# BUG TRACKING SYSTEM

MINOR PROJECT REPORT

By

**PRATIP DAS RA2311056010188**

**ASUTOSH RANJAN RA2311056010190**

Under the guidance of   
**Venkatesh S.***In partial fulfilment for the Course*

of

**21CSC206P – ADVANCED OBJECT ORIENTED AND PROGRAMMING**

In

**Data Science and Business System**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2024**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that this minor project report for the course **21CSC206P** **ADVANCED OBJECT ORIENTED AND PROGRAMMING** entitled in "**Bug Tracking System**" is the Bonafide work of **Pratip Das (RA2311056010188)** and **Asutosh Ranjan (RA2311056010190)** who carried out the work under my supervision.

# SIGNATURE SIGNATURE

Dr. Venkatesh S. Dr. V. Kavitha

Assistant Professor Professor & Head

**Data Science and Business System Data Science and Business System**

SRM Institute of Science and Technology SRM Institute of Science and

Kattankulathur Technology, Kattankulathur

# ABSTRACT

A Bug Tracking System is a software solution designed to streamline the identification, documentation, tracking, and resolution of software bugs throughout the development lifecycle. This system enables a structured and collaborative approach by bringing together four primary roles: Tester, Developer, Project Manager, and Admin. Testers report bugs with detailed information, allowing developers to diagnose and address issues efficiently. Project Managers oversee bug prioritization, ensuring alignment with project goals and timelines, while Admins maintain system integrity and manage user roles. The Bug Tracking System enhances productivity by providing real-time updates, automated notifications, and comprehensive reporting features, which improve both individual accountability and team collaboration. Ultimately, this platform facilitates quicker bug resolution, higher software quality, and more effective resource management, making it an indispensable tool in agile and traditional software development environments.

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. Muthamizhchelvan**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Lakshmi M, Professor & Head, Department of Networking and Communication** for her constant encouragement and support.

We are highly thankful to my Course project Faculty **Venkatesh S., Assistant Professor, Data Science and Business System,** for his/herassistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. V. Kavitha, Professor, Department of Data Science and Business Systems** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **CONTENTS** | **PAGE NO** |
| **1** | **INTRODUCTION** | **1** |
|  | 1.1 Motivation | **2** |
|  | 1.2 Objective | **3-4** |
|  | 1.3 Problem Statement | **5** |
|  | 1.4 Challenges | **6-7** |
| **2** | **LITERATURE SURVEY** | **8-10** |
| **3** | **REQUIREMENT ANALYSIS** | **11-14** |
| **4** | **ARCHITECTURE & DESIGN** | **15-16** |
| **5** | **IMPLEMENTATION** | **17-19** |
| **6** | **EXPERIMENT RESULTS & ANALYSIS** | **20-21** |
| **7** | **CONCLUSION** | **22** |
| **8** | **REFERENCES** | **23** |

1. **INTRODUCTION**

A **Bug Tracking System** is a critical application in the software development process, designed to streamline the reporting, tracking, and resolution of bugs or issues encountered in software. As projects grow in complexity, maintaining a reliable method for tracking bugs becomes essential to ensure software quality and maintain efficient development workflows.

This application enables developers, testers, project managers, and other stakeholders to effectively document bugs, assign priorities, and track their resolution progress over time. By providing a centralized platform to log issues with essential details such as bug description, severity, priority, and steps to reproduce, a bug tracking system ensures that no issue is overlooked and facilitates transparent communication within the development team.

Built using Java, this application offers a cross-platform solution that can be deployed across various operating systems, supporting scalability and adaptability to meet the needs of small teams or large-scale projects. A well-designed bug tracking system enhances productivity, improves accountability, and ultimately contributes to delivering high-quality software by allowing teams to identify and resolve bugs promptly. Additionally, it can integrate with version control systems, CI/CD pipelines, and other project management tools to support a seamless and organized development environment.

* 1. **Motivation**

What drives the creation of and implementation of a Bug Tracking System is its large ability to improve the efficiency, accuracy, and transparency of the software development lifecycle. Since bugs are an inevitable condition of any software project, inappropriate handling causes unavoidable delays, low-quality software, or even the collapse of the project because it has to be handled in a disorganized way. A systematic approach to tracking bugs is therefore crucial for every software team in finishing reliable, high-quality products on time.

A bug tracking system is a structured method to report, categorize, assign, and resolve bugs in an efficient manner. It is a comprehensive tool that helps developers and testers prioritize issues based on their criticality, as well as kind of impact. The resource allocation becomes focused and efficient. Effective communication, accountability, and the scope for bug trends by way of analytics and reporting improve over time.

A Bug Tracking System connects multiple teams or departments into a cohesive unit while still operating in fast-paced, collaborative environments. Being the 'bridge,' it enables constant tracking of progress and encourages collaboration. It becomes an invaluable resource for stakeholders and project managers in maintaining transparency and delivering frequent status updates on the health of a project.

So, the Bug Tracking System in Java, boasting platform suitability and fantastic performance, can be aptly scalable as well as an efficient solution both for the smaller teams and large enterprises as for following up on the bugs in a streamlined, well-organized manner to support high-quality delivery of software.

* 1. **Objective**

A bug tracking system aims to streamline the process of identifying, reporting, and resolving software defects. Here are the primary objectives:

1. **Efficient Defect Management:**

**Centralized Repository:** Provide a centralized platform to store and track all reported bugs.

**Clear Prioritization:** Allow for easy categorization and prioritization of bugs based on severity and impact.

**Detailed Tracking:** Enable detailed tracking of each bug's lifecycle, from initial report to resolution and closure.

1. **Improved Communication and Collaboration:**

**Effective Communication:** Facilitate seamless communication between developers, testers, and other stakeholders involved in the bug-fixing process.

**Collaboration:** Promote collaboration among team members by providing a shared workspace for discussing and resolving issues.

1. **Enhanced Productivity:**

**Time-Saving:** Streamline the bug-reporting and resolution process, reducing wasted time and effort.

**Increased Efficiency:** Improve overall development efficiency by identifying and fixing bugs promptly.

1. **Data Analysis and Reporting:**

**Data Collection:** Gather and analyze data related to bug trends, severity, and resolution times.

**Reporting:** Generate comprehensive reports to help teams identify areas for improvement and make data-driven decisions.

