UNIVERSITY OF LEICESTER

School of Computing and Mathematical Sciences

CO7201 Individual Project

Final Report Template

Software Bug Tracking and Reporting Tool

Harishmitha Raja

hr200@student.le.ac.uk

239053031

Project Supervisor: Dr. James Hoey
Principal Marker: Dr. Newman Lau

Word Count: 9277
28/04/2025

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Harishmitha Raja
Date: 28/04/2025

ABSTRACT

The BugTrackR - Software Bug Tracking and Reporting Tool is a role-based web application developed to streamline the process of reporting, assigning, and resolving software bugs. Existing bug tracking platforms are often complex (even simple bug reporting requires administrators to pre-configure workflows, issue types, field screens, and permissions. This setup overhead slows down adoption, especially for teams needing a fast, structured, and role-specific bug tracking system) or too generalised (lacking built-in role-specific workflows for developers, testers, and non-technical users). This system addresses those issues by creating a more structured and flexible solution designed specifically for managing bugs across multiple applications with dedicated roles and workflows for general users, developers, testers, team leads, and admins.

Key objectives include secure login with OTP verification for reset password, role-based dashboards tailored to each user type. Detailed bug reporting tailored to technical and non-technical users and role-based access ensures each user can only take actions appropriate to their role. Bugs can be manually assigned to developers and testers or automatically assigned, depending on workload, seniority, and team to make sure it's fair. Bug reports can be tracked using a stepper interface which supports better visibility of bug status. Priority levels can be set manually assigned automatically classified. Manual and NLP-based contextual duplicate detection features are included to reduce repeated entries and improve productivity of developers and testers.

The system enables tech team collaboration via structured comments, with mention tagging and email alerts. Separate views for reported, assigned, mentioned bugs etc., are supported for better traceability. Developers and testers can request bug reallocation and bug reopen, and team leads or admins can approve or reject requests. Ability for critical bugs to be closed by admin or team lead is also implemented to prevent early closure of incorrectly fixed bug.

Advanced filtering and contextual search allow teams to efficiently locate bugs based on status, severity, or assigned developer/tester. Real-time scheduled alerts are triggered when bugs remain inactive or unassigned for too long. Email notifications are sent for key events like assignments, comments, and critical bug approval. Role-based analytics and export options allow team leads and admins to monitor team performance.

Optional features such as chat-based communication between reporters and tech users, and bug favouriting enhance collaboration and usability.

The project was developed using the Waterfall model, with some iterative improvements during implementation. The system ensures clarity in bug ownership, reduces redundancy, and improves communication between teams, resulting in a more organised and responsive bug tracking workflow.

Table of Contents

1.Introduction 6
1.1Motivation 6
1.2 Aims and Objectives 6
1.3 Challenges 7
1.4 Risks 8
1.5 System Overview 9
1.5.1 Application Purpose and Structure 9
1.5.2 Key Stakeholders and Functional Access 10
1.5.3 Workflow and Functional Overview 10
2.Background Research 12
2.1. Technical Research and Technology Justification 12
2.2 Review of Related Research 15
2.3 Evaluation of Existing Tools 17
2.4 Project Positioning in Context 18
3.Requirements 18
3.1 Project Requirements 18
3.2 System-Level Requirements 22
3.2.1 Non-Functional Requirements 23
3.3 Changes to Requirements During Development 23
4. Technical Specification 25
4.1 Languages and Frameworks 27
4.2 Libraries and Packages 27
4.3 Database 28
4.4 Tools 28
5. Development Methodology and Project Execution 28
5.1 Overview of Development Methodology 28
5.2 Why Waterfall Was Chosen 29
5.3 Adaptations Made During Implementation 30
6. Design 31
6.1 Architecture Design 31
6.1.1 Client-Server Architecture 31
6.2 Database Design 32
6.3 Use Case Diagram 32
6.4 System Module Breakdown 32
6.5 Logo and Branding 32

7. Implementation 33
7.1 Frontend Implementation 33
7.2 Backend Implementation 33
7.3 Python API for NLP 33
7.4 Notable Code Snippets 33
7.5 Running the Application 33
8. Testing and Evaluation 33
8.1 Requirements Evaluation Plan 33
8.2 Unit Testing 33
8.3 Functional Testing 33
8.4 Integration Testing 33
8.5 Limitations and Edge Cases 33
8.6 Supervisor/User Feedback 33
8.7 Testcases 33
9. Critical Reflection 34
9.1 What Worked and What Didn't 34
9.2 Decisions Made and Their Impacts 34
9.3 Challenges Overcome and Lessons Learned 34
9.4 Recommendations for Future Work 34
10. Conclusion 34
11. References 34

1.Introduction

1.1 Motivation

Effective bug management plays a critical role in ensuring the quality, reliability, and maintainability of software systems. However, in many real-world projects, the bug tracking process suffers from several inefficiencies that negatively affect developer productivity and user satisfaction. Studies show that **vague** or **incomplete** bug reports, **delayed assignments**, **redundant** submissions, and **lack of structured communication** are common issues that lead to wasted effort and slower resolution times [1][3][4].

Manual bug assignment, though often more accurate, is **time-consuming** and becomes difficult to manage when the number of bug reports increases. On the other hand, fully automated triaging (assignment) systems may **misallocate** bugs because they often **lack the contextual** understanding of developer workload, skill set, and team structure [2]. Duplicate bug reports are another major concern, they clutter the system, **increase backlog** count, and **consume unnecessary** developer time and reduce productivity [3]. As reported by Bettenburg et al., duplicates often go unnoticed due to poor detection mechanisms or limited support for contextual similarity checks.

In addition to these technical challenges, **communication gaps** within bug tracking systems remain a persistent issue. When collaboration is unstructured or happens **outside the system** (e.g., via email, third-party tools like MS Teams or Cisco Webex, or informal discussions), critical context and technical details are often lost. Research highlights that unclear, unfocused, or inaccessible comments significantly hinder the debugging process and contribute to delays in issue resolution [4].

Furthermore, as bug volumes grow in large or active projects, scalability and usability become critical. Many systems become **slower**, more **difficult to navigate**, or fail to maintain efficiency when subjected to high data loads, ultimately impacting both developer experience and decision-making quality [5].

These real-world problems motivated the design and development of BugTrackR (Bug Tracking + Reporting), a role-based bug tracking and reporting tool that aims to address the shortcomings observed in existing systems. By offering structured workflows, advanced duplicate detection techniques, intelligent assignment algorithm, and in-bug communication, BugTrackR seeks to reduce redundancy, improve collaboration, and provide a scalable, user-centered solution that supports both technical and non-technical users.

1.2 Aims and Objectives

The aim of this project is to design and develop a role-based software bug tracking and reporting tool, BugTrackR, that **improves** how software defects are **reported**, **tracked**, and **resolved**. The system focuses on making bug reporting more **user-friendly** by **tailoring the** interface and form design to different user roles and by reducing the effort required to submit complete, high-quality reports. It also aims to minimize redundancy through **smarter duplicate detection**, improve collaboration through **integrated communication** features, and support better decision-making through analytics and structured workflows.
To achieve this aim, the following objectives were set:

- Support **structured**, **role-based workflows** by providing distinct dashboards and permissions for general users, developers, testers, team leads, and admins.
- Improve bug reporting quality through **guided input forms** tailored to technical and non-technical users, and automatic environment detection (e.g., browser, OS).
- Enable efficient bug allocation using an assignment algorithm that considers **developer/tester workload, seniority, team,** and bug **priority**.
- Minimize redundancy through both manual and automated duplicate detection, using **semantic similarity** techniques powered by NLP (Natural Language Processing).
- Enhance communication within the system by offering both an **internal comment** system for technical users and a **chat interface** for interactions between reporters and technical roles.
- Provide real-time **scheduled** updates and alerts through features like **email notifications** and **visual indicators** for **inactivity** or **pending updates**.
- Offer **insights** through analytics, such as **resolution trends**, **workload** summaries, and **performance** dashboards for developers, testers, team leads and admin.
- Ensure **data security** and **access control** by implementing secure **authentication**, **encrypted communication**, and **role-based access** enforcement.

These objectives guided the system throughout its development and informed many of the design decisions, ensuring that BugTrackR remains practical, scalable, and relevant to real-world software teams.

1.3 Challenges

- **New to NLP Techniques for Duplicate Detection**
  Implementing duplicate bug detection using natural language processing (NLP) was one of the most challenging parts of the project, especially since there was no prior experience with Python-based NLP tools. Simpler methods were explored like WordNet, the Lesk algorithm, and Cosine similarity but these did not produce accurate results for bug-related content. Based on testing and feedback from the principal marker, I switched to a more advanced approach using Sentence Transformers for understanding context in bug descriptions, and FAISS to perform fast and scalable similarity searches. This change not only improved detection accuracy but also introduced me to building and integrating a Python-based FastAPI microservice alongside my Node.js backend, a valuable learning experience that pushed me beyond my existing skillset.

