Software Requirements Specification

for

Bug Tracking System

Version 1.0 approved

Prepared by Pranay Jain

Manipal University Jaipur

29 September 2024

Table of Contents

Revision History

Name	Date	Reason for Changes	Version
—	—	—	1.0

1 Introduction

1.1 Purpose

The purpose of this document is to define the software requirements for the Bug Tracking System. This system provides a centralized platform to manage software bugs effectively. It aims to streamline the process of identifying, recording, tracking, and resolving issues that arise during software development and testing.

The Bug Tracking System is designed for use in software development projects where maintaining high-quality standards is critical. It will cater to development teams, testers, and project managers by facilitating transparency and ensuring that bugs are resolved efficiently.

The scope of this SRS covers all the core features of the Bug Tracking System, such as bug reporting, status tracking, notifications, and user management. It also addresses nonfunctional requirements, such as performance, security, and scalability.

1.2 Document Conventions

This document adheres to the IEEE standards for software requirements specifications. The following conventions are used:

- **Bold Text:** Indicates terms or headings of particular importance.
- Monospace Font: Used to represent code, database queries, or system identifiers.
- Section and requirement numbering: Sections are numbered hierarchically (e.g., 1, 1.1, 1.1.1). Requirements within each feature or module are labeled as REQ-1, REQ-2, etc.

In addition, placeholders like *Pranay Jain* and *Manipal University Jaipur* should be replaced with appropriate values before final submission.

1.3 Intended Audience and Reading Suggestions

The primary audience for this document includes:

- **Developers:** To understand the required features, interfaces, and constraints of the system.
- **Testers:** To design and implement test cases based on the described requirements.
- **Project Managers:** To plan and allocate resources effectively for the development and testing of the system.
- **Stakeholders:** To verify that the system aligns with business objectives and user needs.

The document is structured to allow readers to navigate easily. It is recommended that readers start with the ”Introduction” to gain an overview, then proceed to ”Overall Description” for a high-level understanding. For detailed functional requirements, the ”System Features” section is critical. Nonfunctional and external interface requirements are also essential for technical teams.

1.4 Product Scope

The Bug Tracking System is a software application designed to help organizations efficiently manage the lifecycle of software bugs. Its primary goals include:

- **Bug Reporting:** Enable users to report issues with detailed information such asseverity, category, and reproducibility.
- **Status Tracking:** Allow users to monitor the progress of reported bugs and theirresolution.
- **Notification Management:** Automatically notify relevant stakeholders of bug updates and status changes.
- **User Roles and Access Control:** Provide role-based access to ensure secure andappropriate use of the system.

The system is intended to integrate seamlessly with popular development tools (e.g., Git, Jenkins) and is compatible with multiple environments (e.g., web-based interfaces). Its implementation will contribute to improved software quality, reduced time-to-market, and enhanced team collaboration.

1.5 References

The following references were used in the preparation of this document:

- IEEE Recommended Practice for Software Requirements Specifications (IEEE Std830-1998).
- Existing Bug Tracking Systems for reference, such as JIRA, Bugzilla, and Redmine.
- Organizational guidelines for software development and quality assurance.
- API and documentation of integration tools like Git and Jenkins.

2 Overall Description

2.1 Product Perspective

The Bug Tracking System is a standalone application developed to address common challenges in managing software defects. It provides a comprehensive solution for tracking, prioritizing, and resolving bugs.

This system is part of the broader software development lifecycle and can integrate with tools like:

- **Version Control Systems (e.g., Git):** Linking bug reports to specific commits.
- **CI/CD Pipelines (e.g., Jenkins):** Associating bugs with failed builds or deployments.
- **Email and Messaging Platforms:** For real-time notifications and team collaboration.

The Bug Tracking System can operate independently or be integrated into larger project management suites like JIRA or Trello. The architecture supports modular design, ensuring scalability and flexibility for future enhancements.

A simple high-level diagram is provided to illustrate the system's interactions:

Diagram Placeholder: Overall Architecture of Bug Tracking System.

2.2 Product Functions

The Bug Tracking System delivers the following core functionalities:

1. **Bug Reporting:**
   - Capture bug details such as title, description, severity, and reproducibility steps.
   - Allow users to upload screenshots or log files as evidence.
2. **Bug Categorization and Prioritization:**
   - Categorize bugs by type (e.g., UI, Backend, Performance).
   - Prioritize bugs based on severity (Critical, High, Medium, Low).
3. **Status Management:**
   - Track bug lifecycle states (e.g., New, In Progress, Resolved, Closed).
   - Provide visual indicators for overdue bugs.
4. **User Roles and Permissions:**
   - Role-based access for Administrators, Developers, and Testers.
   - Audit logs to track user actions for accountability.
5. **Search and Filtering:**

- Advanced search for bugs using filters like date, priority, and status.
- Save commonly used searches for quick access.

6. **Notification and Alerts:**

- Email or in-app notifications for assigned tasks or status changes.
- Configurable alert settings for individual users.