1. **Quality Assurance:**

**Quality Improvement:** Contribute to the overall quality of the software product by ensuring that defects are addressed in a timely and effective manner.

**Compliance:** Help organizations comply with industry standards and regulations related to software quality.

1. **Scalability and Flexibility:**

**Scalability:** Accommodate the growth of the software project and the team over time.

**Flexibility:** Adapt to changing requirements and workflows.

* 1. **Problem Statement**

In any software development process, identifying, tracking, and resolving bugs is critical for delivering a high-quality product. However, traditional bug reporting and tracking methods often face several challenges, such as poor communication between team members, inefficient tracking of bug statuses, and lack of accountability, leading to delays in the software development lifecycle. Teams may encounter additional challenges like redundant bug reports, lack of prioritization, and ineffective tracking of recurring issues, which can impact the product's reliability and user satisfaction.

To design a centralized, user-friendly, and efficient Bug Tracking System that addresses these challenges by enabling:

**Efficient Bug Reporting**: Simplified bug submission with necessary details (description, screenshots, priority level, module affected) to reduce ambiguity.

**Real-Time Collaboration and Notifications**: Streamlined communication between developers, testers, and project managers with notifications on status updates and changes.

**Prioritization and Assignment**: Ability to prioritize bugs based on severity and assign them to the appropriate team members with defined timelines.

**Tracking and Analysis**: Detailed tracking of bug status, recurrence, and historical data for insights on common problem areas.

**Integration with Development Tools**: Seamless integration with version control systems and project management tools to ensure updates across platforms.

The Bug Tracking System should ultimately reduce the time and effort spent on bug resolution, improve transparency and accountability, and support the delivery of a robust, high-quality product.

* 1. **Challenges**

Some of the key challenges in the development phase of Bug Tracking System include:

**Data Accuracy and Completeness:** Ensuring that bug reports are detailed, accurate, and complete can be difficult, especially when users or testers omit crucial information. Incomplete data hampers the efficiency of debugging and may require additional back-and-forth communication.

**User Interface Design:** Creating an intuitive and user-friendly interface is challenging, as the system must accommodate users with varying levels of technical expertise. A complex interface may lead to user errors, incomplete bug reports, or low adoption rates.

**Efficient Search and Retrieval:** As the database grows with numerous bug reports, it becomes challenging to maintain fast, efficient searching and sorting functionalities. Ensuring users can retrieve relevant bug information quickly without redundant entries is essential.

**Prioritization and Severity Assessment:** Accurately assessing the priority and severity of each bug can be complex, as this often depends on subjective criteria. Misprioritization can lead to critical bugs being overlooked or minor bugs consuming excessive resources.

**Scalability:** The system must be able to handle an increasing volume of bugs as projects grow. Designing the system to scale smoothly while maintaining performance and response time can be technically challenging.

**Tracking and Reporting Historical Data:** Providing meaningful analytics, such as trends in bug recurrence, time-to-fix, or root causes, is valuable but challenging, requiring robust data management and analysis features.

**Integration with Other Tools:** Integrating the bug tracking system with existing development tools (e.g., version control, CI/CD pipelines, project management software) can be technically complex and requires compatibility with various software environments.

**Security and Data Privacy:** Bug reports may contain sensitive information about the software, exposing vulnerabilities. Ensuring that the system is secure and adheres to data privacy standards is critical, especially when sensitive or user data is involved.

**Automating Bug Detection and Assignment:** Automating parts of the bug reporting and assignment process (e.g., suggesting likely responsible developers) can be beneficial but technically challenging, often requiring advanced algorithms or machine learning techniques.

**Notification Management:** Designing an effective notification system that informs relevant team members without overwhelming them is challenging. Frequent, irrelevant notifications can lead to notification fatigue, causing users to ignore important updates.

These challenges must be addressed to create a reliable, efficient, and user-friendly bug tracking system that meets the needs of software development teams.

1. **LITERATURE SURVEY**

Bug Tracking Systems (BTS) are essential tools in software development, enabling teams to log, prioritize, and manage defects efficiently. Over the years, research and advancements have focused on improving these systems’ effectiveness, usability, integration capabilities, and overall impact on the software development lifecycle. This survey explores existing studies and research on the evolution, challenges, and innovations in Bug Tracking Systems.

**Historical Evolution of Bug Tracking Systems:** Early studies on software maintenance and quality control emphasized the need for structured tracking systems to manage bugs, defects, and user-reported issues. According to R. Chillarege et al. (1992), a structured defect tracking approach enables development teams to track trends, analyze root causes, and continuously improve software quality. Traditional BTS were limited to manual entry and tracking but laid the groundwork for modern, automated BTS with sophisticated features.

**Features and Functionality of Bug Tracking Systems:** K. Mockus and D. Weiss (2000) discuss key features of an effective bug tracking system, including defect logging, prioritization, assignment, and resolution tracking. They highlight the importance of a centralized database that allows developers to retrieve historical data for analysis and comparison, which can be instrumental in predictive analytics and understanding recurring bug patterns. Current systems often include these features but also extend functionality to include collaboration, communication tools, and automated notifications

**Challenges in Bug Tracking:** Multiple studies, including research by J. Bettenburg et al. (2008), have identified common challenges in BTS, such as ambiguous bug reports, information overload, and redundancy. Ambiguous reports often lead to miscommunication between testers and developers, requiring additional clarification and slowing down the debugging process. Bettenburg et al. proposed a framework for “good quality bug reports,” emphasizing the importance of completeness, reproducibility, and severity assessment in report quality.

**Automated Bug Tracking and Prediction:** Recently, machine learning and natural language processing (NLP) techniques have been applied to improve bug tracking and classification. According to research by M. Tian et al. (2012), predictive models can help categorize bugs based on historical data, automatically assign them to the appropriate developers, and even predict the time required for resolution. These advancements are particularly beneficial in large-scale projects where manual classification and assignment would be time-intensive.

**Integration with Other Development Tools:** Modern BTS are often integrated with software development tools, such as version control systems (e.g., Git) and Continuous Integration/Continuous Deployment (CI/CD) pipelines. Studies by G. Antoniol et al. (2001) show that integrating bug tracking with version control can streamline debugging, as code changes can be directly linked to specific bugs. This integration not only improves traceability but also facilitates historical analysis of how certain changes impact system stability over time.