- **Limited API & Database Experience**
  Working with backend development using REST APIs and MongoDB was a fairly new experience for me at the beginning of the project. One challenge was learning how to properly organize the server code and manage different types of data for various user roles. I used Express.js to define the API routes for different features like bug reporting, status updates, assignments, and analytics. To connect the server with the database, I used Mongoose, which helped me define clear structure (schemas) for each type of data, such as what fields each bug report must contain, what type each field should be (e.g., string, number), and what validations are needed such as which fields are required, what data types are allowed, and what constraints (accepted values) should be applied. I also separated the route logic into different files based on their functionality/roles, which

made the code cleaner and easier to manage. These practices helped me build reliable APIs that supported all core operations of the system.

- **Ensuring Security & Access Control**
  Setting up secure access control across multiple user roles was also technically challenging and required detailed planning. I implemented JWT (JSON Web Token), ensuring that tokens were correctly issued, verified, and expired when needed. Middleware functions were created to check access based on user roles before allowing actions like assigning bugs, approving reopen requests, or viewing analytics. Additionally, the password reset flow was built using email-based OTP verification, adding an extra layer of security. I also configured the project to use HTTPS, enabling secure data transmission. These security features helped enforce data integrity, prevent unauthorized actions, and ensure that each user only had access to the features relevant to their role.

- **Balancing Manual & Automated Bug Assignment**
  Designing the bug assignment logic was one of the more technically demanding tasks in the project. It involved creating a structured algorithm that could account for factors like developer workload, team specialization, priority level, and seniority. The algorithm was designed and implemented to behave intelligently, distributing bugs fairly while allowing overrides by leads and admins. The logic had to be carefully structured and tested to handle different scenarios, including worst-case cases such as multiple high-priority bugs being reported at once. This challenge was addressed through a combination of workload tracking (through analytics page), fallback paths (the next available person), and user-triggered reallocation workflows (if the auto assigned person is not able to handle the bug).

- **Real-Time Collaboration**
  Real-time communication was a new area of development for me, and implementing this feature using Socket.io required understanding WebSocket connections, event handling, and state synchronization across clients. This was initially challenging, but I was able to learn and apply the concepts effectively to build a responsive, in-app chat feature. The system now supports two types of communication: internal discussions among technical users via comments (asynchronous), and a dedicated real-time chat channel between bug reporters and tech users. This significantly improved the flow of information and reduced dependency on asynchronous communication like email.

1.4 Risks

While planning and developing BugTrackR, several risks were identified that could potentially impact the quality or timely delivery of the project. These were considered early and monitored throughout the development process to minimize their effects.

- **Learning Curve for New Technologies:**
  One of the primary risks was the need to work with technologies and tools that I had limited experience with, such as Python-based NLP libraries, FastAPI, and FAISS for duplicate detection. This required additional learning time, and any unexpected issues could have delayed the implementation of key features.

- **Ambitious Scope:**
  The project began with an extensive list of essential, recommended, and optional features. Given the fixed timeline and the fact that this was an individual project, there was a clear risk of scope creep, especially when working with modules like automated assignment, role-based analytics, and contextual search. Additionally, several new features were added during development, including the reopen request workflow, bug change history storage, list views for reported and assigned bugs, a view for mentioned bugs, and a chat interface for communication between reporters and technical users. While these additions expanded the scope beyond the original plan, they were introduced to address practical gaps observed during implementation. To manage this expanded scope, two originally planned features, one desirable and one optional, were dropped after reassessment, allowing the project to remain focused and achievable within the available timeframe.

- **Integration Complexity:**
  BugTrackR includes multiple modules such as authentication, role-based dashboards, internal comments, real-time chat, duplicate detection using a separate Python microservice, email notification an more. Integrating these components smoothly was technically challenging, and there was a risk of runtime errors or incompatibility between services.

- **Testing Limitations:**
  Due to time constraints and the size of the project, thorough automated testing could not be fully implemented across all modules. To mitigate this, extensive manual validation was carried out throughout development, with focused testing after each major feature was completed. End-to-end workflow testing was also performed to ensure that interactions between components behaved as expected. Several issues were identified during this process and were resolved promptly. This approach helped maintain the stability of the application as features were added and served as an effective substitute for automated testing under time constraints.

- **Real-Time Features and Performance:**
  Socket-based communication and semantic similarity matching required asynchronous processing and good response times. Any issues in these real-time or performance-sensitive components could have led to degraded user experience. Risk was reduced by using MongoDB, indexing for MongoDB, and choosing FAISS over raw similarity comparisons to improve speed and scalability.

By identifying and planning for these risks, the project was able to adapt to unexpected challenges while still delivering a robust, functional system aligned with its original objectives.

1.5 System Overview

1.5.1 Application Purpose and Structure

BugTrackR is a role-based bug tracking and reporting application designed to streamline issue resolution across technical and non-technical users in a structured software environment. Unlike generic issue trackers, BugTrackR emphasizes role-specific interaction, guided submission forms, collaborative workflows, and intelligent automation. By providing tailored

functionalities (role-specific) and structured data handling (Within UI), the system helps development teams minimize redundancy, manage workloads efficiently, and ensure issues are resolved through clearly defined processes.

1.5.2 Key Stakeholders and Functional Access

The system is structured around five key roles, End Users (non-technical users), Developers, Testers, Team Leads, and Admins each with distinct responsibilities:

- End Users can submit bug reports through guided forms, add additional information, track the status of their submissions, receive email alerts, and chat with technical users for clarifications.
- Developers can self-assign bugs (if they are reporters), view bugs assigned or mentioned to them, update status, add/update/delete comments, chat with reporters, and request bug reallocation or reopening. They also access personalized analytics dashboards.
- Testers have similar access as developers but operate in later phases of the bug lifecycle. They handle assigned bugs, update status based on test outcomes, and raise reallocation or reopen requests if needed.
- Team Leads manage bugs within their assigned team. They can assign developers or testers, set priority, add/update/delete comments, chat with reporters, update status, review critical bug closure and reopen requests, and approve reallocation. They also access team-wide analytics to monitor workload and performance.
- Admins oversee the system at an organizational level, includes all features by all the above roles including assigning bugs, setting priority, adding comments across all applications, viewing analytics for all teams, and handling critical approvals.

This separation of access ensures that each stakeholder sees and interacts with features relevant to their role, which strengthens security and improves workflow clarity.

1.5.3 Workflow and Functional Overview

The workflow in BugTrackR begins when a bug is reported by a user (general or tech). The form adapts based on user type, with structured prompts for end users and advanced fields for technical users (e.g., stack trace, error logs, self-assignment).

As soon as a bug is submitted, the system performs duplicate detection using NLP-based similarity checking (both during submission and on-demand later). If a potential match is found, the user is notified, but they may still proceed to submit.

Once submitted, a priority is assigned manually based on the report content. The bug is then either auto-assigned (triggered) or manually assigned to a developer by the team lead or admin. Real-time chat between the reporter and the tech team is available during this stage, along with internal comments for developers/testers/teamleads/admins.

The bug then progresses through the following status phases:

"Open",
"Assigned",
"Fix In Progress",
"Fixed (Testing Pending)",
"Tester Assigned",
"Testing In Progress",
"Tested & Verified",
"Ready for Closure", (if critical bug)
"Closed",
"Duplicate" (if marked)

During this lifecycle:

- Reallocation Requests can be raised by developers/testers if they believe someone else is better suited to handle the bug. These are reviewed and approved by team leads or admins.
- Reopen Requests can be submitted if the bug reappears or wasn't fixed properly. Once approved, the bug moves back to the open phase and can be reassigned.
- For critical bugs, closure requires an extra oversight step by team leads or admins. Non-critical bugs can be closed directly by testers.
- Email alerts are sent for important actions such as status changes, assignments, and inactivity (e.g., no status update within a threshold time). Unassigned bugs also trigger alerts to admins.

Before assigning, team leads and admins can refer to analytics dashboards showing current workload, bug trends, and performance data to make better decisions.

The system supports advanced features like:

- Semantic and exact-match search
- Role-specific filters and sorting
- Marking bugs as favourites for quick access
- Admin assignment of teams to unassigned bugs
- Downloadable bug reports (PDF/CSV)

This structured workflow, built with flexibility and clarity in mind, ensures that bugs move efficiently through the resolution pipeline while providing full visibility and control to all stakeholders involved.