2.3 User Classes and Characteristics

The Bug Tracking System is designed to accommodate a diverse range of users, catego-rized as follows:

- **Developers:**
  - Responsible for resolving bugs.
  - Require access to detailed bug descriptions and related logs.
- **Testers:**
  - Log new bugs and verify resolved bugs.
  - Need an intuitive interface for submitting bug reports.
- **Project Managers:**
  - Monitor bug resolution progress.
  - Require dashboard summaries and analytics.
- **End Users (Optional):**
  - Report bugs from the user interface.
  - Access limited to submitting and tracking bugs they reported.

Each user class has unique needs that the system addresses through role-based accesscontrol and customized interfaces.

2.4 Operating Environment

The Bug Tracking System is designed to operate in the following environments:

- **Client Side:**
  - Web browsers: Google Chrome, Mozilla Firefox, Microsoft Edge (latest versions).
  - Device compatibility: Desktop and tablet support with responsive design.
- **Server Side:**
  - Operating Systems: Linux (preferred) or Windows.
  - Database: PostgreSQL or MySQL.

  - Application Server: Node.js or Apache Tomcat.
  - Other Dependencies: JavaScript frameworks (e.g., React, Angular).

- **Network:**
  - Supports HTTP/HTTPS protocols.
  - Minimum bandwidth requirement: 2 Mbps per client.

2.5 Design and Implementation Constraints

Several constraints affect the design and implementation of the Bug Tracking System:

- **Technological Constraints:**
  - Must use PostgreSQL for data storage.
  - Integration with Git and Jenkins using their respective APIs.
- **Regulatory Constraints:**
  - Data encryption for compliance with GDPR and other data privacy regulations.
- **Performance Constraints:**
  - The system must handle 500 concurrent users without latency exceeding 2 seconds.

2.6 User Documentation

The following documentation will be provided to ensure proper use of the Bug Tracking System:

- **User Manual:** Detailed guide for navigating and using the system.
- **Quick Start Guide:** Simplified steps for common tasks like reporting bugs and assigning users.
- **Online Help System:** Context-sensitive help accessible directly within the appli-cation.
- **Video Tutorials:** Short instructional videos covering major features.

Documentation will be delivered in both PDF format and as part of an integrated help system within the application.

2.7 Assumptions and Dependencies

The following assumptions and dependencies have been identified:

- **Assumptions:**
  - Users have basic knowledge of operating web applications.
  - Stable network connectivity is available for accessing the system.
- **Dependencies:**
  - External APIs for integration with Git and Jenkins are reliable and maintained.
  - Open-source libraries and frameworks used in development are compatiblewith future versions.

The project may be affected if these assumptions prove incorrect or dependencies are not met.

3 External Interface Requirements

3.1 User Interfaces

The Bug Tracking System provides a user-friendly interface that facilitates seamless interaction between users and the system. The design adheres to established UI/UX principles to ensure ease of use and accessibility.

- **Web Interface:**
  - **Dashboard:** Displays an overview of active bugs, status summaries, and assigned tasks.
  - **Bug Reporting Form:** Intuitive form with fields such as title, description, severity, category, and attachments (e.g., screenshots or log files).
  - **Search and Filters:** Advanced search bar with customizable filters, includingdate ranges, priority, and status.
  - **Responsive Design:** Optimized for both desktop and tablet screens, ensuringcompatibility across devices.
- **Notification Interface:**
  - Real-time notifications displayed in a dedicated panel.
  - Configurable email alerts for status updates or assigned tasks.
- **Accessibility Features:**
  - Keyboard shortcuts for navigation and common actions.
  - High-contrast mode and screen reader compatibility for visually impaired users.

3.2 Hardware Interfaces

The Bug Tracking System interacts with hardware components for both user-facing devices and server-side infrastructure. The hardware requirements are as follows:

- **Client Devices:**
  - Minimum hardware requirements:
    - Processor: 1.5 GHz dual-core or higher.
    - RAM: 2 GB or more.
    - Storage: At least 500 MB free space for browser cache and temporary files.
  - Supported devices:
    - Desktop computers running Windows, macOS, or Linux.
    - Tablets with Android or iOS.
- **Server Infrastructure:**
  - Hardware specifications for the server hosting the application:
    - Processor: 2 GHz quad-core or higher.
    - RAM: 8 GB minimum (16 GB recommended for high traffic).
    - Storage: SSD with 100 GB free space for logs, databases, and backups.

3.3 Software Interfaces

The Bug Tracking System integrates with various software components to provide a complete bug management solution. These software interfaces include:

- **Operating Systems:**
  - Server-side: Supports Linux (Ubuntu, CentOS) and Windows Server.
  - Client-side: Compatible with Windows, macOS, Android, and iOS.
- **Database Systems:**
  - PostgreSQL 13+ as the primary database.
  - MySQL as an optional alternative.
- **Integration APIs:**
  - GitHub API for linking bugs to commits and pull requests.
  - Jenkins API for associating bugs with failed builds or deployments.
- **Email Servers:**
  - Integration with SMTP servers (e.g., Gmail, Outlook) for sending notifications.
- **Web Frameworks:**
  - Frontend: React or Angular for building the user interface.
  - Backend: Node.js or Django for processing and managing server-side operations.