**Usability and User-Centered Design:** Studies have highlighted the importance of designing user-friendly BTS to ensure high adoption rates and efficient reporting. Research by D. Cao and A. Ramesh (2008) emphasizes the role of human-computer interaction (HCI) in BTS design, suggesting that a well-designed user interface and user experience (UI/UX) are critical for efficient bug reporting and tracking. User-centered design principles, such as simplified reporting forms, guided input, and real-time feedback, can help reduce report ambiguity and improve overall BTS usage.

**Security and Privacy in Bug Tracking:** As software applications increasingly store sensitive data, security in BTS has become a focal point. Bug reports can inadvertently expose system vulnerabilities, making secure access control and data encryption essential. C. Sheffer et al. (2014) discuss the importance of safeguarding BTS data to prevent unauthorized access, especially in enterprise environments where sensitive business logic or customer data might be involved in bug reports.

**Advanced Analytics and Reporting:** Advanced analytics in BTS help teams gain insights into common defects, recurring issues, and team performance. Research by H. Zhang et al. (2013) presents the use of data mining techniques within BTS to discover patterns in bug recurrence and repair times, which can inform process improvements. These analytics help teams assess trends in software quality over time and make data-driven decisions to enhance system stability and efficiency.

**Mobile and Cloud-Based Bug Tracking Systems:** With the rise of cloud computing, many modern BTS have migrated to cloud-based platforms, providing easier access, scalability, and remote collaboration options. Studies by S. Pham and M. Phung (2015) emphasize the advantages of cloud-based BTS, including enhanced accessibility for distributed teams and reduced maintenance overhead. Cloud-based BTS can also provide real-time updates and synchronization, supporting agile methodologies and faster development cycles.

1. **REQUIREMENTS**

**3.1. Functional Requirements**

* + 1. **User Registration and Authentication**

Users should be able to register with unique usernames and secure passwords.

User authentication must be secure, and role-based access control (e.g., admin, developer, tester) should be implemented.

* + 1. **Bug Reporting**

Users can log new bugs with essential details such as title, description, priority, severity, affected module, and steps to reproduce.

Bug reports should support attachments (e.g., screenshots, log files) for additional context.

* + 1. **Bug Management**

Users should be able to update bug statuses (e.g., New, Open, In Progress, Resolved, Closed).

The system should allow assigning bugs to specific team members and setting due dates.

* + 1. **Prioritization and Severity**

Bugs should be categorized by priority (e.g., Low, Medium, High) and severity (e.g., Critical, Major, Minor).

Users should be able to filter and sort bugs based on these criteria.

* + 1. **Search and Filter**

Users should be able to search bugs using keywords, filters (e.g., status, priority, assignee, creation date), and tags.

The system should allow saving commonly used search filters.

* + 1. **Notifications and Alerts**

Users should receive notifications on relevant actions (e.g., when a bug is assigned, updated, or closed).

Notification preferences should be customizable for each user.

* + 1. **Comments and Collaboration**

Users should be able to comment on bug reports for additional context or clarifications.

Threaded discussions should be supported to streamline communication within bug reports.

* + 1. **Dashboard and Analytics**

A dashboard should display an overview of active bugs, priority breakdowns, team workload, and trends.

Analytics on average resolution time, bug recurrence rates, and developer productivity should be provided.

* + 1. **Role-Based Access Control (RBAC)**

Role-based access control should ensure that only authorized users can perform specific actions (e.g., only admins can delete bugs, testers can report bugs but not close them).

* + 1. **Audit Trail**

The system should log all actions and changes made to each bug report, providing a historical trail for accountability.

**3.2. Non-Functional Requirements**

* + 1. **Usability**

The system should have an intuitive and user-friendly interface to facilitate ease of use.

It should provide help documentation, tooltips, and guidance for new users.

* + 1. **Scalability**

The BTS should handle a growing volume of bug reports and users as projects and teams expand.

It should support horizontal or vertical scaling to manage increased data loads.

* + 1. **Performance**

The system should respond within acceptable time limits for various actions (e.g., opening a bug report, searching, filtering).

It should support efficient retrieval of bug data even as the number of reports grows.

* + 1. **Reliability and Availability**

The BTS should be highly reliable and have minimal downtime.

Data recovery mechanisms should be in place to prevent data loss.

* + 1. **Security**

Data encryption should be used to protect sensitive information in bug reports.

Role-based access controls should prevent unauthorized access to critical system features.

The system should log and monitor for suspicious activity to protect against security threats.

* + 1. **Integration**

The BTS should integrate with other development tools (e.g., version control, CI/CD tools, project management software) to streamline the workflow.

Support for API access should allow third-party tool integration.

* + 1. **Data Backup and Recovery**

Regular backups should be automated to prevent data loss, with a defined recovery procedure in case of data corruption or system failure.

* + 1. **Compatibility**

The system should be compatible with major web browsers and mobile platforms if mobile accessibility is required.

It should also adapt to different screen resolutions to support various devices.

* + 1. **Compliance**

If handling sensitive data, the system should adhere to data privacy regulations (e.g., GDPR, CCPA) and ensure user data protection.

* 1. **System Requirements**
     1. **Server Requirements**

Should be hosted on a reliable server infrastructure with backup and failover capabilities.

Suitable for cloud or on-premises deployment, depending on organizational needs.

* + 1. **Database Requirements**

A relational or NoSQL database should be used to store bug report data, user information, and logs.

The database should support indexing and optimized querying for high performance.