The below workflow diagram Figure 1, was created using Excalidraw [6] to give a general overview of the application flow.

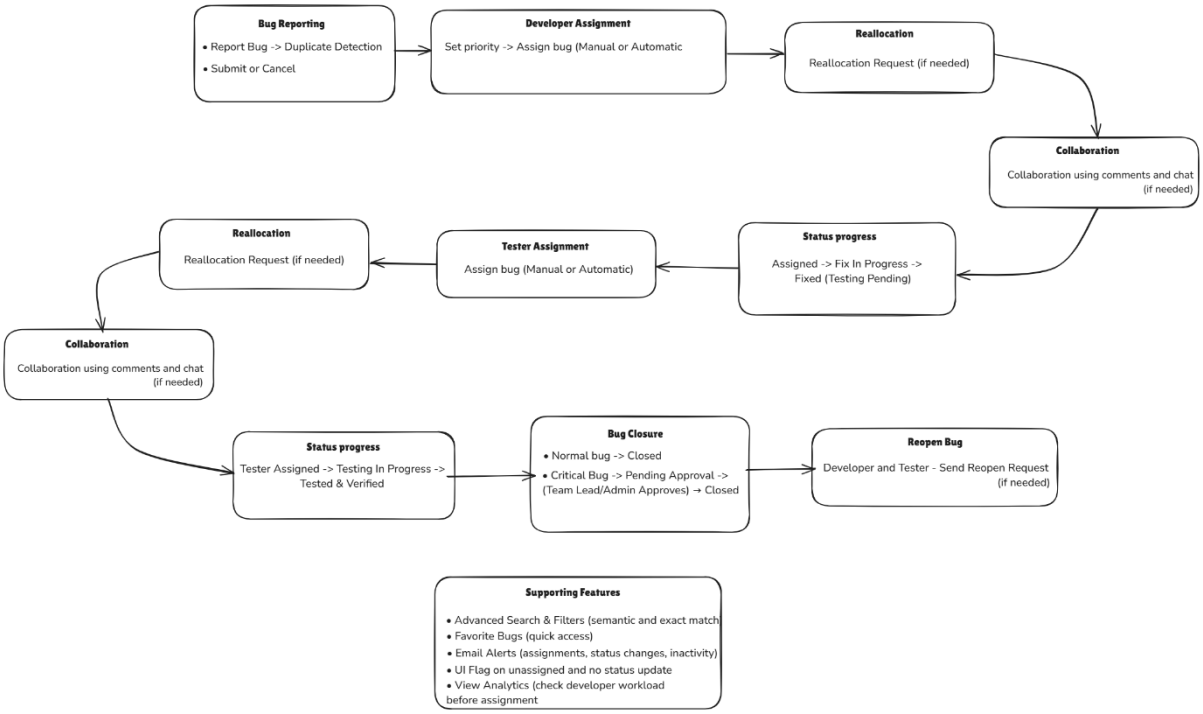

Figure 1:Workflow Diagram

2. Background Research

2.1. Technical Research and Technology Justification

The technologies used in BugTrackR were carefully chosen after comparing several popular options in terms of development time, performance, learning curve, flexibility, and how well they fit my project goals. Since this is a role-based system involving bug reporting, assignment, collaboration, and automation features like duplicate detection and priority classification, the chosen tech stack had to support flexibility, structured data handling, and fast development, while keeping things simple and maintainable.

Frontend: React, Material UI, and Tailwind CSS
For the frontend of BugTrackR, React was chosen after comparing it with other widely used frameworks such as Angular. Based on prior experience with Angular, React's simpler structure and smaller learning curve made it more manageable in an individual development context. Unlike Angular, which is a full-featured framework often used in enterprise environments, React offers greater flexibility in structuring the application without being constrained by boilerplate-heavy patterns, have to write and maintain a lot of extra setup code (modules, decorators, service injections) even before start writing actual feature, whereas React lets us directly start building features with minimal structure requirements.

Since the project involved building five different dashboards for distinct user roles (general users, developers, testers, team leads, and admins), the chosen frontend technology needed to be flexible, maintainable, and allow for rapid development of interactive features. React made it easy to create reusable components for features such as displaying header, sidebar, bug lists and details, viewing mentioned bugs, applying filters, and tracking bug statuses. These

components were reused across multiple dashboards with slight variations, which improved development speed and maintained consistency across the interface.

Another reason React was preferred was its virtual DOM, which helps improve performance by only updating parts of the interface that have actually changed. In BugTrackR, users frequently interact with the UI, such as updating bug statuses, toggling filters, or opening bug details, and the virtual DOM made these interactions feel smooth without reloading the whole page. According to Stoyan Stefanov in React Up and Running [7], this approach leads to faster render times and better user experience by not reloading the entire page, especially in apps that have dynamic and state-heavy interfaces. Similarly, Freeman in Pro React 16 [8] notes that React's component-based structure and virtual DOM make it ideal for interactive platforms like issue tracking systems.

Although Angular is a full-featured framework with built-in tools for routing and form handling, it also comes with more complexity and more setup time. It's better suited for large-scale enterprise projects with bigger teams, where that structure is necessary. For a solo developer building a full-stack system, Angular's boilerplate and strict patterns would have slowed development. React's wider adoption, active developer community, and strong ecosystem, including integration with tools like Material UI made it a more reliable and future-proof choice for this project. React's use of JSX (JavaScript + HTML syntax) also made UI logic easier to implement.

In terms of styling, Material UI (MUI) was used to implement common UI elements like steppers, tables, modals, and buttons. It helped speed up the development of consistent and accessible interfaces. However, for more custom layout control and spacing tweaks, Tailwind CSS was used alongside MUI. This combination gave both the structure of a design system and the flexibility of utility-first (predefined) CSS.

Backend: Node.js, Express.js, and MongoDB with Mongoose

For the backend of BugTrackR, Node.js was chosen as the main runtime environment along with Express.js to create the APIs. One of the biggest reasons for choosing Node.js was that it allowed the entire application to be built using JavaScript, both on the frontend and backend. This made the development smoother and faster because there was no need to switch between different languages. Since React was already used for the frontend, sticking with JavaScript across the whole project helped keep things consistent and easier to manage.

Express.js is a lightweight framework that works with Node.js and makes it simple to create routes for all the features in the system. This includes things like submitting bug reports, assigning bugs to developers and testers, updating their status, managing comments, and handling reallocation and reopen requests. The flexibility of Express.js also made it easier to organise logic cleanly, for example, by separating routes based on functionality or role and adding middleware functions that check permissions before certain actions are allowed. This helped build role-based access control, where only admins or team leads could assign bugs, and status updates can happen only if the bug is not closed or duplicate, for example.

Other options like Java with Spring Boot were also considered, since Java is known for its strong performance and good security. But Spring Boot requires a lot more setup and is harder to learn. Even simple tasks like reading or writing to the database require extra steps and more

code in Java. On the other hand, Node.js supports non-blocking operations, which is helpful for BugTrackR because multiple users can be using the system at the same time, like reporting bugs or updating their status, and everything still works quickly without slowing down.

For storing data, MongoDB was chosen because of how flexible it is. Bug reports in this project can look very different depending on who submits them. For example, general users just provide basic information, while developers or testers add more detailed things like error logs or stack traces. With a traditional relational database like MySQL or PostgreSQL, this would have needed multiple linked tables and complex queries. MongoDB's document-based structure made it easier to store all types of bug reports in the same collection, even if their fields were different., which made it a better choice for my application.

MongoDB also handles large volumes of data well, especially when it comes to querying and indexing. According to the official MongoDB documentation, it is built to scale horizontally across servers using a feature called sharding, which allows it to store and manage massive amounts of data efficiently [9]. This makes MongoDB well-suited for BugTrackR, which stores many bug reports across multiple applications, along with user comments, status changes, applications, users all of which can grow quickly as more users interact with the system.
Another option, SQLite, is good for small apps or offline use, but it isn't made for larger web apps like BugTrackR where many users are interacting with lots of data. MongoDB made it easier to search, sort, and organise large sets of bug reports.

To keep some structure and enforce rules on the database, Mongoose was used as a layer on top of MongoDB. Mongoose allows to define the structure (schema) of each collection, for example, making sure that every bug report has a title, description, and application etc., and that these fields are present and valid before being saved. It also supports strict data type validation, such as ensuring that numerical fields only accept numbers, and that string fields only accept string values. This helped prevent invalid or incomplete data from being stored in the system.

