular update cycle based on stakeholder input will ensure that the system continues to evolve in line with user needs and expectations.

# Table of Figures and Tables

[Figure 2.1 Accuracy Results (SVM) Silva et al. (2024) 17](#_Toc196042093)

[Figure 2.2 Accuracy Results (KNN), Silva et al. (2024) 17](#_Toc196042094)

[Figure 2.3 Overall Accuracy with SVM & KNN, Silva et al. (2024 18](#_Toc196042095)

[Figure 2.4 Comparison of models, Prihandono et al. (2020) 18](#_Toc196042096)

[Figure 2.5 Comparison Chart of Accuracy between Traditional Method and Improved Traditional Method, Zhang et al. (2022) 23](#_Toc196042097)

[Figure 2.6 Pattnaik et al. (2019) Text Summarization Process 23](#_Toc196042098)

[Figure 2.7 Gawhankar et al. (2024) Proposed System for Resume Text Retrieval 24](#_Toc196042099)

[Figure 2.8 Systems Steps Explained, Gawhankar et al. (2024) 25](#_Toc196042100)

[Figure 2.9 Levels in text classification systems, Shetty et al. (2024) 27](#_Toc196042101)

[Figure 2.10 Eckstein et al. (2016) Approach design 28](#_Toc196042102)

[Figure 2.11 Rinartha and Kartika (2022) Word Frequency Algorithm 28](#_Toc196042103)

[Figure 2.12 Solution Architecture, Beyranvand and Aytekin (2020) 29](#_Toc196042104)

[Figure 3.1 Presentation, Application, and Data Layer Diagram 38](#_Toc196042105)

[Figure 3.2 Data Flow Diagram 39](#_Toc196042106)

[Figure 3.3 Relationships between key components 43](#_Toc196042107)

[Figure 3.4 Database Schema Relationships Diagram 47](#_Toc196042108)

[Figure 3.5 Submit Incident Page Mockup 50](#_Toc196042109)

[Figure 3.6 Admin Dashboard Mockup 50](#_Toc196042110)

[Figure 3.7 Incident Details Breakdown Mockup 50](#_Toc196042111)

[Figure 3.8 Filtered Incidents Mockup 51](#_Toc196042112)

[Figure 3.9 Similarity Results Mockup 51](#_Toc196042113)

[Figure 4.1 System Architecture: Overview of user interactions, frontend, backend, database, and NLP module 57](#_Toc196042114)

[Figure 4.2 Frontend Architecture Diagram 61](#_Toc196042115)

[Figure 4.3 Backend Architecture Diagram 62](#_Toc196042116)

[Figure 4.4 Database Connection Configuration 66](#_Toc196042117)

[Figure 4.5 User Creation and Role Retrieval Code Snippet 69](#_Toc196042118)

[Figure 4.6 Code Snippet for Role-based Access Control Implementation 70](#_Toc196042119)

[Figure 4.7 Code Snippet for Incident Submission Endpoint 71](#_Toc196042120)

[Figure 4.8 Code Snippet for Updating Incident Statuses 71](#_Toc196042121)

[Figure 4.9 Preprocessing Text Code Snippet 72](#_Toc196042122)

[Figure 4.10 TF-IDF Code Snippet 72](#_Toc196042123)

[Figure 4.11 Cosine Similarity Code Snippet 73](#_Toc196042124)

[Figure 4.12 The Comparison of words in Incidents 74](#_Toc196042125)

[Figure 4.13 Blended Similarity Score - TF-IDF and Word Semantics 75](#_Toc196042126)

[Figure 5.1 Role-Based Access 92](#_Toc196042127)

[Figure 5.2 Sample Incidents used in Testing 93](#_Toc196042128)

[Figure 5.3 Incident Breakdown and Details 94](#_Toc196042129)

[Figure 5.4 Incident Submission Form 95](#_Toc196042130)

[Figure 5.5 Incident Confirm Submission Response 95](#_Toc196042131)

[Figure 5.6 Incident Submission Success Confirmation 96](#_Toc196042132)

[Figure 5.7 Unresolved Incidents displaying 'Resolve' Button 96](#_Toc196042133)

[Figure 5.8 Similarity Search Feature Results Returned 97](#_Toc196042134)

[Figure 5.9 Invalid Incident Number Response 98](#_Toc196042135)

[Figure 5.10 Admin Access to Admin Dashboard 98](#_Toc196042136)

[Figure 5.11 Regular User denied access 99](#_Toc196042137)

[Figure 5.12 Incident Filtering 99](#_Toc196042138)

[Figure 5.13 Incident Search Filter 100](#_Toc196042139)

[Table 4.1 Libraries and Dependencies 60](#_Toc196042140)

# Bibliography

Atlassian, 2024. The importance of incident management. [online] Available at: <https://www.atlassian.com/incident-management#the-importance-of-incident-management> [Accessed 27 July 2024].