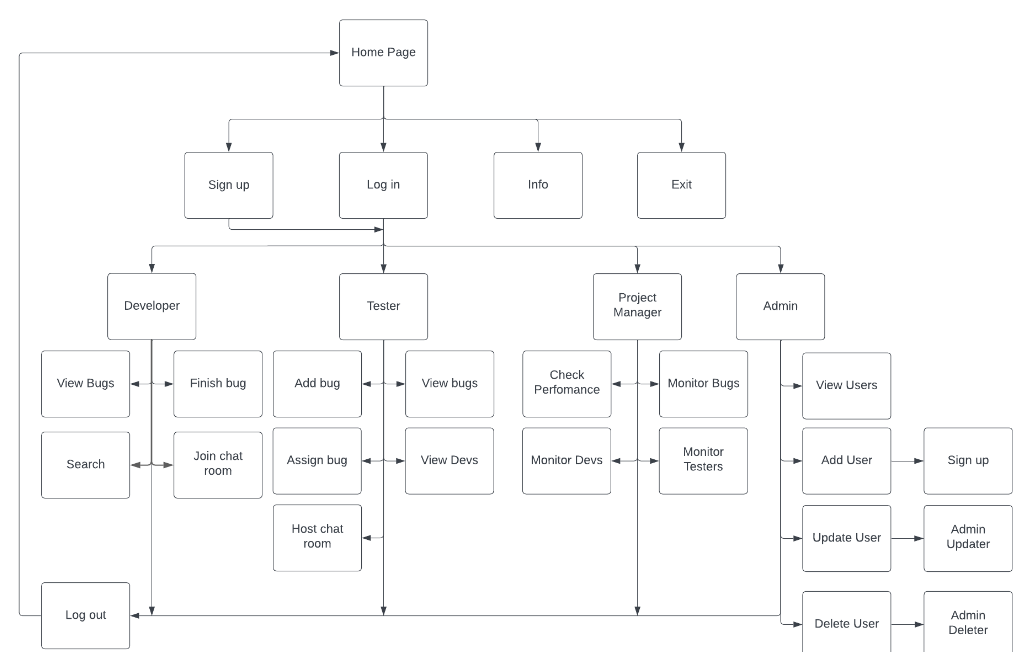
* + 1. **Technology Stack**

The technology stack may vary, but the system should ideally be built using a scalable and secure framework (e.g., Node.js, Django, or Spring Boot) and support web technologies (HTML, CSS, JavaScript).

Consider RESTful APIs or GraphQL for backend communication and frontend frameworks (React, Angular, or Vue.js) for an interactive user interface.

1. **ARCHITECTURE AND DESIGN**
   1. **Architecture Diagram**

The architecture diagram is as follows:



* 1. **GUI Design**
     1. **Bug Reporting Module:**

**Bug Details:** Fields for entering bug title, description, severity level, priority, assigned developer, and due date.

**Attachments:** Allows users to upload screenshots, log files, or other relevant documentation.

**Custom Fields:** Provides flexibility to add additional fields specific to the organization's needs.

**Workflow:** Defines the bug lifecycle, such as "New," "Assigned," "In Progress," "Resolved," and "Closed."

* + 1. **Dashboard:**

**Overview:** Displays a summary of the bug tracking system's status, including the number of open, closed, and in-progress bugs.

**Charts and Graphs:** Visualizes data to provide insights into bug trends, severity levels, and resolution times.

**Recent Activity:** Shows a feed of recent bug updates, such as new reports, status changes, or comments.

* + 1. **Bug Search and Filtering:**

**Search Bar:** Allows users to search for bugs by keyword, title, description, or other criteria.

**Filters:** Provides options to filter bugs by status, severity, priority, assigned developer, due date, and custom fields.

**Saved Searches:** Enables users to save frequently used search criteria for quick access.

* + 1. **Bug Detail View:**

**Comprehensive Information:** Displays all relevant details about a specific bug, including its status, history, comments, and attachments.

**Comments:** Allows users to add comments, questions, or updates to a bug.

**Attachments:** Provides a list of all attached files.

**Workflow Actions:** Enables users to change the bug's status, assign it to a different developer, or update other details.

* + 1. **Developer Portal:**

**Assigned Bugs:** Displays a list of bugs assigned to the developer.

**Bug Detail View:** Provides the same functionality as the bug detail view for general users.

**Time Tracking:** Allows developers to record time spent on each bug.

**Code Repository Integration:** Links to the organization's code repository to facilitate bug analysis and resolution.

1. **IMPLEMENTATION**

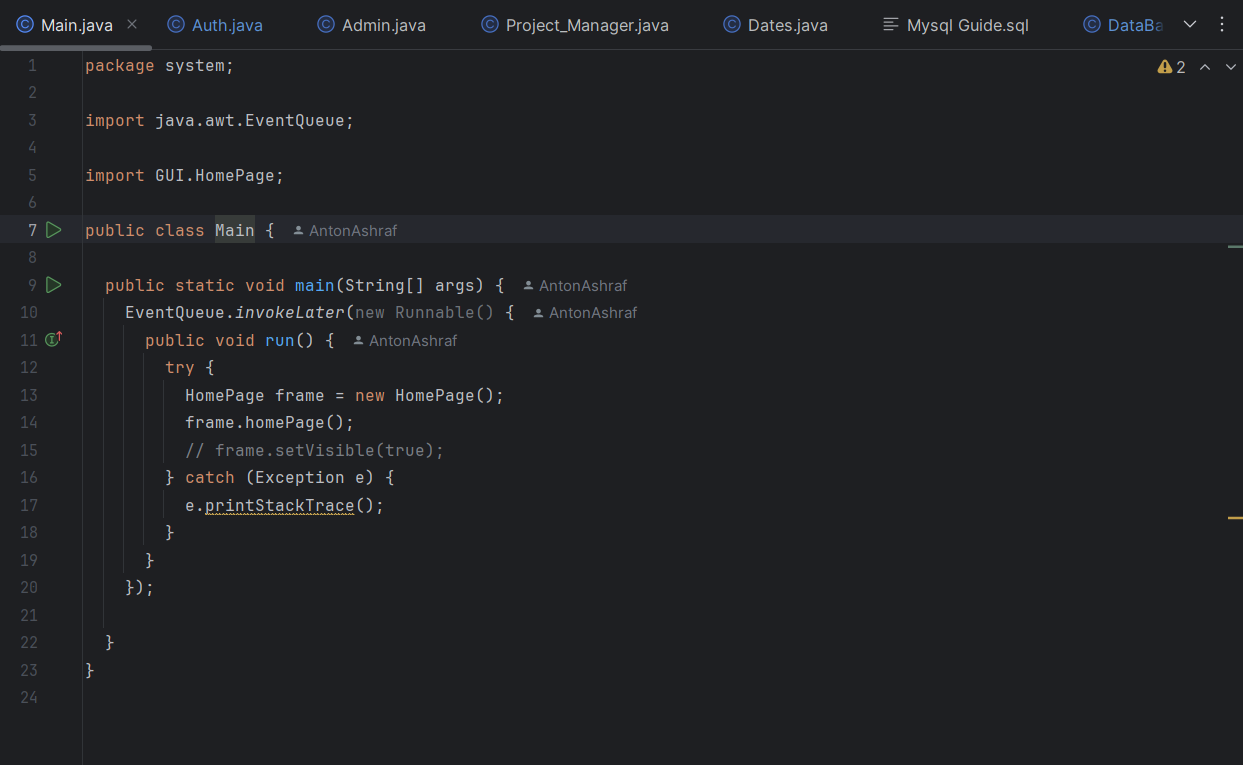
The system's GUI, developed using Intellij IDEA, provides a user-friendly interface for interacting with the bug tracking functionalities. It integrates an online shared MySQL Workbench 8.0 CE storage as the database backend, ensuring reliable and secure data storage and retrieval.