Mongoose also includes built-in methods that make it easier to interact with the database instead of writing complex queries. In addition, Mongoose provides middleware features which allows logic to run automatically. This was helpful for handling tasks like setting Bug IDs during bug report submission without writing the same logic in API route. Together, these features helped maintain a balance between MongoDB's flexibility and the need for structured, validated data in BugTrackR's backend.

Overall, using Node.js, Express.js, MongoDB, and Mongoose made it possible to build a backend that is fast, easy to manage, and flexible enough to handle different types of data and user actions. It worked well for the needs of BugTrackR, which required handling a variety of user roles and types of bug reports while still being easy to maintain and scale.

NLP & Duplicate Detection: Python, FastAPI, Sentence Transformers, and FAISS

For the duplicate detection feature in BugTrackR, Sentence Transformers, FAISS, and FastAPI were chosen. These technologies were selected after comparing several alternatives in terms of accuracy, scalability, and how well they handled natural language input, especially in real-world bug descriptions.

Sentence Transformers [10] was used to convert bug report titles and descriptions into dense vector embeddings that capture the context and meaning of the input. This helped the system detect duplicates even when users described the same issue in different ways. Traditional techniques like TF-IDF combined with cosine similarity were also considered, but they lacked contextual understanding and struggled with varied phrasing.

WordNet-based synonym matching was explored in the early stages as well. While it is designed to identify words with similar meanings, it often returned unrelated or out-of-context results that didn't make sense for software bugs. This caused inaccurate matches and made it unreliable for this use case.

To compare the embeddings efficiently, FAISS (Facebook AI Similarity Search) [11] was chosen instead of computing cosine similarity manually. FAISS is designed for fast and scalable similarity search and supports indexing strategies that help maintain performance as the dataset grows. This made it a better fit for BugTrackR, where the number of bug reports could increase rapidly.

FastAPI was selected over Flask to build the similarity detection microservice used in BugTrackR's duplicate bug report feature. Although Flask would have been easier to set up initially, FastAPI provided several advantages that were better suited for my project needs. Since the duplicate detection logic involves handling text data and returning similarity results quickly, FastAPI's asynchronous request handling allowed better performance when multiple bug reports were submitted at the same time. Its built-in Swagger UI documentation also made it much easier to test and integrate the microservice with my Node.js backend during development. Overall, choosing FastAPI helped ensure that the NLP service remained fast, reliable, and easy to maintain within the BugTrackR system.
Further details on how the approach evolved during development are provided in Section 5.3 (Adaptations Made During Implementation).

2.2 Review of Related Research

Research into key areas such as duplicate bug detection, automated triage/assignment, and user-centered design (UCD) has significantly influenced the planning and feature design of BugTrackR. These concepts directly address common challenges in bugs management, including issue duplication, triaging overhead, and usability concerns for technical users. These areas helped identify important challenges in bug tracking systems and informed the selection of appropriate technologies.

The way bug reports are written plays an important role in how quickly and effectively issues are resolved. A widely cited study [1] looked at feedback from developers and bug reporters across several open-source projects. It found that developers considered details like steps to reproduce, stack traces, and system environment information (such as browser or OS) to be especially helpful when trying to fix bugs. However, the study also showed that reporters often struggled to provide these types of information. For example, even though many developers found operating system details useful, fewer than 10% of bug reports actually included that information. This gap between what developer's need and what reporters tend to submit highlighted the importance of using structured and guided forms to help users create better reports. Based on this, BugTrackR's reporting forms were designed to adapt based on user role. General users are supported with helpful prompts to describe the issue clearly, while technical

users like developers and testers are provided with more direct fields for things like stack traces, logs, and assigning the bug to themselves.

One major area of research focuses on detecting duplicate bug reports, which is essential for minimizing redundancy and improving developer efficiency. According to [12], analyzing textual descriptions alongside execution logs can enhance the accuracy of identifying duplicates. Traditional keyword-based approaches are often too shallow to recognize reworded or paraphrased versions of the same issue. This has led to the adoption of more advanced techniques using natural language processing (NLP) and semantic similarity models, which are better suited for matching the meaning behind different descriptions. While BugTrackR does not currently use execution logs, this research reinforced the importance of using context-aware similarity techniques rather than relying on simple keyword matching. These insights influenced the decision to adopt Sentence Transformers for semantic similarity detection between bug reports

Another well-researched challenge is bug triage/assignment, where incoming issues must be routed to the appropriate developers or teams. Manual triage can be time-consuming, inconsistent, and prone to delays, especially in larger or more active systems. Studies such as [13] highlight the role of machine learning models in improving triage by analyzing historical bug data, descriptions, and developer expertise to recommend suitable assignees. This automation reduces the manual effort involved in assigning bugs while increasing the consistency and speed of resolution. Although BugTrackR does not implement ML-based triaging at this stage (uses algorithmic approach), this research helped guide initial design thinking, particularly the emphasis on considering developer workload and team specialization when planning auto assignment of developers/testers based on certain factors like seniority, expertise, priority and workload.

The importance of user-centered design (UCD) is also strongly emphasized in research related to software development tools. Research shows that tools built with structured, role-specific interfaces help improve navigation, reduce cognitive load, and enhance productivity for users like testers, developers, and leads [14]. Features such as role-based dashboards, advanced search, and clear workflows align with UCD principles and are shown to contribute to better user experiences in bug tracking systems. These findings directly influenced BugTrackR's interface design, which includes role-based dashboards, advanced search, and streamlined workflows for developers, testers, team leads, and admins.

Communication during bug fixing plays a critical role in how efficiently and successfully issues are resolved. A study by Ramírez-Mora et al. [15] analyzed over 500,000 comments from open source issue trackers and found that referential comments, those sharing objective, factual information were strongly associated with faster and more complete bug fixes. The study showed that not just the presence of comments, but the type and structure of communication, had a measurable impact on resolution time and success. These findings influenced the design of communication features in BugTrackR by highlighting the value of structured, in-system discussions. In response to this, BugTrackR includes a private internal comment section for technical users such as developers, testers, and leads to collaborate and make implementation-related decisions and a chat feature for reporters and the tech users to talk to each other.

2.3 Evaluation of Existing Tools

To gain a clearer understanding of the current landscape of bug tracking systems, two well-established platforms, JIRA [16] and Bugzilla [17] were reviewed. These tools are widely used in both enterprise and open-source projects and have played a significant role in shaping many of the foundational design choices in BugTrackR.

JIRA, developed by Atlassian, is a commercial bug and issue tracking platform that supports flexible workflows, customizable fields, and strong integration capabilities with tools like GitHub, Bitbucket, and Confluence. It offers advanced project tracking features such as dashboards, email notifications, and granular status transitions. Similarly, Bugzilla, an open-source tool by Mozilla, is known for its lightweight setup, detailed change tracking, and support for structured bug tracking workflows. Both platforms provide essential features like issue categorization (by type and priority), search and filtering, and status tracking through workflow states, which have been incorporated into BugTrackR as well.

However, during the evaluation of these tools, certain limitations were also identified, particularly in areas related to user experience, automation, and communication. Bugzilla, although reliable and well-structured, feels quite outdated in terms of user experience. Its interface is mostly form-heavy and looks more like a traditional database system, which can be a bit overwhelming, especially for users who aren't very technical. There aren’t many visual cues to guide users, and actions like updating a bug or checking its status often take several steps. For someone new to bug tracking, it’s easy to get lost or confused trying to figure out where to go or what to do next.

JIRA, although more modern and feature-rich, often requires significant setup and customization to access its more advanced features. Many automation tools, such as automated assignments or priority workflows, require additional configuration or paid plugins. This creates a steeper learning curve and may lead to inconsistencies in how different teams use the system.

BugTrackR addresses these challenges by introducing built-in features aimed at reducing manual workload and enhancing workflow automation. One such enhancement is the auto-assignment mechanism, which considers developer workload, seniority, expertise and priority when assigning bugs. This ensures that critical issues are assigned efficiently and fairly. Additionally, the system includes automated alerts and reminders for bugs that remain inactive, helping team leads and testers follow up on pending actions, a feature not natively emphasized in either JIRA or Bugzilla.

Another key area where BugTrackR introduces improvement is duplicate bug detection. Traditional systems often rely on manual identification or simple keyword-based search, which may not be effective when similar bugs are reported using different wording. BugTrackR uses context-aware semantic similarity detect potential duplicates based on meaning rather than exact matches, reducing redundancy and improving issue tracking accuracy.

Finally, communication within bug reports is enhanced in BugTrackR through a role-restricted internal comment section. While JIRA and Bugzilla allow for comments, they do not separate discussions between internal technical teams and bug reporters. BugTrackR supports private discussions among developers, testers, and team leads, improving focus and decision-making without cluttering user-facing communication.

