A Role-based bug management system on JavaFX and MVC Architecture Design and Implementation

Pushkar Raj[1], Rahul Tiwari[2], Dr. C Ramesh Kumar[3]
[1,2]*Computer Science and Engineering, Galgotias University Greater Noida, India*
[3]*Guide Computer Science and Engineering, Galgotias University Greater Noida, India*

***Abstract*—The paper is a design and implementation of a bug management system in Java and a JavaFX GUI that is based on the Model-View-Controller architecture (MVC). The system does give secure user access control through role-based access control (RBAC) to three user roles Admin, Project Manager and Developer. Every role has a tailored dashboard that contains actions pertaining to the duties of the role. Users with the permission to create projects can allocate developers, and monitor bug tickets within a specific project (with assignment, status and comments). There is the use of core SOLID principles e.g. controllers are devoted to one task (Single-Responsibility Principle), extension of functionality through new classes, without altering the existing code. JavaFX interfaces were created using FXML (Scenes Builder), and data persistence is created using a MySQL database. A comparison reveals that the given system includes the necessary bug-tracking features (issue creation, issue status management, comments) and does not have advanced features (including agile boards and wide-ranging third-party integrations) that other tools, such as Jira, do. Using IntelliJ IDEA, this prototype was created and tested using MySQL as a backend to show proper role enforcement and functionality. On the whole, this project provides a full bug-tracking system with an explicit separation of roles and a decent software structure, which can be used in educational and small teams.**

***Index Terms*—Role-based Access Control, Bug Tracking System, JavaFX, MVC Architecture, Software Design Principles**

I. INTRODUCTION

In software development, bugs and defects are part and parcel and systematic tracking is a must to ensure that software quality is upheld [9], [10]. The bug tracking tools offer a centralized database of bugs, where teams can track the statuses of bugs and rank the bugs to get them resolved [2]. Role-based access control (RBAC) is an administrative system in most systems where permissions are granted to roles but not given to individuals [1]. But most of the bug trackers are complicated, and they can be too confusing to small teams. This project will respond to the requirement of having a lean, desktop-based bug tracking tool that has a distinct delineation of roles. We create a Java application that is represented by a JavaFX GUI [3] and MySQL [6] rear end. The system is based on MVC and separates the model (data), view (UI), and controller (logic) components [11], [13]. Using the principles of SOLID, each class has a specific task to perform and an extension can add new functionality without the need to make any changes to an existing code [8], [17]. The system provides three roles (Admin, Project Manager, Developer) with an individual dashboard and specific capabilities. The main ones are the creation of projects, the life cycle of bug tickets (creation, assignment, status upgrades) and team-based commenting. Secure authentication is used to ensure that a user can only carry out what he/she is allowed to do by his/her role. The general architecture promotes the simplicity of the workflow and teamwork of small-to-medium scale [4], [5].

The associated literature reveals that a good bug tracking enhances interaction between the developers and testers. The previous solutions, however, do not have fine-grained role-specific interfaces or are web-based, which can be less intuitive when it has to be controlled. In this paper, a completely operational Java/JavaFX application that includes design features of RBAC and MVC is presented. The remainder of the paper is structured in the following manner Section II is a survey of related work Section III introduces the system requirements and features Section IV is the description of MVC-based architecture Section V

presents the role-based workflows Section VI contains the description of the database schema and the ER design Section VII is the description of the JavaFX implementation Section VIII is the description of security and access control Section IX discusses the features compared to the existing tools Section X presents the results and evaluation section XI is the discussion of the limitations and future work and Section IX is the conclusion.

II. RELATED WORK

There are a number of bug trackers. Jira by Atlassian is an all-encompassing agile issue management system [4], and Bugzilla and MantisBT are open trackers that are well-established [5]. Bugzilla, in particular, is capable of advanced searching, email notification, scheduling and duplicate detection. MantisBT is extensible through the use of plugins and has a variety of clients. Batra and Yadav make a comparative analysis of Jira, Bugzilla, Zoho and MantisBT and point out the differences in features and usability of the applications. The focus of such studies is that various tools are appropriate in various situations.