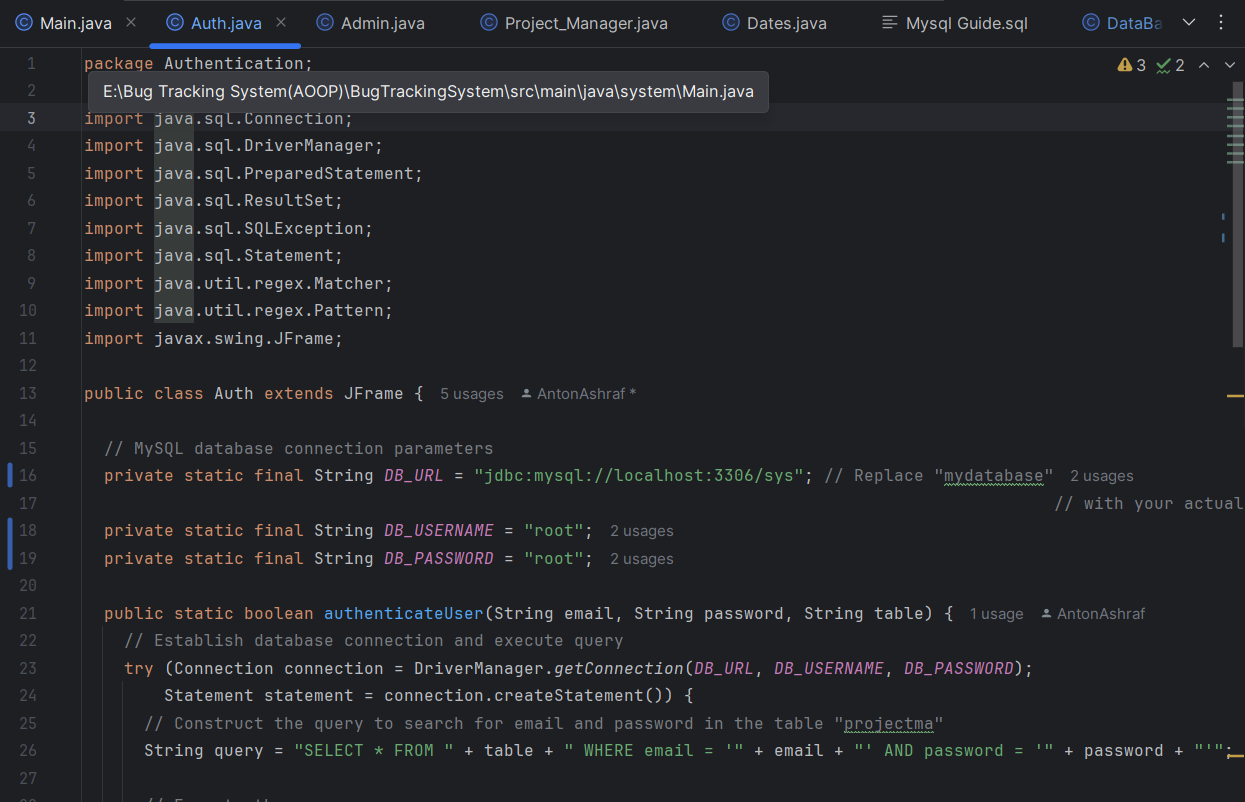
In addition to bug tracking capabilities, the project also includes an authentication system that validates email addresses and ensures data integrity in the database. This adds an extra layer of security and prevents redundant or invalid data from being stored.

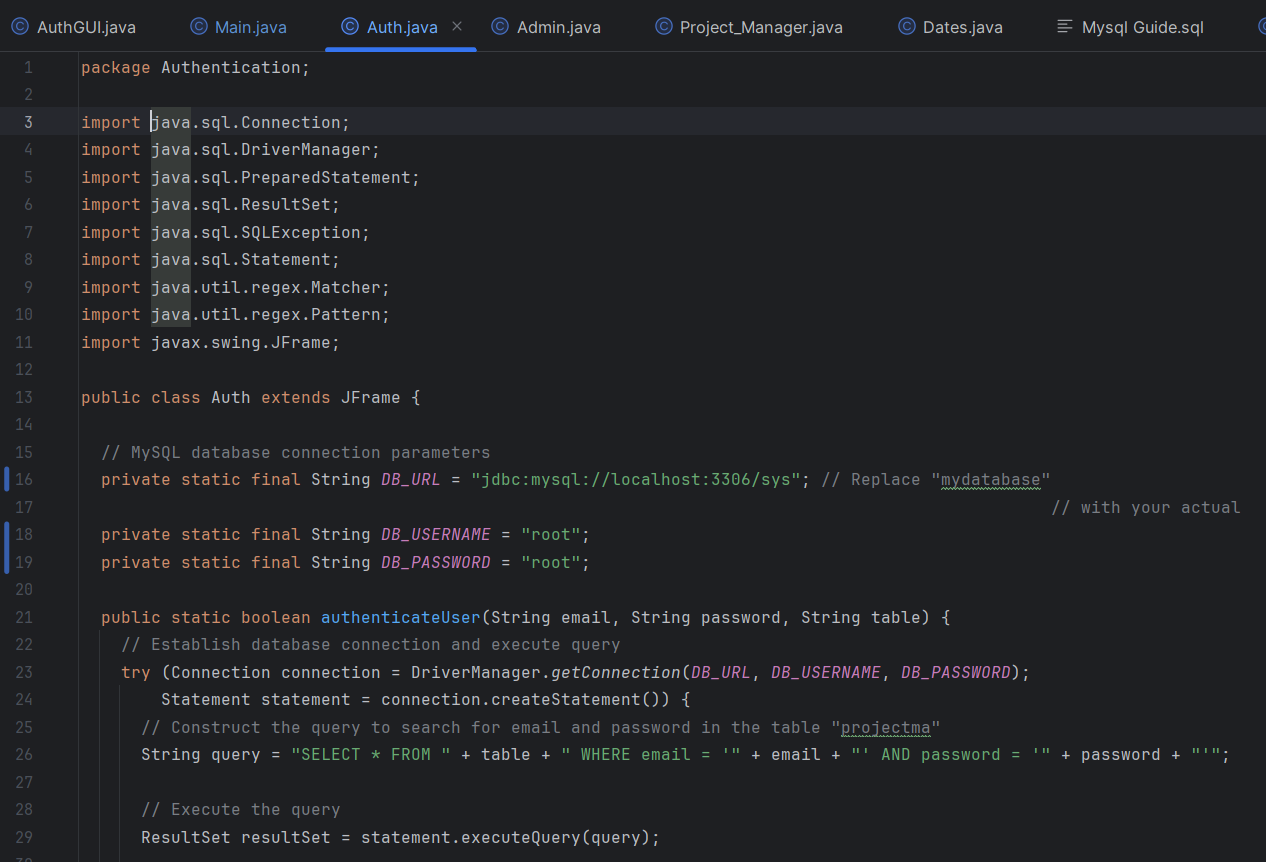
Furthermore, to the mentioned functionalities, the project is structured as a Maven project. Maven is a powerful build automation tool for Java projects that helps manage dependencies, build processes, and project configuration. By utilizing Maven, the project benefits from simplified dependency management, consistent build processes, and improved project structure.