3.4 Communications Interfaces

The Bug Tracking System requires reliable communication protocols and standards to ensure secure and efficient data exchange between components and users.

- **Network Communication:**
  - HTTPS is used for secure communication between client devices and the server.
  - RESTful APIs facilitate communication between the application and third-party tools.
- **Data Exchange Formats:**
  - JSON is used for API responses and requests.
  - CSV/Excel export for generating bug reports.
- **Messaging Protocols:**
  - Integration with email protocols like SMTP, IMAP, and POP3.
  - Real-time notifications using WebSocket protocols.
- **Security Measures:**
  - TLS 1.2+ for encrypting data in transit.
  - OAuth 2.0 for secure API authentication.

4 System Features

This section describes the primary system features of the Bug Tracking System. Each feature is detailed with its description, priority, stimulus/response sequences, and functional requirements.

4.1 Bug Reporting

4.1.1 Description and Priority

This feature allows users to report bugs with all necessary details. Users can specify the bug's severity, type, description, and attach relevant files like screenshots or logs.

- **Priority:** High
- **Rationale:** Accurate and detailed bug reporting is essential to effective debugging and resolution.

4.1.2 Stimulus/Response Sequences

1. User navigates to the "Report Bug" section.
2. User fills in fields like title, description, severity, and reproducibility steps.
3. User uploads optional attachments.
4. System validates input and saves the bug report to the database.
5. System sends notifications to assigned team members.

4.1.3 Functional Requirements

- REQ-1: Users must be able to enter and save bug details.
- REQ-2: The system must validate required fields before submission.
- REQ-3: Users must be able to upload attachments of up to 10 MB per file.
- REQ-4: Notifications must be triggered upon successful bug submission.

4.2 Bug Categorization and Prioritization

4.2.1 Description and Priority

This feature allows bugs to be categorized by type (e.g., UI, Performance, Security) and prioritized (e.g., Critical, High, Medium, Low) based on their impact.

- **Priority:** High
- **Rationale:** Proper categorization and prioritization help allocate resources effectively.

4.2.2 Stimulus/Response Sequences

1. User selects a category and priority while reporting or editing a bug.
2. System assigns the selected category and priority to the bug.
3. Dashboard updates to reflect bugs sorted by category and priority.

4.2.3 Functional Requirements

- REQ-1: Users must be able to assign categories to bugs during reporting or editing.
- REQ-2: Priority levels must be pre-defined and consistent across the system.
- REQ-3: Dashboard must support sorting and filtering based on category and pri-ority.

4.3 Bug Lifecycle Management

4.3.1 Description and Priority

The system tracks the status of bugs through a lifecycle, including states like New, InProgress, Resolved, and Closed.

- **Priority:** High
- **Rationale:** Efficient tracking ensures bugs are addressed in a timely manner.

4.3.2 Stimulus/Response Sequences

1. User updates a bug’s status to ”In Progress” after starting work on it.
2. Tester changes the status to ”Resolved” after confirming the fix.
3. System archives bugs marked as ”Closed” for over 90 days.

4.3.3 Functional Requirements

- REQ-1: Users must be able to update bug statuses.
- REQ-2: The system must log status changes with timestamps and user details.
- REQ-3: Closed bugs must be archived automatically after a specified period.

4.4 Notification and Alert System

4.4.1 Description and Priority

This feature notifies users about updates, assignments, and deadlines related to bugs. Alerts are delivered via email and in-app notifications.

- **Priority:** Medium
- **Rationale:** Notifications improve team collaboration and ensure timely responses.

4.4.2 Stimulus/Response Sequences

1. System sends a notification when a bug is assigned to a user.
2. User receives a reminder notification for overdue bugs.
3. System logs notification delivery status for auditing purposes.

4.4.3 Functional Requirements

- REQ-1: Notifications must be configurable based on user preferences.
- REQ-2: Email notifications must include bug details and relevant links.
- REQ-3: In-app notifications must display in real time.

4.5 Advanced Search and Filtering

4.5.1 Description and Priority

This feature allows users to search and filter bugs based on various criteria such as status, category, priority, and assigned user.

- **Priority:** Medium
- **Rationale:** Advanced search capabilities improve efficiency in large projects.

4.5.2 Stimulus/Response Sequences

1. User inputs search criteria into the search bar.
2. System retrieves matching results and displays them in a table view.
3. User saves the search criteria as a custom filter for future use.

4.5.3 Functional Requirements

- REQ-1: Users must be able to filter bugs by multiple criteria simultaneously.
- REQ-2: The system must display search results within 2 seconds for up to 1000 records.
- REQ-3: Saved search filters must be editable and delete-able.