Design wise, MVC is a uniform framework of GUI [11], [13]. The principles of SOLID (Single Responsibility, Open-Closed, etc.) are popularized in order to have maintainable, and extensible codebases. These paradigms [8], [17] are used in our system.

To conclude, available tools such as Jira focus more on agile processes and integrations, and Bugzilla focuses more on efficient querying and extensibility. Desktop-based Java implementations with explicit RBAC have had less work. The paper seals the gap by creating a desktop bug tracker that separates roles and one that has an updated user interface.

III. SYSTEM OVERVIEW

The following are the main features exhibited by the implemented system:

Role-Based Access Control: There are three user roles (Admin, Project Manager, Developer), each of which has particular permissions and a personal dashboard [9].

Authentication: This is done by authentication of both the signup and the login process to make sure that the user is authenticated to access any system functionality.

Dashboard System: Once a user logs in, they can view a dashboard that includes role-specific information and summarized data and actions that users are permitted to perform.

Project and Task Management: Authorized users will be able to create new projects, define project characteristics and assign a developer (or developers) to a particular project. Project Managers control the given projects.

Bug Lifecycle Tracking: In every project, authorized users (Admins or Project Managers) are able to create bug tickets by filling the following information a title, descriptions, severity, and initial status. Every bug can be then allocated to a particular developer. The assigned bugs are displayed in the dashboard of developers and can be edited by them (e.g., open, in progress, resolved) and it is possible to comment on the bug. Statuses and timestamps are recorded so as to give detailed lifecycle of each bug.

All the features are consistent with the common sense of using bugs-tracking and are introduced following the MVC design [11].

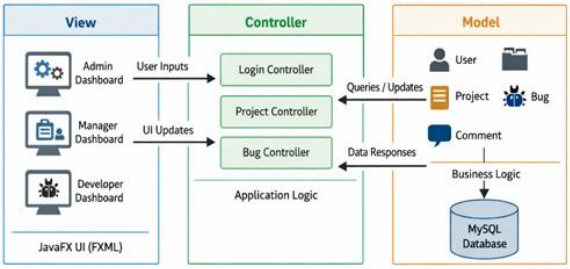

Figure 1: System Architecture Diagram (MVC-based)

IV. ARCHITECTURAL DESIGN (MVC-BASED)

The system is strongly designed according to the Model-View- Controller (MVC) architectural design. Here, Model layer contains application data and business code (such as classes of Users, Projects and Bugs, and database access code). The View layer comprises the JavaFX UI each screen is represented by an FXML file (written by Scene Builder) rendering the model data and getting the input. The Controller layer has event-handling code an example is an AddBugController, which reacts to the UI events in order to create new bug records. This implements the Single-Responsibility Principle each controller is dealing with a single functional unit (e.g. adding a bug or assigning a developer). The design is also extensible

it is possible to add new features without having to change the existing code (Open-Closed Principle) [11],[13] through the addition of new controllers and views.

In our implementation, Java packages are grouped into model, view and controller. The database tables are represented by model classes (User, Project, Bug etc.) and specify data fields and access methods. The files with the view are FXML layouts (created using Scene Builder) which allow isolating the UI design and logic. These views are bound to these controller classes controllers are injected with fx:id identifiers through annotations of type FXML. On interaction by the user (e.g. by clicking a button), the controller processes this event and updates the model (and, by extension, the database).

The MVC structure represented in figure 1 is as follows model classes are associated with a MySQL database, controllers are the ones that mediate between the events of a view and the actions of a model, and the views are those that render the UI. This division enhances modularity and maintainability as the UI and data components can be developed separately.

V. ROLE-BASED WORKFLOW AND FUNCTIONAL MODULES

The system is separated into modules that reflect significant functionality, which are imposed by RBAC: Authentication Module: All users (Admins, Project Managers, Developers) can log-in through a secure login form. The system compares credentials and identifies the role of the user against the database.

Project management Module: It allows project admins and project managers to create new projects (name and description) and allocate developers to projects. Specific projects and activities are then put on individual developer dashboard.