Beyranvand, P., & Aytekin, T. (2020). Automating Customer Claim Registration by Text Mining. Innovations in Intelligent Systems and Applications Conference (ASYU), Istanbul, Turkey, 2020, pp. 1-5.

BigPanda, 2024. *How Abbott Transformed Its Incident Management Process with Workflow Automation*. [online] Available at: <https://www.bigpanda.io/blog/how-abbott-transformed-its-incident-management-process-with-workflow-automation/> [Accessed 28 July 2024].

Blameless, 2024. *Automated Incident Management: Benefits and Challenges*. [online] Available at: <https://www.blameless.com/blog/automated-incident-management#:~:text=Automated%20incident%20management%20is%20the,automated%20incident%20management%20is%20speed> [Accessed 27 July 2024].

Eckstein, L., Kuehl, N., & Satzger, G. (2016). Towards Extracting Customer Needs from Incident Tickets in IT Services. IEEE 18th Conference on Business Informatics (CBI), Paris, France, 2016, pp. 200-207.

Fernandes, D., & Pizzutti, C. (2010). Effect of Recovery Efforts on Consumer Trust and Loyalty in E-Tail: A Contingency Model. International Journal of Electronic Commerce, 14(4), 127–160. Available at: https://doi.org/10.2753/JEC1086-4415140405

Gawhankar, K., Deorukhkar, A., Miniyar, A., Kapure, H., & Ivin, B. (2024). NLP-Driven ML for Resume Information Extraction. IEEE 9th International Conference for Convergence in Technology (I2CT), Pune, India, 2024, pp. 1-6.

Harwahyu, R., Prihandono, M. A., & Sari, R. F. (2020). Performance of Machine Learning Algorithms for IT Incident Management. 11th International Conference on Awareness Science and Technology (iCAST), Qingdao, China, 2020, pp. 1-6.

Li, H., & Zhan, Z. (2012). Machine Learning Methodology for Enhancing Automated Process in IT Incident Management. IEEE 11th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 2012, pp. 191-194.

Pattnaik, S., & Nayak, A. K. (2019). Summarization of Odia Text Document Using Cosine Similarity and Clustering. International Conference on Applied Machine Learning (ICAML), Bhubaneswar, India, 2019, pp. 143-146.

Pereira, R., Ribeiro, R., & Silva, S. (2018). Machine learning in incident categorization automation. 13th Iberian Conference on Information Systems and Technologies (CISTI), Caceres, Spain, 2018, pp. 1-6.

Prathyakshini & Shetty, J. (2024). DeepText: Pioneering the Future of Text Classification with Innovative Deep Learning Techniques. 5th International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2024, pp. 911-917.

Rinartha, K., & Kartika, L. G. S. (2021). Rapid Automatic Keyword Extraction and Word Frequency in Scientific Article Keywords Extraction. 3rd International Conference on Cybernetics and Intelligent System (ICORIS), Makasar, Indonesia, 2021, pp. 1-4.

ServiceNow, 2023. *3 Benefits of Automated Issue Response*. [online] Available at: <https://www.servicenow.com/blogs/2023/3-benefits-automated-issue-response> [Accessed 28 July 2024].

Sharan, Aditi, & Siddiqi, Sifatullah. (2015). Keyword and Keyphrase Extraction Techniques: A Literature Review. International Journal of Computer Applications. 109(2), 18-23.

Software Engineering Institute, 2024. *Top 5 Incident Management Issues*. SEI Insights. Available at: <https://insights.sei.cmu.edu/blog/top-5-incident-management-issues/> [Accessed 27 July 2024].

W3C, 2024. RESTful Web Services. [online] Available at: <https://www.w3.org/2001/sw/wiki/REST> [Accessed 16 July 2024].

Wang, F., Zhang, J., Ma, F., & Song, G. (2022). Text Similarity Calculation Method Based on Optimized Cosine Distance. International Conference on Electronics and Devices, Computational Science (ICEDCS), Marseille, France, 2022, pp. 37-39.

Wen, S., et al. (2022). Sentiment Analysis of Social Media Comments based on Random Forest and Support Vector Machine. IEEE International Conference on Advances in Electrical Engineering and Computer Applications (AEECA), Dalian, China, 2022, pp. 277-281t.