4.6 Analytics and Reporting

4.6.1 Description and Priority

This feature provides visual analytics and reports on bug trends, resolution times, and team performance. Users can export reports for external analysis.

- **Priority:** Low
- **Rationale:** Analytics help stakeholders evaluate team productivity and system health.

4.6.2 Stimulus/Response Sequences

1. User selects a date range for the analytics dashboard.
2. System generates charts and graphs showing bug resolution trends.
3. User exports the report in CSV or PDF format.

4.6.3 Functional Requirements

- REQ-1: The system must generate visual reports, including pie charts and bargraphs.
- REQ-2: Reports must be exportable in CSV and PDF formats.
- REQ-3: Data refresh must occur in real time or upon user request.

5 Other Nonfunctional Requirements

This section describes the nonfunctional requirements that ensure the Bug Tracking Sys-tem meets performance, safety, security, quality, and business constraints.

5.1 Performance Requirements

The Bug Tracking System must be able to handle high volumes of concurrent users and maintain acceptable response times under various operational conditions.

- The system must support up to 500 concurrent users without performance degradation.
- Average response time for critical operations (e.g., submitting a bug, updating status) should not exceed 2 seconds under normal load conditions.
- The system must process and display search results for up to 1000 records within2 seconds.
- Bulk operations, such as exporting large datasets (e.g., 10,000 records), should complete within 10 seconds.
- The system must be able to scale horizontally by adding servers to accommodateincreasing user loads.
- Real-time notifications must be delivered within 5 seconds of the triggering event.

5.2 Safety Requirements

The Bug Tracking System must safeguard against potential harm or data loss, ensuringsystem reliability and stability during operation.

- Regular backups of the database must be performed daily, with the ability to restore data to the state of any backup within 15 minutes.
- The system must ensure data integrity by validating all input fields to prevent invalid data from being stored.
- User actions, such as deleting or archiving bugs, must include a confirmation prompt to avoid accidental changes.
- The system must log critical errors and send alerts to the administrator for resolution within 30 minutes of detection.
- Failover mechanisms must be implemented to ensure minimal downtime in the event of hardware or software failures.

5.3 Security Requirements

The Bug Tracking System must adhere to strict security standards to protect user data and ensure system integrity.

- All communication between the client and the server must be encrypted using HTTPS with TLS 1.2 or higher.
- User authentication must be enforced using strong password policies:
  - Minimum password length: 8 characters.
  - Passwords must include at least one uppercase letter, one lowercase letter, one number, and one special character.
- Multi-factor authentication (MFA) must be supported for users with administrative privileges.
- Role-based access control (RBAC) must be implemented to restrict access to sen-sitive data and operations based on user roles.
- All user actions (e.g., bug updates, deletions) must be logged with timestamps and user IDs for auditing purposes.
- The system must prevent SQL injection, cross-site scripting (XSS), and other com-mon vulnerabilities through input sanitization and secure coding practices.

5.4 Software Quality Attributes

The Bug Tracking System must demonstrate high standards of quality, ensuring usability, maintainability, reliability, and portability.

- **Usability:**
  - The user interface must adhere to modern UI/UX standards for accessibility and ease of navigation.
  - The system must support localization for different languages and regions.
- **Maintainability:**
  - The codebase must follow clean coding practices with proper documentation to facilitate easy maintenance and upgrades.
  - Modular architecture must be adopted to allow independent updates to fea-tures or components.
- **Reliability:**
  - The system must have an uptime of at least 99.9% per month.
  - Automatic monitoring tools must detect and alert for errors or anomalieswithin 5 minutes of occurrence.
- **Portability:**

- The system must run on major server operating systems, including Linux andWindows Server.
- The client-side application must be compatible with the latest versions ofmajor web browsers (e.g., Chrome, Firefox, Edge).

5.5 Business Rules

The Bug Tracking System must adhere to the organization's operational principles and business constraints.

- Only users with the appropriate roles (e.g., Administrator, Tester) can create, edit, or delete bug records.
- Critical bugs (priority: Critical) must be addressed within 24 hours of reporting.
- Users must update the status of assigned bugs daily to maintain progress tracking.
- Notifications must only be sent during working hours (e.g., 9 AM to 6 PM) to avoid unnecessary disturbances.
- Data retention policies must comply with regulatory requirements, such as GDPR, which mandates user data deletion upon request.
- Audit logs must be retained for at least one year for compliance and auditing purposes.

6 Other Requirements

This section outlines additional requirements that are not addressed elsewhere in this SRS, including database requirements, internationalization, legal compliance, reuse ob- jectives, and other pertinent considerations.

6.1 Database Requirements

The Bug Tracking System relies on a robust database to store and manage bug-relateddata securely and efficiently.