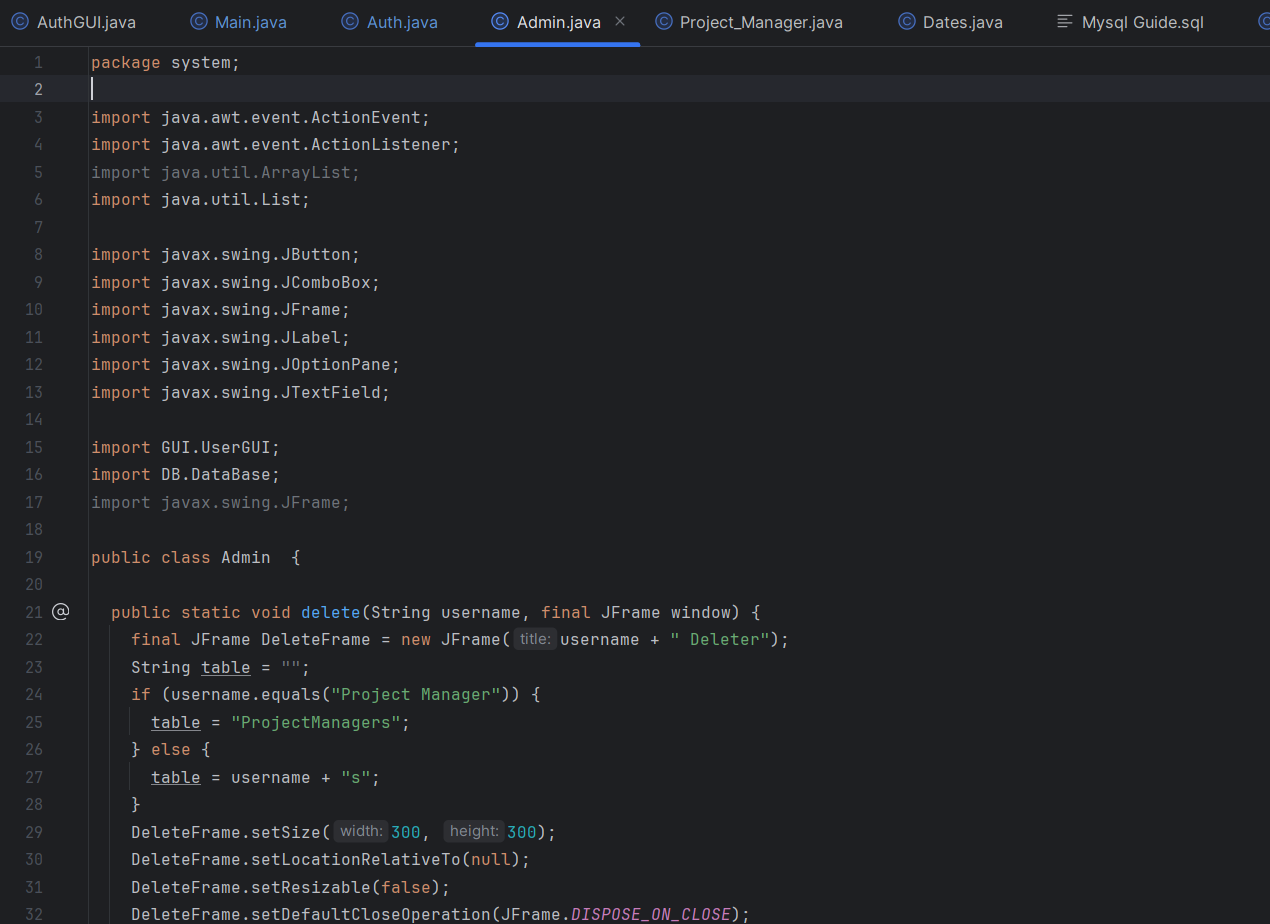
With Maven, the project can easily manage and resolve external libraries and dependencies required by the Intellij IDEA GUI and the MySQL database integration. Maven's dependency management simplifies the process of adding, updating, and resolving library dependencies, ensuring that the project has all the necessary components for successful compilation and execution.