Bug Management Module: In every project, authorized users (Admin or Project Manager) are able to add bug tickets, with the following fields title, description, severity, and status. The developers are given bugs and the bugs are displayed on their dashboard. The developers have the ability of updating the status of a bug (e.g. marking it as resolved) and commenting on it. Changes (status and timestamps) are recorded to the controller code to give a history of life cycles.

Comment/Collaboration Module: Team members can make textual comments on any bug ticket to support the exchange of comments. These modules will be accessed based on the role of the individual user. Examples are that a Project Manager would have choices to add projects and bugs, unlike a Developer who only has the tasks that should be done. This gives the users a privilege of only accessing what they are allowed to as per the privileges of the role. The flow of role-based access is depicted in figure 2.

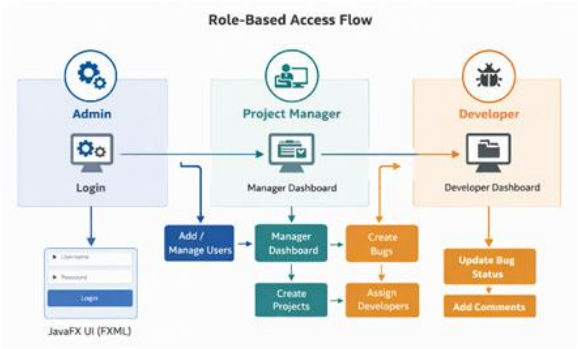

Figure 2: Role-Based Access Flow Diagram

VI. DATABASE DESIGN AND ENTITY RELATIONSHIPS

The schema of the database is based on the standard entity relationship (ER) model, which describes both entities (tables) and relationships between the entities [16]. Key entities include:

- User: User data (userID, username, password hash, and so on) are stored with a foreign key to Role table. Users are enforced to have one role each (RBAC).
- Role: Defines roles (Admin, Project Manager, Developer) and associated privileges. The roles have their own ID and name.
- Project: Depicts a project (projectID, title, description, the creation date). Projects belong to many-to-many (through a join table) developers and one or more bugs.
- Bug: This is an example of a bug ticket (bugID, title, description, status, severity, creation, and update time). The bugs are given foreign keys to the project they belong in and the user they are associated with.
- Comment: Represents comments on bugs (commentID, bugID, authorID, text, timestamp). There is one comment to each bug and author (user).

Relationships Each Bug is connected to a single Project each Bug is assigned to a single User projects and users are many to many relationships (there is a many-to-many association between projects and developers) [6]. Figure 3 depicts this design.

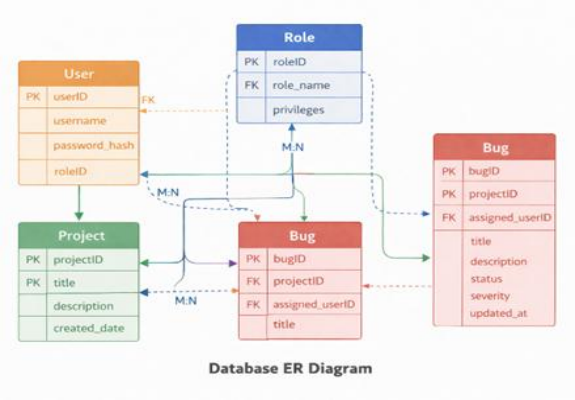

Figure 3: Database ER Diagram

Information is stored in a MySQL database (the most popular open-source database of the world). The schema is a close-up of the entities and relationships above, guaranteeing the referential integrity and fast search of the bugs and project data.

VII. IMPLEMENTATION DETAILS USING JAVAFX

JavaFX and FXML [3] were used as the implemented user interface. Each screen was designed with Scene Builder (Oracle JavaFX Scene Builder) to produce FXML files that separate the design of the UI and code. Every FXML view is an object which is accompanied by a controller class (e.g., AddProjectController, BugListController). Controllers Event-handler methods (i.e. button clicks) are connected to UI controls, using @FXML. As an illustration, the Create Bug button gets a controller method, which reads input fields and makes a new Bug model object.