- **Database Technology:**
  - PostgreSQL (preferred) or MySQL as the relational database managementsystem.
- **Data Storage:**
  - Each bug entry must include fields for title, description, severity, status, assigned user, timestamps, and attachments.
  - File attachments such as screenshots and logs must be stored in a dedicatedfile system with database references for metadata.
- **Database Performance:**
  - Indexing must be implemented on frequently queried fields, such as status, priority, and assigned user, to optimize search operations.
  - Support for database partitioning to handle large datasets efficiently.
- **Backup and Recovery:**
  - Full database backups must be taken daily, with incremental backups performed hourly.
  - Recovery procedures must ensure restoration within 15 minutes of failure.
- **Retention Policy:**
  - Archived bug records must be retained for a minimum of five years.
  - Older records must be moved to cold storage but remain accessible through the system.

6.2 Internationalization Requirements

The system must support users from various regions by providing internationalization features.

- **Language Support:**
  - The system must be available in at least three languages: English, Spanish, and French.

  - Language preferences should be configurable per user.

- **Date, Time, and Currency Formatting:**
  - All dates and times must follow the ISO 8601 standard by default but allow user-specific regional formats (e.g., DD/MM/YYYY or MM/DD/YYYY).

- **Character Encoding:**
  - All text fields must support UTF-8 encoding to accommodate special charac-ters and scripts.

6.3 Legal and Regulatory Requirements

The Bug Tracking System must comply with applicable laws and regulations to ensureits lawful use and operation.

- **Data Privacy:**
  - The system must comply with GDPR, CCPA, or other applicable data pro-tection regulations.
  - User data must be encrypted both in transit and at rest.
  - Users must have the ability to request data deletion or anonymization as per regulatory requirements.

- **Accessibility Standards:**
  - The system must conform to WCAG 2.1 AA accessibility standards to ensureusability for individuals with disabilities.

- **Audit and Compliance:**
  - All system logs must be retained for a minimum of one year to meet compliance and audit requirements.
  - Regular penetration testing must be conducted to ensure security compliance.

6.4 Reuse Objectives

To reduce development time and cost, the Bug Tracking System must incorporate reusable components wherever feasible.

- **Code Reusability:**
  - Modular design principles must be employed to facilitate the reuse of compo-nents across different projects or modules.
  - Common utilities, such as authentication modules and database access layers,must be abstracted for reuse.

- **Open Source Libraries:**

  - The system must leverage open-source libraries and frameworks for features such as email notifications, UI components, and database ORM (Object Relational Mapping).

- **Extensibility:**
  - The architecture must allow new features to be added without significant redesign.
  - APIs must be designed for integration with other tools or systems.

6.5 Disaster Recovery and Business Continuity Requirements

The system must ensure reliability and availability in the event of disasters or unexpected failures.

- **High Availability:**
  - The system must be designed to achieve 99.9% uptime per month.
- **Disaster Recovery Plan:**
  - A comprehensive disaster recovery plan must be in place to address scenarios such as server crashes or data corruption.
  - Secondary backup servers must be available in geographically distant locationsto prevent data loss due to regional disasters.
- **Incident Response:**
  - A response team must be notified automatically in the event of a critical failure, and resolution efforts must commence within 30 minutes.

6.6 Other Considerations

- **System Updates:**
  - Updates must be deployed during non-peak hours to minimize disruptions.
  - Rollback mechanisms must be implemented to revert changes if an update fails.
- **Scalability:**
  - The system must scale both horizontally and vertically to accommodate growing user demands and larger datasets.

Appendix A: Glossary

Term	Definition
Database	Collection of all the information monitored by this system.
Software Requirements Specification	A document that completely describes all the functions of a proposed system and the constraints under which it must operate. For example, this document
Bug Lifecycle	The series of states a bug goes through, from identification to resolution and closure.
GDPR (General Data Protection Regulation)	A legal framework that sets guidelines for the collection and processing of personal data of individuals within the European Union.
JSON (JavaScript Object Notation)	A lightweight data interchange format used for communication between the client and the server.
SSL/TLS (Secure Socket Layer/ Transport Layer Security)	Protocols for encrypting data in transit to ensure secure communication between the client and the server.
Test Case	A set of conditions or steps used to verify the functionality of a specific feature or system behavior
UTF-8 (Unicode Transformation Format-8)	A character encoding standard used to represent text in computers and systems.
WebSocket	A protocol enabling two-way communication between the client and server for real-time updates.
Stakeholder	Any person with an interest in the project who is not a developer
Administrator	person responsible for carrying out the administration of a business or organization.
Customer	A member listed in the database and purchased a system and the system resulted in an error.
Bug	A mistake, misconception, or misunderstanding on the part of a software developer.
staff	all the people employed by a particular organization.
User	Admin or Staff or Customer

Appendix B: Analysis Models

LAB-2

B1. Cost Estimation (COCOMO) model for the Bug Tracking System:

Effort Estimation Using COCOMO

The **COCOMO** (Constructive Cost Model) formula for **Effort** is:

$$\text{Effort (Person-Months)} = a \times (\text{KLOC})^{b}$$

Where:

- **a**: Multiplicative constant (for semi-detached, **a=3.0**)
- **b**: Exponent constant (for semi-detached, **b=1.12**)
- **KLOC:** Estimated thousands of lines of code (for this project, **KLOC = 10**)

$$\text{Effort} = 3.0 \times (10)^{1.12} = \underline{\textbf{31.62 Person-Months}}$$

Development Time

The formula for Development Time (in months) is:

$$\text{Time (Months)} = c \times (\text{Effort})^{d}$$

Where:

- **c=2.5, d=0.35** (constants for semi-detached projects).

$$\text{Time} = 2.5 \times (31.62)^{0.35} = \underline{\textbf{8.6 Months}}$$

Cost Estimation

The cost of development is calculated as:

$$\text{Cost (₹)} = \text{Effort (Person-Months)} \times \text{Monthly Rate (₹/Person)}$$

For simplicity, we'll assume the adjusted team composition:

- 1 Developer: ₹40,000/month
- 1 Tester: ₹30,000/month
- Duration: Limited to 3 months (streamlined).

$$\textbf{Developer Cost} = 1\text{Developer} \times ₹40{,}000 \times 3 = ₹1{,}20{,}000$$

$$\textbf{Tester Cost} = 1\text{Tester} \times ₹30{,}000 \times 3 = ₹90{,}000$$

Detailed Breakdown:

Category	Description	Estimated Cost (₹)
Development Costs	Salaries for development team (1 Developer + 1 Tester). - Developer (₹40,000/month × 3 months) - Tester (₹30,000/month × 3 months)	₹1,20,000 + ₹90,000 = ₹2,10,000
Infrastructure Costs	Cloud hosting and basic infrastructure costs. - Cloud hosting (e.g., AWS, Azure): ₹7,500/month × 3 months	₹7,500 × 3 = 22,500
Software and License Costs	Costs for any paid tools/licenses (assumes mostly open-source tools). - Database: PostgreSQL (open-source, ₹0 cost assumed) - Testing tools: Paid tools (if required) or open-source (e.g., Selenium, Postman)	₹10,000
Security and Compliance Costs	Expenses for securing the system and meeting legal compliance standards. - SSL/TLS certificate for secure communication: ₹5,000/year - Penetration testing by a third-party vendor: ₹15,000	₹5,000 + ₹15,000 = ₹20,000
Maintenance and Support Costs	Post-launch support (1 Developer for 2 months). - Developer or part-time support personnel for bug fixes: ₹15,000	₹15,000 × 1 = ₹15,000
Miscellaneous Costs	Includes minor unplanned costs like subscriptions, internet, or electricity. - Internet and electricity for development - Minor unplanned purchases	₹5,000
Total Cost	Total estimated cost of the project.	₹2,00,000

Final Calculation Summary:

1. Development Costs:
    - Developer: ₹1,20,000
    - Tester: ₹90,000
2. Infrastructure Costs:
    - Hosting: ₹22,500
3. Software and License Costs:
    - Assumed: ₹10,000
4. Security and Compliance Costs:
    - SSL + Penetration Testing: ₹20,000
5. Maintenance and Support Costs:
    - ₹15,000 (Post-launch support for 1 developer).
6. Miscellaneous Costs:
    - ₹5,000

Total: ₹2,00,000 (Aligned with the budget constraint).

So, finally, we can say that our project of BUG TRACKING SYSTEM requires approximately 2-Lakhs to get completed, assuming all costs and budgets.

LAB-3

B2. Use Case Diagram

Here is a Use Case Diagram for a Bug Tracking System. It depicts three main actors—Admin, Staff, and Customer (including New Customer)—and their interactions with various system functionalities.

Actors:

1. **Admin**: The system administrator who manages projects, employees, bugs, and assigns tasks.
2. **Staff**: Employees or staff members who interact with bugs assigned to them.
3. **Customer**: Users who interact with the system to report bugs. New customers first need to register.

Use Cases:

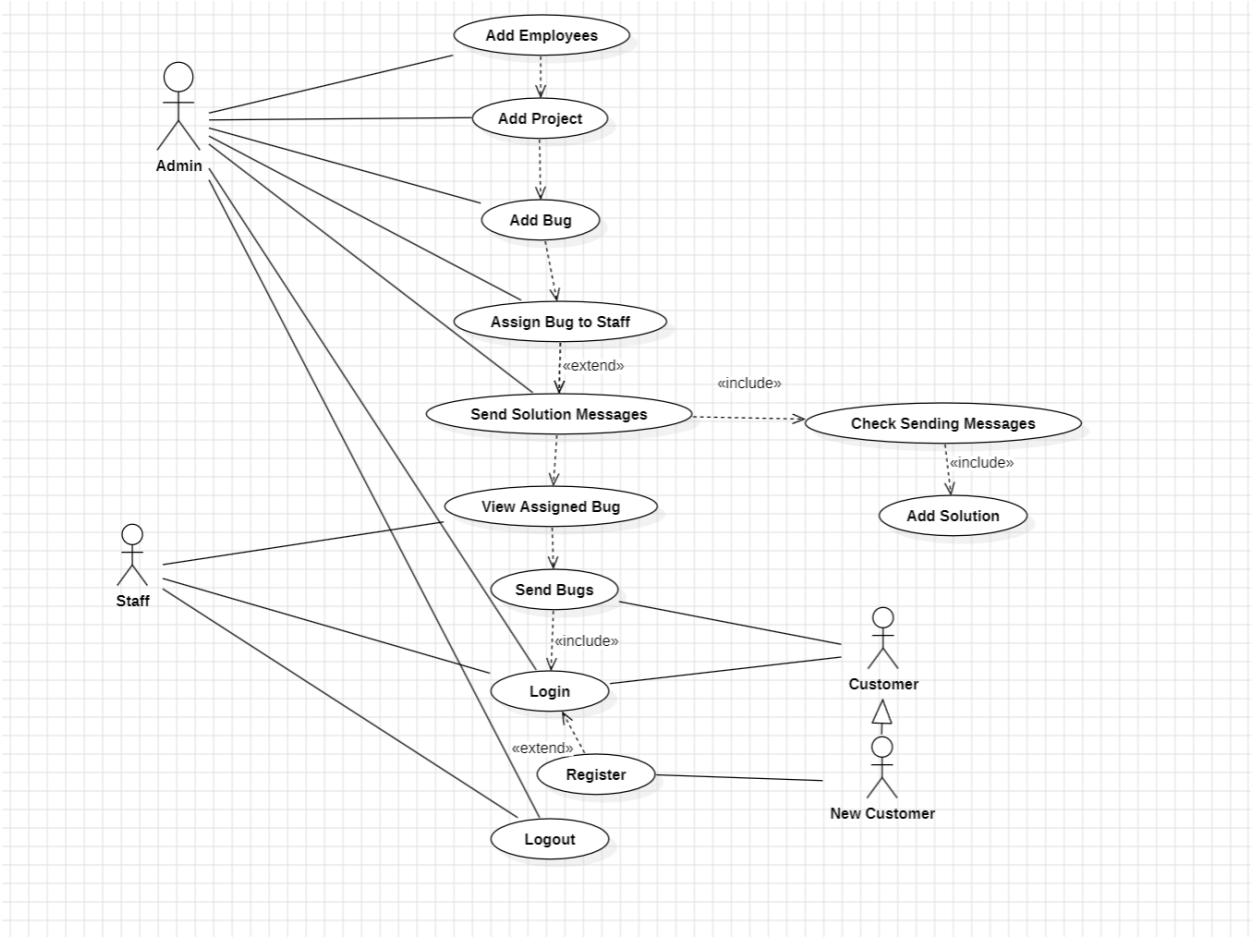
- **Admin Use Cases**:
    - **Add Employees**: The admin can add new employees to the system.
    - **Add Project**: The admin can add a new project.
    - **Add Bug**: Bugs can be logged into the system.
    - **Assign Bug to Staff**: The admin assigns reported bugs to staff members.
    - **Send Solution Messages**: Admin can send solution messages to customers or staff.
    - **View Assigned Bug**: Admin views the details of the bugs assigned to staff members.
    - **Login/Logout/Register**: Basic authentication actions for all users, extending to Admin.
- **Staff Use Cases**:
    - **View Assigned Bug**: Staff can view bugs that have been assigned to them by the admin.
    - **Send Bugs**: Staff can report or escalate bugs.
    - **Login/Logout/Register**: Basic system access functionalities for staff.
- **Customer Use Cases**:
    - **Send Bugs**: Customers can report bugs in the system.
    - **Login/Register/Logout**: New customers register, and all customers can log in and log out.

Relationships:

- **Includes**: Several use cases are connected with "include" relationships, indicating dependencies:
    - **Send Solution Messages** includes **Check Sending Messages** and **Add Solution**.
    - **Send Bugs** includes **Login** (indicating that the user must log in to send bugs).

- **Extends**: Some use cases extend others:
    - **Assign Bug to Staff** extends **Add Bug**.
    - **Register** extends **Login**.

This diagram outlines the major interactions in the bug tracking system, focusing on how admins manage bugs, how staff interact with their assignments, and how customers report issues.



(Use case diagram for BTS)

LAB-4

B3. Activity and State Diagram

1. Activity Diagram

This activity diagram illustrates the process flow for an Admin using a Bug Tracking System. It outlines the steps the admin takes to log in, manage bugs, and send solutions to customers.