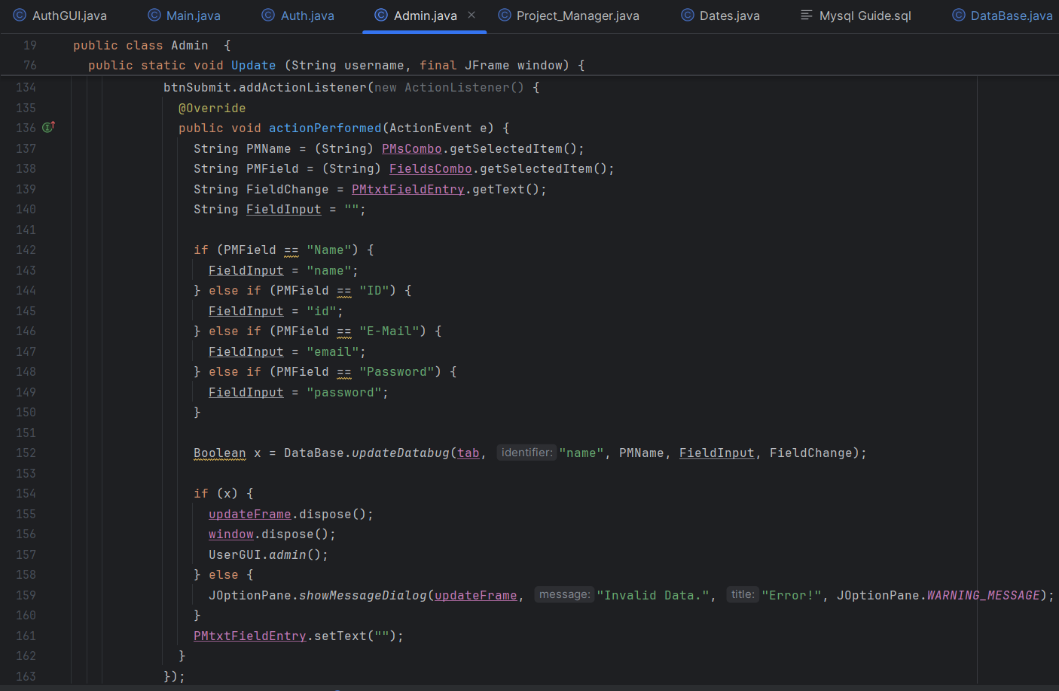
* 1. **Sample Code**

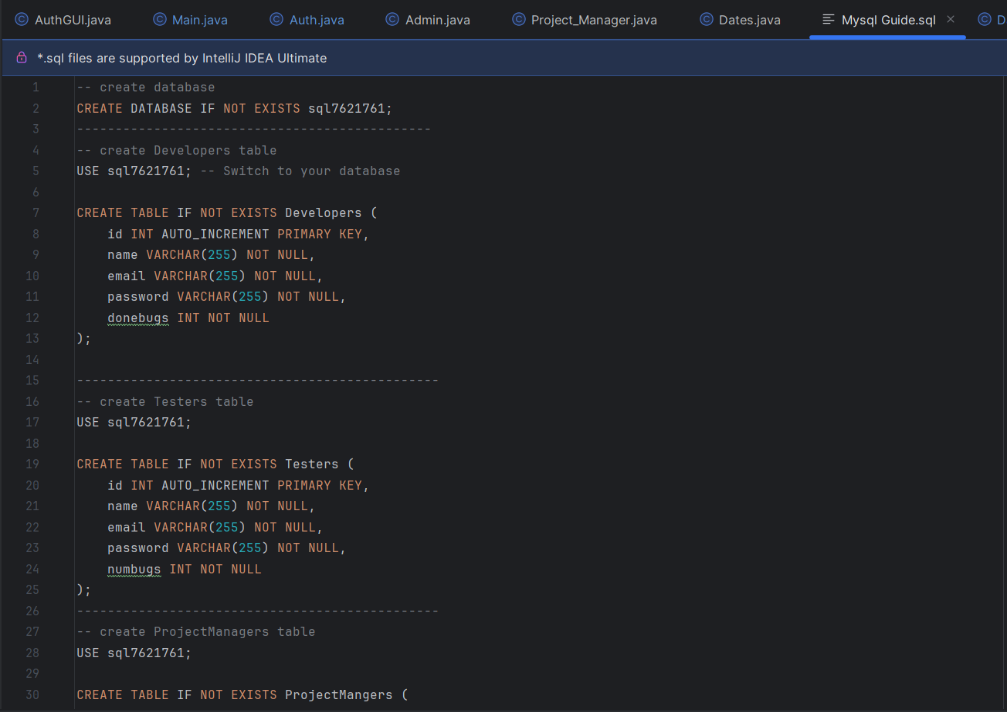


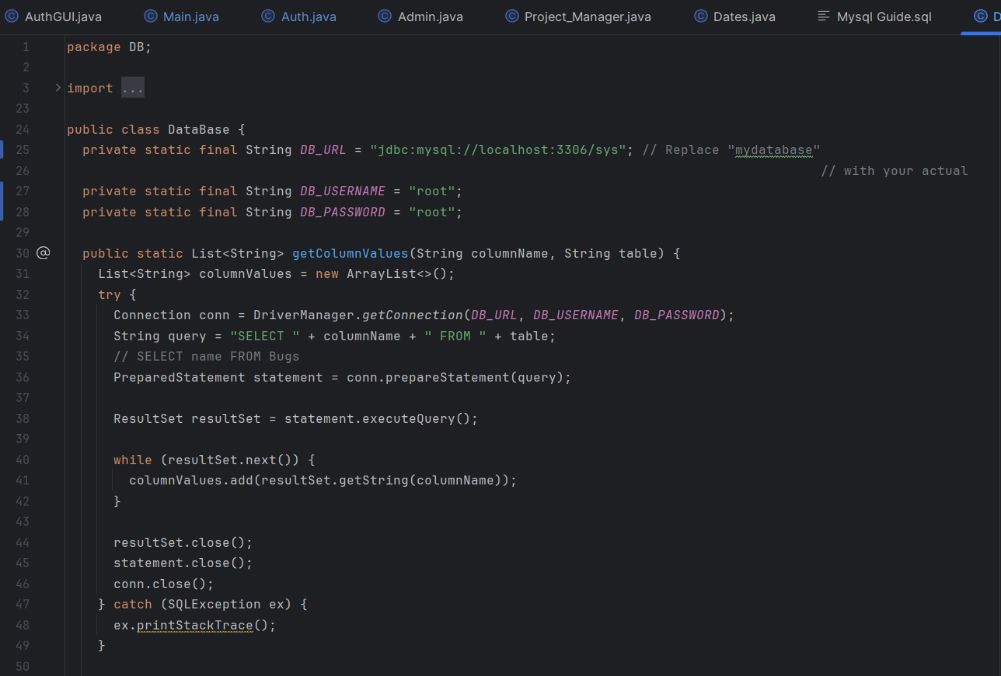


****

****

****

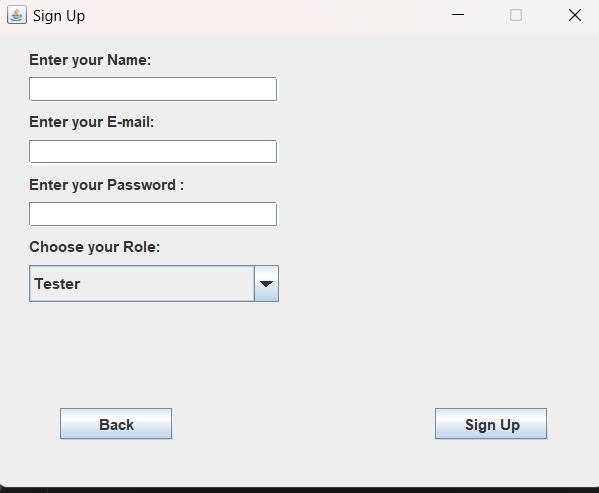
****

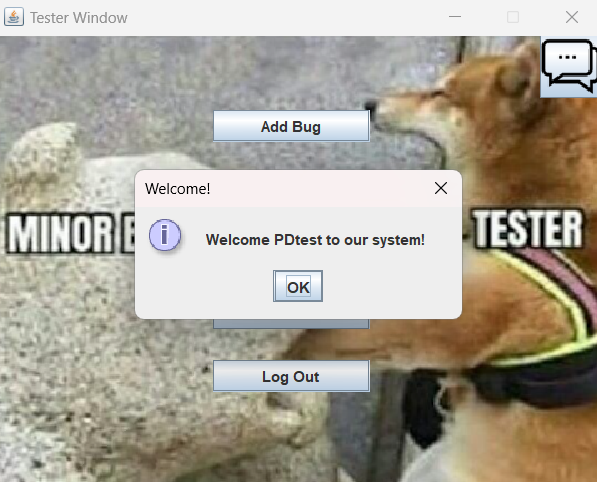
****

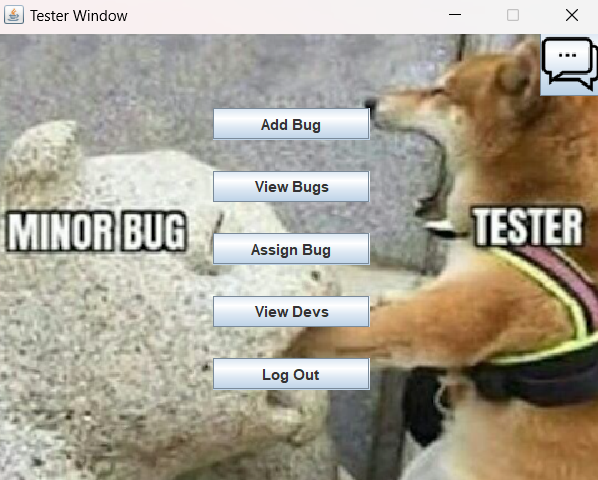
1. **RESULTS AND DISCUSSION**

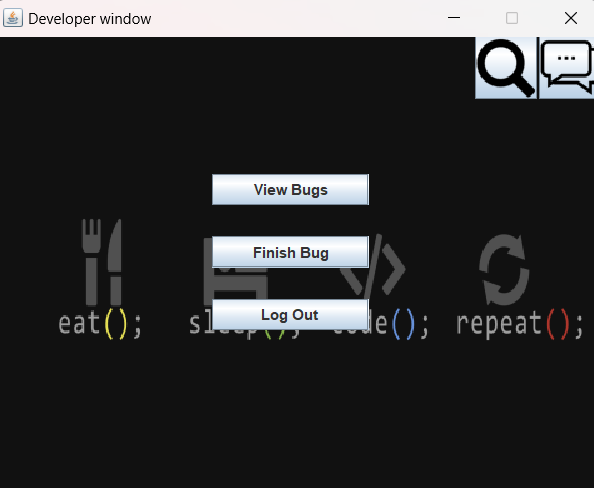
The Bug Tracking System serves as a collaborative workspace where four key roles—Tester, Developer, Project Manager, and Admin—come together to detect, track, and debug software issues efficiently. This unified platform enhances collaboration by enabling each role to fulfill its responsibilities seamlessly. Testers report and categorize bugs, provid4ing the necessary details for Developers to diagnose and fix issues. Project Managers monitor the status and prioritize bugs based on project timelines, ensuring that high-impact issues are resolved quickly. Admins manage permissions, oversee workflows, and ensure data integrity across the system. As a result, this structured approach leads to faster bug resolution, improved product quality, and streamlined communication among team members.





****

****

****

1. **CONCLUSION**

In a nutshell, a BTS is always quite an invaluable tool that software development teams can use for managing, identifying, and resolving software issues efficiently. A well-designed BTS keeps a centralized platform for documentation, prioritization, and monitoring bugs, thereby contributing to more collaborative and accountable teams, which then translates into improved user satisfaction with the quality of the software.

Systematic bug tracking enables teams to classify the issues as to their severity and impact; hence critical problems are addressed quickly, while smaller bugs or problems are managed efficiently. Moreover, a BTS provides insights regarding the project's progress, resolutions of bugs and trends, and product stability through analytics and reporting features that facilitate the making of informed decisions for continuing improvement processes.

Despite challenges such as the need for thorough user training, system customization, and regular maintenance, the benefits of implementing a Bug Tracking System far outweigh these considerations. When tailored to meet specific project needs, a BTS becomes an essential component in delivering reliable, high-quality software products in a competitive and dynamic industry.

In conclusion, investment in a bug tracking system essentially streamlines the lifecycle of software development while, at the same time, promoting greater proactivity in quality assurance activities, leading to, in the long run, better software project delivery.

1. **REFERENCES**

* GitHub: <https://github.com/>
* GeeksforGeeks: <https://www.geeksforgeeks.org/>
* YouTube: <https://www.youtube.com/>
* Bugzilla Project: <https://www.bugzilla.org/docs/>