In summary, evaluating JIRA and Bugzilla helped identify widely adopted features that were worth incorporating, such as structured workflows, issue tagging (to relevant teams and developer/tester), notifications, and dashboards. At the same time, it highlighted opportunities to improve in areas like automation, duplicate detection, and structured role-based communication. BugTrackR builds upon these insights to offer a more tailored, modern, and intelligent bug tracking experience.

2.4 Project Positioning in Context

The background research covered in the section provided valuable insights into existing tools, research findings, and technologies that directly influenced the development of BugTrackR. Established platforms like JIRA and Bugzilla offer strong foundations in bug tracking, with features like structured workflows, issue categorization, and status management. However, several areas for improvement were identified, including the lack of role-specific guidance, limited automation for bug assignment, and minimal support for structured, in-context communication between users and technical teams.

Academic literature further emphasized these challenges, particularly highlighting the importance of well-structured bug reports, effective comment communication, and user-centered design. Studies showed that incomplete or poorly formatted bug reports delay resolution, and that technical users benefit from different types of information than non-technical reporters are typically able to provide. These findings shaped many of BugTrackR's design decisions, such as implementing guided reporting based on user role, introducing contextual duplicate detection using semantic similarity techniques, and offering built-in assignment logic that considers team roles, developer workload, seniority and priority.

BugTrackR aims to bridge the gap between usability and technical depth by applying research-backed improvements to everyday bug tracking practices. While building on the core principles seen in existing systems, the project introduces a more modern, scalable, and collaborative workflow, positioning itself as a system that is not only technically functional but also thoughtfully designed for diverse user roles in real-world development environments.

3.Requirements

3.1 Project Requirements

Essential:

The essential requirements represent the foundational components without which the system would not be a functional system.

Login & Authentication

- Implemented login, OTP verification, password reset, and JWT-based authentication.
- Added proper validation for login and password reset functionalities.
- Enabled SSL for both frontend and backend.

Role-Based Dashboards

- Implemented role-based dashboards with below for each role
    - Users: Report bug, view and track reported bugs and settings.
    - Developers/Testers: Report bug, view reported and assigned bugs, update status, add comments, request reallocation, view analytics and settings.
    - Team Leads: Report bug, view reported and team assigned bugs, assign/reassign bugs to developer and tester, set priority, view team performance and settings.
    - Admins: Report bug, view reported and team assigned bugs across apps, assign/reassign bugs to developer and tester, set priority, view all teams' performance across apps, archive bugs and settings.

Bug Reporting

General Users:

- Developed API for submitting bug reports secured with authentication middleware with basic details (browser, device type, description).
- Implemented functional UI for bug reporting, integrated with the backend.
- Fields for general users with proper validation and limits - Title, description, application, issue type (dropdown with descriptive (help end user) categories of issue they are facing. Eg: Button or link is not working etc), Help reproduce issue (guided - what they were trying to do and what happens instead), browser (self-identified also have option to choose), OS (self-identified also have option to choose), attachments (upto 5 images or videos)

Technical Users:

- Secured API and UI allow developers, testers, team leads, or admins to submit detailed bug reports, including logs, error messages, and replication steps.
- Fields for tech users with proper validation and limits - Title, description, application, issue type (Direct issue types. eg: Frontend issue etc), Steps to reproduce issue, browser (self-identified also have option to choose, OS (self-identified also have option to choose), error logs, stack trace, attachments (upto 5 images, videos, txt, logs, pdf, word etc.,)

List Reported and Assigned bugs

- Developed API for listing all reported and assigned bugs.
- Implemented a UI (varies based on user roles) to display reported and assigned bugs.

Note: Added during development

Bug Status Tracking

- Users can view and track the status of bugs they have reported in a very user-friendly manner (Stepper component - logical flow of status with appropriate states highlighted based on the status)

Bug Status Updates

- Implemented APIs for updating bug status.
- Developed UI components for status updates integrated with backend APIs.
- Implemented drag-and-drop UI for developers and testers as they frequently update bug statuses. Dropdown for team leads and admin for update status.

Bug Discussion & Collaboration
- Implemented API and UI to add comments on assigned bug reports for collaboration.
- Implemented email notifications to mentioned users in the comment.

Additional Reporter Input
- Users can provide additional information after submitting a bug report.

Manual Bug Assignment
- Implemented APIs for manual bug assignment and reassignment by team leads and admins.
- Developed UI components for bug assignment integrated with backend APIs.

Priority Management
- Implemented manual priority setting and updating (Low, Medium, High, Critical) via APIs and UI.

Duplicate Detection
- Implemented manual marking of bugs as duplicates and undoing duplicate markings.

Email Notifications
Implemented email notifications for
- Password reset OTP via email
- Bug report submissions
- Bug assignments
- Status updates
- Critical bug closure request
- Comments mentioning users
- Request reallocation
- Inactive bug report (Scheduled)
- Marking duplicate automatically

Advanced Search & Filtering
- Implementing advanced search functionalities, including combination filters (eg: searching for bugs assigned to a specific developer with priority of high) and semantic searching.
- Adding sorting and filtering capabilities to bug lists.

Recommended:
The recommended requirements describe features that enhance functionality, efficiency, and user satisfaction, though the system would still be operational without them.

Automated Bug Assignment
- Implemented algorithm to consider developer/tester workload, seniority, expertise, and work hours to automatically allocate bugs.

Automatic Duplicate Detection

- Implemented automatic detection of similar bug reports using NLP-based techniques is implemented to assist users by suggesting potential duplicates during or after bug submission.

Automated Priority Classification

- Develop a model to categorize reported bugs based on their priority levels.

Bug Resolution Analytics

- Developed analytics for users, developers, testers, team leads, and admins to track performance and progress.

Bug Reallocation Request

- Added functionality for reassigning bugs when the assigned developer/tester cannot handle or work on them
- Implemented approval and rejection processes for team leads and admins.

In-App Notifications

- Work on real-time notifications within the application for bug status updates, assignments, and approvals. (Eg: When a developer is assigned a bug show in app notification)

Note: Dropped during development

Custom Bug Report Export

- Allowed users to export bug reports in PDF/CSV format with applied filters.

Time-Based Alerts for Bug Resolution

- Implemented automated alerts and UI warnings for inactive bugs and unassigned bugs, notifying relevant stakeholders (developers, testers, team leads and admin).

Optional:

The optional requirements are enhancements that, while not necessary to meet the core aims, further enhance the usability, flexibility, or integration capabilities of the system.

GitHub Integration

- Integrated GitHub for automatic issue creation and updates based on the reported bugs.

Note: Dropped as project progressed

Customization Options

- A user can personalize the application’s UI by selecting themes and fonts.

Note: Dropped as project progressed

Favourite Bugs

- Added the ability for users to mark bugs as favourites for quick access.

Unassigned Bug Queue (For Team Assignment)

- Implemented an unassigned bug queue where issues marked as "Other" can be manually allocated to the appropriate team (Frontend, Backend, or DevOps).

Note: Done for admin. Dropped implementation for team lead.

Communication between Bug Reporters and Tech Users

- Implemented a chat-like feature to improve communication between reporters and technical users.

Note: Added during development

Requirements/Enhancements added as project progressed.

List Mentioned bugs

- Developed API for listing all mentioned bugs.
- Implemented a UI to display mentioned bugs.

Note: Added during development

Bug Reopen Request

- Added functionality for reopening bugs when the assigned developer/tester wants to reopen.
- Implemented approval and rejection processes for team leads and admins.

Critical Bug Closure Workflow

- Developing a workflow for closing critical bug closure state and trigger follow-up emails if critical bugs are not closed or action taken (returned to testing or fixing) within a certain timeframe.

Role-Based Access Control (RBAC)

- Worked on improving RBAC by creating middleware that handles permissions more efficiently across endpoints.

Revision/Change History

- List of changes made to the bug report (assignment, status, priority, duplicate mark/undo). Helped with implementing other features like update status (to get latest and revert if requested) and undo duplicate etc.,

Toasts/Alerts Showing Updates or Changes

- Show toast notifications for updates or changes done in the application. Eg: Login successful, update status etc.,

Issue Fixes and Logical Improvements

- Worked on enhancing existing features, issue fixing and improving overall logic where necessary.

Changes to the original requirements and the reasons for their additions or removal have been explained clearly in Section 3.3 Changes to Requirements During Development.

3.2 System-Level Requirements

The user-facing functionalities (functional requirements) of BugTrackR have been detailed in Section 3.1. This section focuses on the key system-level qualities that ensure the platform remains secure, scalable, and user-friendly across all workflows. While functional capabilities

define what the system does, these requirements highlight how the system operates behind the scenes to deliver a smooth, efficient, and sustainable user experience.