Dynamic data (e.g. lists of bugs or projects) are presented using JavaFX TableView controls and ObservationalLists[7]. The controllers refresh these lists as the database is changed and the UI is automatically updated. SQL exception handling and input validation is done in the controller code. As an example, failure to insert data in a database will cause an error dialog to be displayed.

The transitions between the scenes (between login, dashboards, and forms) occur through loading alternate FXML scenes to the main stage. Upon a user log in, the system loads this or that role-based dashboard. Connection to the database data-base connectivity will be provided with the help of the MySQL Connector/J driver (version 8.0.26). Connection (URL, credentials) is stored in a utility class, which allows using it in all controllers. Parameter binding (Prepared Statement) is employed on all operations of the database in order to avoid SQL injection. In the testing, the MySQL database was located on an external development server (freemysqlhosting.net) to replicate a production environment. With JavaFX and FXML, the definition of the UI is totally decoupled of application logic. This is a modular design (a controller per screen) which is easy to maintain and extend.

VIII. SECURITY AND ACCESS CONTROL MECHANISMS

The multi-level security is employed. The user credentials (usernames/ passwords) are stored in a MySQL table. The passwords will be hashed securely and then stored which means that the database will not contain plain-text passwords. When a user logs in the password typed in is hashed and matched with the existing hash to eliminate credential theft [6].

RBAC is applied in the application after authentication. Any action is authorized by checking at the role of the logged in user before a record is made. As an example, menu items and buttons on each dashboard are turned on or off depending on the role and every controller method checks the role of the user before making important alterations. An Admin will be able to do everything (create or delete user accounts), Project Manager create projects and bugs, and the Developer will only be able to update the bugs assigned to him. This will make sure that a given function is only accessed by the authorized users. PreparedStatement is used to run SQL queries to prevent the occurrence of SQL injection. Any exceptions or invalid inputs are identified and generated into user friendly error messages. In production, the application can be run with connections enabled with an encryption of data in transit using SSL-enabled connections (Connector/J supports SSL). Overall, the system is best practice passwords are hashed, all operations are role checked, and database queries are secure.

IX. COMPARATIVE FEATURE ANALYSIS

Table 1 compares the proposed system features with the Jira Software and Bugzilla features, according to the official documentation of features. The suggested system will offer the fundamental tracking capabilities (creation of cores, creation of projects/issues, status updates, commenting) [4], [5]. Jira, on the contrary, has complete agile-support (e.g. Scrum/Kanban boards), advanced dashboards and reports, REST APIs and a huge marketplace of add-ons. Bugzilla can focus on effective search and reporting (scheduling, charts) and allow customization of workflows and many authentication systems. In contrast to such systems, our application is not a desktop-only tool and lacks a plugin system and a fixed set of features.

Table 1: Feature comparison of proposed system vs. Jira and Bugzilla.

Feature	Proposed System	Jira Software	Bugzilla
Role-based access control	Yes (Admin, PM, Dev)	Yes (user/project roles)	Yes (group-based)
Project/Issue creation	Yes	Yes	Yes
Scrum/Kanban boards	No	Yes	No
Custom workflows	No (fixed)	Yes	Yes
Dashboards & reports	Basic (role dashboards)	Advanced, interactive dashboards	Charting and reports
REST API & integrations	No	Yes (extensive REST API)	Yes (XML/JSON web services)
Plugin ecosystem	No	Extensive add-ons	Limited (extensions possible)
Email notifications	No	Yes (configurable)	Yes (configurable)
Deployment	Desktop application (JavaFX)	Cloud or Data Center	Self-hosted web server

X. RESULTS AND DISCUSSION

Results: Simulates normal situations of each of the roles. An example is Admins who created projects and user accounts and Project Managers who created projects and bug tickets and Developers who saw their assigned bugs and updated them. The system granted role permissions in the right way and ensured data consistency. Error checking was done (ex:invalid inputs were rejected and error report on the database). A test project that had several bugging tickets was developed and tested out and showed that the changes in the database were reflected in the UI. Simple performance tests had revealed that database queries could be done in a short time (milliseconds) and the GUI could hold moderate-sized workloads. In short, the system meets the requirements the authenticated and role-specific workflows, as well as end-to-end bug lifecycle, were met [9], [10], [17].