3.2.1 Non-Functional Requirements

The non-functional requirements of BugTrackR focus on delivering a system that is secure, scalable, reliable, and user-friendly. Security is achieved through JWT-based authentication for session management, role-based access control enforced via backend middleware, encrypted communication over HTTPS, and secure password hashing using bcrypt. Performance and responsiveness are maintained through efficient API design using Express.js, modular backend architecture, and the use of WebSockets (Socket.io) and Node Cron for real-time chat and scheduled notifications respectively. Scalability is supported by choosing MongoDB as the primary database, enabling flexible data structures, implementing indexing for faster query performance, and using FAISS for efficient similarity searches in duplicate detection. Usability is enhanced through an intuitive and role-specific frontend built with React and Material UI. For example, providing dashboards tailored to each user role and guided bug reporting forms were designed to help users submit complete and structured reports, with different prompts shown based on whether the user is a general reporter or a technical user like a developer or tester. This ensures that users of all technical backgrounds can interact with the system efficiently and without confusion.

3.3 Changes to Requirements During Development

While the initial requirements for BugTrackR were carefully planned, several changes were made during the development phase based on practical observations, workflow testing and feedback. Some features were added to address gaps identified during real-world use case walkthroughs, while others were dropped or adjusted to better focus on core functionality within the available project timeframe. The following sections explain the changes made, along with the reasons behind each decision.

Additions and Adjustments:

List of Reported and Assigned Bugs: (Addition)
Although the initial plan included role-specific dashboards, the need for separate views for reported and assigned bugs became clear during development. Providing users with clear visibility into both the bugs they reported and those assigned to them enhanced traceability and supported better management of bug workflows.

Mentioned Bugs Listing: (Addition)
The ability to mention users in comments was implemented early, but during full workflow testing before the principal marker interview, it became apparent that mentioned users needed a way to access and interact with bugs where they were referenced. A separate view for "mentioned bugs" was introduced, allowing mentioned users to add comments without being able to modify core bug details, improving collaboration without compromising access control.

Reopen Bug Requests: (Addition)
While developing the reallocation request feature, the idea of supporting reopen requests emerged. Developers and testers sometimes needed to reopen a bug if they felt the issue was

not properly resolved. The reopen request workflow was added to allow technical users to flag bugs for re-evaluation, with approval managed by team leads and admins.

Chat between Reporters and Technical Users: (Replaced UI Customisation)
The initial plan included an internal comment system for technical users but did not account for structured communication between reporters and technical users. Generally, the reporters and developers/testers often needed to exchange clarifications through external channels like email, leading to fragmented communication. A dedicated chat feature was introduced within the bug report to enable direct and contextual communication, reducing dependency on external tools and improving traceability.

Critical Bug Closure: (Addition)
Critical bugs require additional oversight before closure. To avoid premature closure of critical issues, a critical bug closure workflow was implemented. Team leads and admins are required to review and close critical bugs, ensuring higher quality control. Notifications are also triggered if closure requests are not processed promptly, maintaining responsiveness.

Change History (Backend Only): (Addition)
Although a user interface for viewing change history was initially planned, only the backend logic for tracking changes such as assignment updates, priority changes, and duplicate markings was completed. This change history supported critical features like status revert operations and undoing duplicate markings. Due to time constraints and prioritization of core features, the UI display for change history was deferred as a future enhancement.

Dropped or Refined Features:

In-App Notifications: (Dropped)
While initially planned, real-time in-app notifications were dropped to focus development efforts on features that added greater value to communication and workflow clarity, such as the chat feature between reporters and technical users. Since email notifications already covered major events like assignments and status changes, in-app notifications were deemed redundant for the project's scope.

UI Customization (Themes and Fonts): (Dropped)
The customization feature allowing users to modify themes and fonts was dropped during development prioritization. Instead, the focus shifted to building more impactful collaboration features, such as real-time chat, which better supported the core objectives of BugTrackR.

Unassigned Bug Queue for Team Leads: (Modified – limited scope)
The initial plan included allowing team leads to allocate bugs marked as "Other" from the unassigned bug queue. However, during role access planning, it was decided that unassigned bug distribution should remain centralized under admin control to maintain consistency and avoid potential conflicts or duplication of effort across teams. Limiting this responsibility to admins ensures clearer ownership and better system-wide coordination. Enabling team leads to tag unassigned bugs to their teams can be considered as a future enhancement if scalability demands increase.

GitHub Integration:
The optional feature for GitHub integration was dropped to focus development time on strengthening core system workflows such as reallocation, reopen requests, communication etc., Given the time required to building a reliable two-way integration within the timeframe, this feature was dropped.

Ongoing
Bug Resolution Analytics:
Develop analytics for developers, testers, team leads, and admins to track performance and progress.

Priority Classification:
Development of an automatic priority classification feature based on bug description content is ongoing. The feature aims to assist in categorizing bugs into appropriate priority levels using machine learning techniques.

4. Technical Specification

Category	Technology & Version	Purpose
Frontend	React 18.2	JavaScript based library for frontend development
	JavaScript ES6	Modern JavaScript standard used for building dynamic and interactive web applications.
	HTML5	For structure and content of web applications
	CSS3	For styling web pages
	Tailwind CSS 3.4.17	CSS framework for quick and efficient UI development
	JSX (JavaScript XML)	Syntax extension for combining HTML and JavaScript in React components
	React Router DOM	Handles frontend routing and navigation between views
	MUI (Material UI) 6.4.4	Prebuilt React UI components with accessibility and styling support
	FontAwesome	Library for adding scalable vector icons in the UI
	Recharts	Library for building charts and graphs for analytics (e.g., bug fix efficiency trends)
	React Scripts	Provides scripts for running, building, and testing React apps (via Create React App)
	jspdf	Creates PDF documents in frontend
	jspdf-autotable	Generates formatted tables inside PDFs
	file-saver	Allows saving files from browser (e.g., exported bug reports)

	react-chat-ui-kit	Provides prebuilt chat UI components for real-time communication
	react-dnd	Enables drag-and-drop functionality (e.g., bug status management)
	react-dropzone	Allows drag-and-drop file uploads (e.g., bug screenshots, logs)
Backend	Node.js 22.11.0	Runtime environment to run JavaScript in server
	Express.js 4.21.2	Framework for creating and managing RESTful APIs
	JavaScript ES6	Modern JavaScript standard used for building dynamic and interactive web applications.
	Multer	Middleware for handling file uploads (e.g., screenshots, logs)
	JWT (JSON Web Token)	Authentication and authorization using tokens
	Bcrypt	Hashing and verifying user passwords securely
	Cors	Middleware for handling Cross-Origin Resource Sharing requests
	Nodemailer	Sends email notifications (e.g., bug updates, password reset)
Database & ORM	MongoDB 8.0.4	NoSQL database for handling flexible, document-based data storage
	Mongoose	Object Data Modeling library for MongoDB to manage data efficiently
	MongoDB Compass 1.46.1	GUI for viewing, querying, and debugging MongoDB documents and collections
Authentication & Security	JWT (JSON Web Token)	For secure authentication and user authorization
	SSL	Used for secure data communication between client and server
Microservice (NLP)	Python	Programming language used for the NLP-based duplicate detection microservice
	FastAPI	Python framework used to quickly build and serve APIs
Version Control	GitLab	Platform for version control
Real-time Features	Socket.io	Enables real-time features (e.g., chat system, live status updates)
IDE	Visual Studio Code (VSCode) 1.99.3	Code editor
	Sourcetree	GUI tool for simplifying Git operations
API Validation	Postman	API testing tool
Testing Frameworks	Jest & React Testing Library	Frameworks for unit and integration testing in React.
NLP Libraries	Sentence Transformers	Converts text into vector embeddings for semantic similarity

	FAISS	Library for fast similarity search used for efficient duplicate bug detection.
Task Scheduling	Node Cron	Schedules and automates background tasks at regular intervals.
Operating System	Windows 11	Operating System used during development
Planning & Documentation	OneNote	Used to take notes during meetings, track issues, and document feature ideas
	Microsoft Word	Used for drafting and formatting project reports
Other Utilities	Day.js	Lightweight JavaScript library for date and time formatting
	dotenv	Loads environment variables from .env files

Note: Priority classification related technologies (ML) will be added after the feature is completed.

4.1 Languages and Frameworks

Frontend - React, JSX, HTML5, CSS3
BugTrackR's frontend was developed using React for building modular and reusable components tailored for role-based dashboards and workflows. JSX combined logic and layout efficiently, while HTML5 and CSS3 defined structure and styling. Tailwind CSS was integrated alongside Material UI to create responsive, consistent designs with minimal manual CSS.

Backend - Node.js and Express.js
The backend was built using Node.js and Express.js, enabling a JavaScript-based full-stack system. Express facilitated clear route management, middleware handling, and role-based access control, while Node.js' non-blocking model ensured smooth concurrent user operations.

NLP Microservice - Python and FastAPI
A separate Python-based FastAPI microservice was developed for duplicate bug detection using NLP. FastAPI provided fast asynchronous request handling and auto-generated API documentation for easy integration with the Node.js backend.

4.2 Libraries and Packages

A range of libraries and packages supported BugTrackR's features:

Frontend: Material UI (UI components), Tailwind CSS (utility-first (predefined classes) styling), React Router DOM (routing), React DnD (drag-and-drop for bug status updates), chat-ui-kit-react (chat interface), React Toastify (notifications), Axios (HTTP requests), Day.js (date formatting).

Backend: Mongoose (MongoDB data modeling), JWT (authentication), bcrypt (password security), Multer (file uploads), Nodemailer (email notifications), Socket.io (real-time communication), Node-cron (scheduled tasks).

NLP Microservice: Sentence Transformers (text embeddings) and FAISS (fast similarity search) for scalable duplicate detection.

4.3 Database

MongoDB was used as the primary database to flexibly store varied bug reports without rigid schema constraints. Collections such as BugReports, Users, Comments, and Applications were structured using Mongoose, which enforced field validations and supported features like timestamps and ID generation. This document-based approach aligned with BugTrackR's dynamic and role-based workflows and ensured scalability as data volumes grew.

4.4 Tools

Development and project management tools included:

- **Coding:** Visual Studio Code (IDE with integrated terminal and Git support)
- **Version Control:** GitLab (repository management), Sourcetree (visual Git client)
- **API Testing:** Postman (backend API validation)
- **Planning and Tracking:** Instagantt [18] (Gantt chart planning), Microsoft OneNote (meeting notes and task tracking)
- **Database Inspection:** MongoDB Compass (GUI for querying collections)

5. Development Methodology and Project Execution

5.1 Overview of Development Methodology

The development of BugTrackR followed a **Waterfall**-based approach with **iterative refinements**. The Waterfall model is a step-by-step development method where each stage is usually completed before moving to the next one. For this project, the main phases were planning the background research, requirements, designing the system, building the features, and finally testing and writing the report.

Although the plan was based on Waterfall, there were a few changes made along the way when needed. For example, some features like the chat between users and tech roles, or the reopen request workflow, were added later after getting ideas during development. Also, the order of some tasks was adjusted depending on how much time was available or which parts were to be built first. These were small changes that helped keep the project on track without completely changing the overall structure.

This method worked well for BugTrackR because the key features and goals were already planned before starting the actual development. Since this was an individual project with a clear scope, the structured step-by-step nature of Waterfall helped in maintaining focus and progress across each stage.

5.2 Why Waterfall Was Chosen

The Waterfall model was chosen for this project because it matched both the nature of the development and the structure of the university's assessment schedule. Waterfall follows a stage-by-stage approach, where each phase, such as requirements gathering, design, development, and testing is completed before the next begins. This worked well for BugTrackR because the overall goals and key features were clear from the start.

In contrast to Agile methodologies, which focus on flexibility, short sprints, and constant feedback, Waterfall allowed for more structured planning and long-term decision-making. Agile is commonly used in team-based environments, where practices like daily standups, iteration planning every one to two weeks, backlog refinement sessions, and retrospectives are carried out to support continuous improvement. Since this project was developed individually and had no external client, these collaborative Agile practices were not relevant. There was no need for team coordination, which made Waterfall a more practical and manageable choice.

Similarly, Scrum, which is a widely used Agile framework, defines specific team roles such as the Scrum Master, who ensures the process runs smoothly and removes obstacles, and the Product Owner, who manages the product backlog and represents stakeholder interests. Scrum teams also include developers who work together within fixed-length sprints to deliver small increments of the product. The framework relies heavily on structured meetings and continuous team collaboration, which were not applicable in this project setting. However, feedback was still received regularly from the project supervisor, usually every two weeks, and adjustments were made based on that input. While this provided some flexibility, it was not part of a formal Agile sprint structure. For these reasons, Scrum was not chosen.

Although the Iterative Waterfall Model allows formal feedback and revision between phases, this was not required for BugTrackR. The original Planning and Design stages were thorough enough that only minor refinements, such as adding chat functionality and reopen request workflows, were made during development. These changes did not require formal revisiting or reworking/restructuring of the earlier phases. Therefore, a structured Waterfall approach with minor iterative refinements was more appropriate for the project.

The university's academic structure further supported the use of Waterfall. The set deadlines for the preliminary, interim, and final reports aligned closely with the distinct phases of the Waterfall model. For example, the preliminary report focused on requirement planning and research, the interim report captured design choices and implementation progress and the final report involved implementation, testing, analysis, and reflection. This made it easier to divide the project into manageable parts and track progress based on clear academic expectations.

Although Waterfall is sometimes seen as less flexible, it was still possible to make smaller adjustments during development when needed, without changing the core structure. This balance between planning and adaptability is explained in the next section (5.3).

5.3 Adaptations Made During Implementation

Although the development of BugTrackR was planned using a Waterfall model with structured phases, several changes were made during implementation based on ongoing feedback, feasibility checks, and practical observations. These changes were not large enough to shift the methodology, but they helped improve the system's usefulness and ensured that the final outcome was more complete and aligned with real-world needs.

One major change was the addition of a chat-like communication feature between bug reporters and technical users. This was not planned in the initial requirements, but it became necessary during implementation. While working on the internal comments section (which allows developers, testers, and team leads to collaborate), it became clear that there was no dedicated way for the technical team to ask follow-up questions to the reporter if additional information was needed. Since the internal comments were only visible to technical users, a separate communication channel was designed where both parties could chat within the bug report itself. This allowed for more efficient and contextual exchanges, especially in situations where information was incomplete or unclear.

Another feature added during implementation was the reopen request workflow. This was included to support cases where a developer or tester believed that a bug had not been properly resolved or had resurfaced. Originally, once a bug was marked as fixed and closed, it would stay in that state. However, in practice, there were cases where reopening a bug could help ensure correctness and accountability. This feature was introduced to allow reopening requests with proper justification, with approval managed by team leads or admins.

The order of feature development was also adjusted compared to what was originally planned in the preliminary report. Core features such as bug listing, filtering, and assigned bug views were prioritized earlier than expected because they were required across all user roles. More advanced features, including automated duplicate detection and analytics, were shifted to a later phase, after the basic functionality of the system had been completed and tested. This change was reflected in the interim report.

There were also refinements in the duplicate detection and search logic. Initially, the plan was to implement synonym-based detection using WordNet and the Lesk algorithm, both of which were explored and partially implemented. However, WordNet provided overly rigid or unrelated synonym suggestions that did not match the context of bug tracking. For example, the word "crash" could return meanings like "car crash" or "stock market crash", which were not relevant in the context of a software bug. Similarly, the Lesk algorithm, which works by comparing dictionary definitions (known as glosses) to find overlapping words, did not perform well in understanding the actual meaning of bug descriptions. Since it relies on exact matches in word definitions, it struggled with the kind of natural, varied language used in real-world bug reports. This limitation in handling word sense disambiguation (figuring out which meaning of a word is being used in a given context) made it unsuitable for the system.

Based on these findings, and also feedback from the principal marker during the interview, a more advanced and scalable solution was adopted. The system was updated to use Sentence Transformers, a library that generates context-aware vector embeddings for bug descriptions, and FAISS (Facebook AI Similarity Search), which allows for fast nearest-neighbour searches over those embeddings. This approach provided better accuracy for identifying similar bugs, and FAISS ensured that performance remained high even as the number of bug reports grew. The principal marker specifically raised questions about the scalability of the system, which

led to the decision to use FAISS instead of basic cosine similarity methods, which are slower and less efficient at scale.

In-app notifications, which was originally included in the preliminary report as a recommended feature, was later dropped. After evaluating priorities and available time, it was decided to focus on more impactful features like communication, assignment workflows, and analytics. Email notifications were already in place for key actions, and adding in-app notifications would have added extra complexity with limited additional benefit.

In addition to these major changes, several smaller but meaningful adjustments were also made during the project. One example is the introduction of backend-level change tracking for assignments, priority updates, and duplicate markings. While a frontend interface to display this history was not implemented due to time constraints, the backend logic was completed to support future improvements.