Discussion: The MVC pattern was good the data model was not changed with the UI layouts, which validated the separation of concerns. Compliance with SOLID principles made it easier to add such functionality (such as adding new functionality can be implemented by writing new controllers/views without bringing any changes to the existing code). JavaFX (accompanied by FXML) made development of the UI faster, whereas MVC and SOLID enhanced maintainability. Table 1 indicates that the system satisfies the minimum tracking requirements but excludes the advanced capabilities of the enterprise tools. In total, the findings support the assumption that a modular JavaFX/MVC system can provide a functional bug tracker that can be adapted to be used by an educational institution or a small team.

XI. LIMITATIONS AND FUTURE SCOPE

This prototype was experimented using a small user base and small data. It does not have enterprise features like email notices, version control or CI/CD integration and sophisticated search or reporting. The interface also simplifies (there is no full-text search or rich-text comment) [13], [15]. The database is simple (no indexes or caching) and it now stores passwords in plaintext (some work will be required to add secure hashing in the future). The work might include in the future add-on extensibility, reporting dashboard, full-text search, hashing of passwords and auditing/logging. The system has not been tested at the level of heavy concurrency or security penetration. The move to web or mobile interface might enhance accessibility. It should also be improved on with automated testing and continuous integration to improve reliability.

XII. CONCLUSION

The work has generated a role-based bug management system that meets the necessary criteria of issue tracking (creation of the project, assigning tickets, status, commenting) with Java, JavaFX, and the MVC architecture [9], [10]. Using the principles of MVC and SOLID, the design is a modular one: independent controllers are in charge of specific features and it can be extended easily. The system implements three-role user authentication and permissions based on role and a MySQL database to store the system. A comparator study demonstrated that it accommodates all the main tracking features, which is compatible with the regular industry tools. The software engineering principles (architecture, design patterns) are also implemented in the project, which makes it a valuable case study that can be used by students. Its prospects (additional extensions, e.g. REST API access, better reporting, or collaborative integrations, etc.) may be further extended.

REFERENCES

[1] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," IEEE Computer, vol. 29, no. 2, pp. 38–47, Feb. 1996.

[2] Y. Batra and S. Yadav, "Study of various bug tracking tools (a case study of Jira, Bugzilla, Zoho and MantisBT)," International Journal of Computer Applications, vol. 2, no. 6, pp. 45–54, 2020.

[3] Atlassian Jira Software Features, "Jira Software Features," Atlassian Documentation, 2022.

[4] Bugzilla Documentation, "Bugzilla: The Bug-Tracking System," Mozilla Foundation, 2021.

[5] MySQL 8.0 Reference Manual, Oracle, "MySQL 8.0 Reference Manual," 2021.

[6] I. Sommerville, Software Engineering, 10th ed. Pearson, 2015.

[7] R. S. Pressman, Software Engineering: A Practitioner's Approach, 8th ed. McGraw-Hill, 2014.

[8] M. Fowler, Patterns of Enterprise Application Architecture. Addison-Wesley, 2003.

[9] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, 2nd ed. Addison-Wesley, 2005.

[10] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, 3rd ed. Addison-Wesley, 2013.

[11] J. L. Hellerstein, M. Stonebraker, and J. Hellerstein, Readings in Database Systems, 4th ed. MIT Press, 2009.

[12] W. Stallings, Operating Systems: Internals and Design Principles, 9th ed. Pearson, 2017.

[13] P. P. Chen, "The entity-relationship model—Toward a unified view of data," ACM Transactions on Database Systems, vol. 1, no. 1, pp. 9–36, 1976.

[14] R. C. Martin, Agile Software Development: Principles, Patterns, and Practices. Prentice Hall, 2003.

[15] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object-oriented design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, Jun. 1994.