The advanced search and filtering system also evolved during development. Originally, only combination filtering was planned, but the feature was expanded to support contextual search mode. The synonym-based search, which was explored using WordNet and Lesk, was also dropped due to poor relevance and performance, and replaced with a more robust embedding-based system like in duplicate detection.

Minor UI/UX refinements were also made based on user testing and feedback. These included adding helper texts for form fields, better validation messages, and restrictions on comment editing/deleting after submission. Lastly, although the role-based access control (RBAC) system was part of the early plan, it was significantly improved during development with middleware-based enforcement in API routes in addition to conditional checks, ensuring better security and permission handling for technical roles.

In summary, although the project was guided by a structured development plan, these adaptations allowed it to stay relevant and practical. The changes were made thoughtfully, based on technical results, feedback from the supervisor and principal marker, and a clear focus on improving the user experience across all roles

6. Design

6.1 Architecture Design

In this section, I will present the overall system architecture of BugTrackR, showing how the frontend, backend, microservice (Python), and database components interact with each other to form the complete application.

6.1.1 Client-Server Architecture

Here, I will explain how BugTrackR follows a client-server architecture. The React frontend acts as the client, sending HTTP requests to the Node.js/Express backend server. The backend processes these requests, interacts with the MongoDB database and/or the Python-based microservice when needed, and sends responses back to the client.

6.2 Database Design

In this section, I will describe the database design for BugTrackR, including the collections used (such as applications, bugreports, users, comments, etc.,). I will also provide a logical database diagram and explain the purpose of each major collection.

6.3 Use Case Diagram

In this section, I will present the use case diagram that shows how different users (such as users, developers, testers, leads, and admins) interact with the system.

6.4 System Module Breakdown

Here, I will break down BugTrackR into its major modules, including Bug Management, Email Notifications, Analytics etc., I will briefly explain the purpose and functionality of each module.

6.5 Logo and Branding

The BugTrackR logo, Figure 2, was designed using Figma [19], a widely used design tool for creating digital graphics and UI assets. The logo visually represents the core ideas of the application, including structured bug reporting, organized bug management, status tracking, and maintaining systematic bug records.

Figure 2: BugTrackR Logo

The primary colour scheme for BugTrackR is based on #102B4E, a deep blue tone selected to convey professionalism, that is crucial for a defect management platform. This colour is consistently applied across the application's headers, navigation bars, buttons, and key interface elements to ensure visual coherence.

For typography, BugTrackR uses a clean and modern font family comprising Inter, Roboto, Arial, and sans-serif. This font stack was chosen to maintain readability and a professional appearance.

7. Implementation

7.1 Frontend Implementation

Here, I will explain the UI implementation role-based dashboards for different user types, dynamic bug reporting forms, advanced filtering and sorting functionalities, and status tracking interfaces with drag-and-drop and dropdown-based updates, chat and more with code and screenshots.

7.2 Backend Implementation

Here, I will explain the backend implementation (API + DB) for the features as above and more with code and screenshots.

7.3 Python API for NLP

In this section, I will explain the automatic duplicate detection feature implementation details along with code and screenshots.

7.4 Notable Code Snippets

Here in this section I will add 3 code snippets written by myself along with screenshots for implementing complex features.

7.5 Running the Application

In this section, I will explain the steps required to start the application.

8. Testing and Evaluation

8.1 Requirements Evaluation Plan

Here, I will explain the evaluation plan followed for BugTrackR.

8.2 Unit Testing

In this section, I will describe any unit testing performed on key backend functions or frontend components, along with screenshots.

8.3 Functional Testing

Here, I will explain the functional testing performed to validate key system features such as bug reporting, assignment flows, status tracking, role-based access restrictions etx.,

8.4 Integration Testing

In this section, I will explain how frontend and backend interactions were tested, such as form submissions, API responses, data consistency checks, and real-time event handling between systems.

8.5 Limitations and Edge Cases

Here, I will describe known limitations of the system, edge cases not fully handled, and areas where there are known gaps.

8.6 Supervisor/User Feedback

In this section, I will summarize any feedback received from my supervisor and/or principal marker during project meetings or demos, highlighting suggestions, improvements, and positive remarks.

8.7 Testcases

In this section, I will add the list of test cases written.

9. Critical Reflection

9.1 What Worked and What Didn't

Here, I will reflect on the features and processes that worked successfully during the development of BugTrackR, and also discuss the requirements that did not fully meet initial expectations, dropped or had limitations.

9.2 Decisions Made and Their Impacts

In this section, I will explain the key decisions taken throughout the project, and reflect on how these choices influenced the development process, system performance, and overall project outcome.

9.3 Challenges Overcome and Lessons Learned

Here, I will describe the main challenges encountered during the project, such as technical difficulties or time constraints, and the lessons learned from overcoming these issues.

9.4 Recommendations for Future Work

Here, I will suggest possible future improvements for BugTrackR.

10. Conclusion

Here, I will summarise the overall contributions made through BugTrackR, briefly compare it with other existing tools, and reflect on the final outcomes and key takeaways.

11. References

[1] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A. and Weiss, C. What Makes a Good Bug Report? IEEE Transactions on Software Engineering, 36(5), pp.618–643. https://doi.org/10.1109/tse.2010.63

[2] Bhattacharya, P., & Neamtiu, I. "Fine-Grained Incremental Learning and Multi-Feature Tossing Graphs to Improve Bug Triaging." Proceedings of the IEEE International Conference on Software Maintenance (ICSM), 1–10. https://dl.acm.org/doi/10.1109/ICSM.2010.5609736

[3] Bettenburg, N & Premraj, R. & Zimmermann, T & Kim, S. Duplicate Bug Reports Considered Harmful ... Really? IEEE International Conference on Software Maintenance, ICSM.337-345.10.1109/ICSM.2008.4658082. https://www.researchgate.net/publication/224343297_Duplicate_Bug_Reports_Considered_Harmful_Really

[4] Hooimeijer, P., & Weimer, W. "Modeling Bug Report Quality." Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE), 34-43. https://dl.acm.org/doi/10.1145/1321631.1321639

[5] Mockus, A., & Herbsleb, J. D. "Expertise Browser: A Quantitative Approach to Identifying Expertise." Proceedings of the 24th International Conference on Software Engineering(ICSE),503-512. https://www.researchgate.net/publication/2543802_Expertise_Browser_A_Quantitative_Approach_to_Identifying_Expertise

[6] Excalidraw. "Excalidraw.", https://excalidraw.com/

[7] Stefanov, S. (2016). React Up and Running: Building Web Applications. O'Reilly Media. https://dl.ebooksworld.ir/books/React.Up.and.Running.2nd.Edition.Stoyan.Stefanov.OReilly.9781492051466.EBooksWorld.ir.pdf

[8] Freeman, A. (2018). Pro React 16. Apress. https://dl.ebooksworld.ir/motoman/Pro.React.16.Adam.Freeman.Apress.www.EBooksWorld.ir.pdf

[9] MongoDB Docs - Horizontal Scalability and Performance Features, https://www.mongodb.com/docs/manual/sharding/

[10] "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Hugging Face. [Online]. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[11] "Faiss/README.md at Main · Facebookresearch/Faiss." GitHub, 2017, https://github.com/facebookresearch/faiss/blob/main/README.md

[12] Wang, Xiaoyin, et al. "An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information." Proceedings of the 13th International Conference on SoftwareEngineering-ICSE'08,2008, https://www.researchgate.net/publication/221554559_An_approach_to_detecting_duplicate_bug_reports_using_natural_language_and_execution_information

[13] Automating Bug Triage: Unleashing the Power of Machine Learning and Natural Language Processing. https://www.researchgate.net/publication/382455930_Automating_Bug_Triage_Unleashing_the_Power_of_Machine_Learning_and_Natural_Language_Processing

[14] Kualitee. User-Centric Bug Resolution: Best Practices with Bug Tracking Software. https://www.kualitee.com/blog/bug-management/user-centric-bugresolution-best-practices-with-bug-tracking-software/

[15] Ramírez-Mora, Sandra L., et al. "Exploring the Communication Functions of Comments during Bug Fixing in Open Source Software Projects." Information and Software Technology, vol. 136, Aug. 2021, p. 106584, https://www.sciencedirect.com/science/article/pii/S0950584921000665

[16] "Jira Documentation. Atlassian Support. Atlassian Documentation." https://confluence.atlassian.com/jira

[17] "Features." Bugzilla, 2025, https://www.bugzilla.org/about/features.html

[18] "Instagantt. Online Gantt Chart Software. Asana Integration." https://app.instagantt.com/

[19] "Figma: The Collaborative Interface Design Tool." Figma, 2024, www.figma.com/.