Flow Overview:

1. **Admin Accesses the Website**:
    - The process begins when the admin accesses the website.
2. **Login Process**:
    - The admin is prompted to log in.
    - They enter their username and password.
    - Upon submission, if the credentials are valid, the admin is granted access to the main page.
3. **Admin Main Page**:
    - Once logged in, the admin is presented with multiple options:
        - **Add Bug**:
            - The admin selects the “Add Bug” option.
            - They enter all relevant data about the bug.
            - After submission, the system provides a confirmation message.
        - **Assign Bug**:
            - The admin selects “Assign Bug” from the main page.
            - They enter the ticket number corresponding to the bug.
            - They proceed by selecting “Send,” and the system confirms the assignment.
4. **Sending Solutions**:
    - The admin can also select the “Send Solution” option.
    - They enter the bug’s ticket number.
    - The system retrieves the customer’s email based on the ticket number.
    - The admin then sends a message with the solution to the customer.
    - A confirmation message is received from the system, marking the completion of this task.

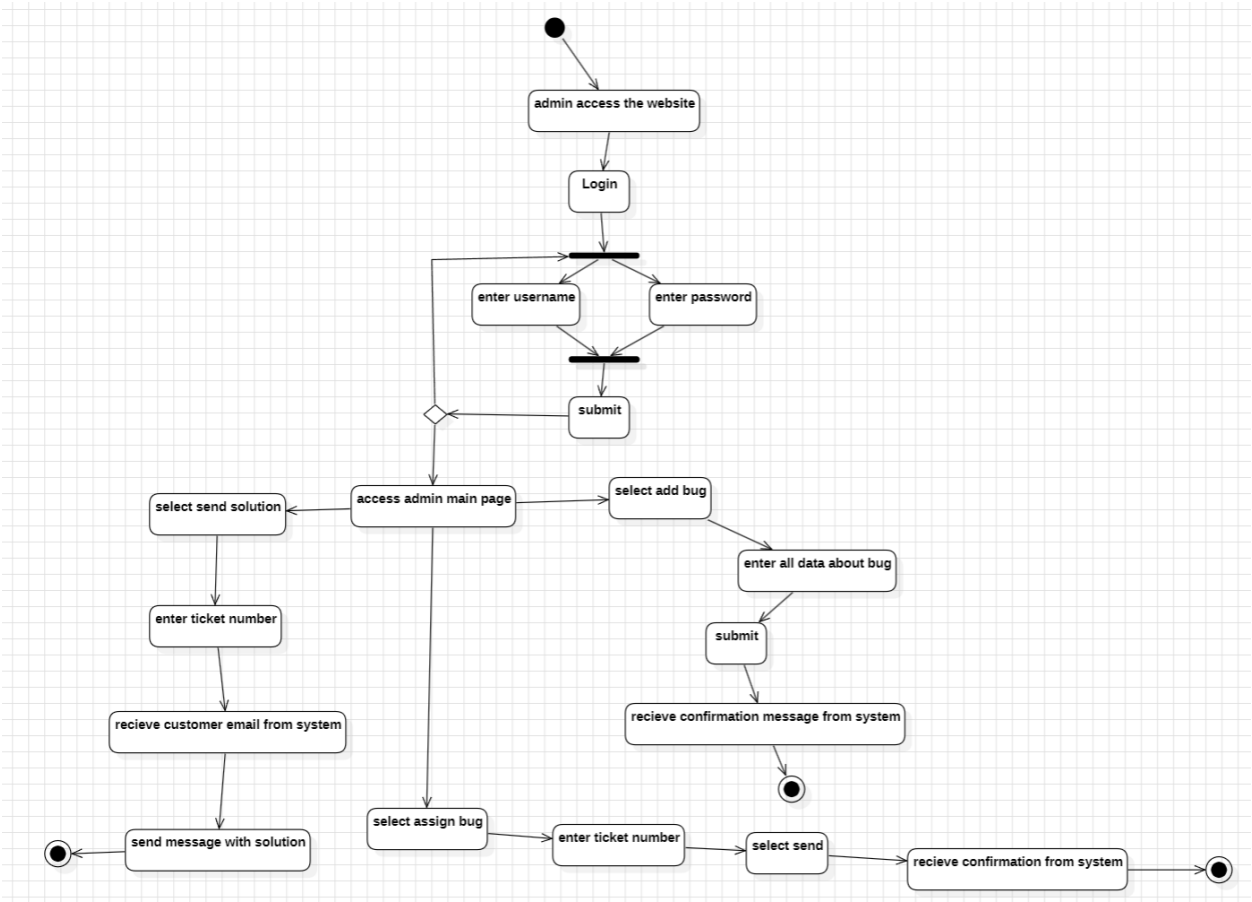
Decision Points:

- A decision point is present after the login submission to check if the admin’s credentials are correct. If correct, they proceed to the main page; otherwise, the login process repeats.

End States:

- The flow ends after the admin receives confirmation from the system, whether for bug assignment or solution delivery.

Summary:
This diagram represents the admin's activities in managing bugs and solutions. The key processes involve logging in, adding or assigning bugs, and sending solutions to customers, with the system providing confirmations at each stage.



(**ACTIVITY DIAGRAM for BTS**)

2. State Diagram:

The **State Diagram** for the Bug Tracking System represents the lifecycle of a bug. It captures the various states a bug can go through, the events or conditions that trigger transitions between states, and the final outcome of the process.

Initial and Final States

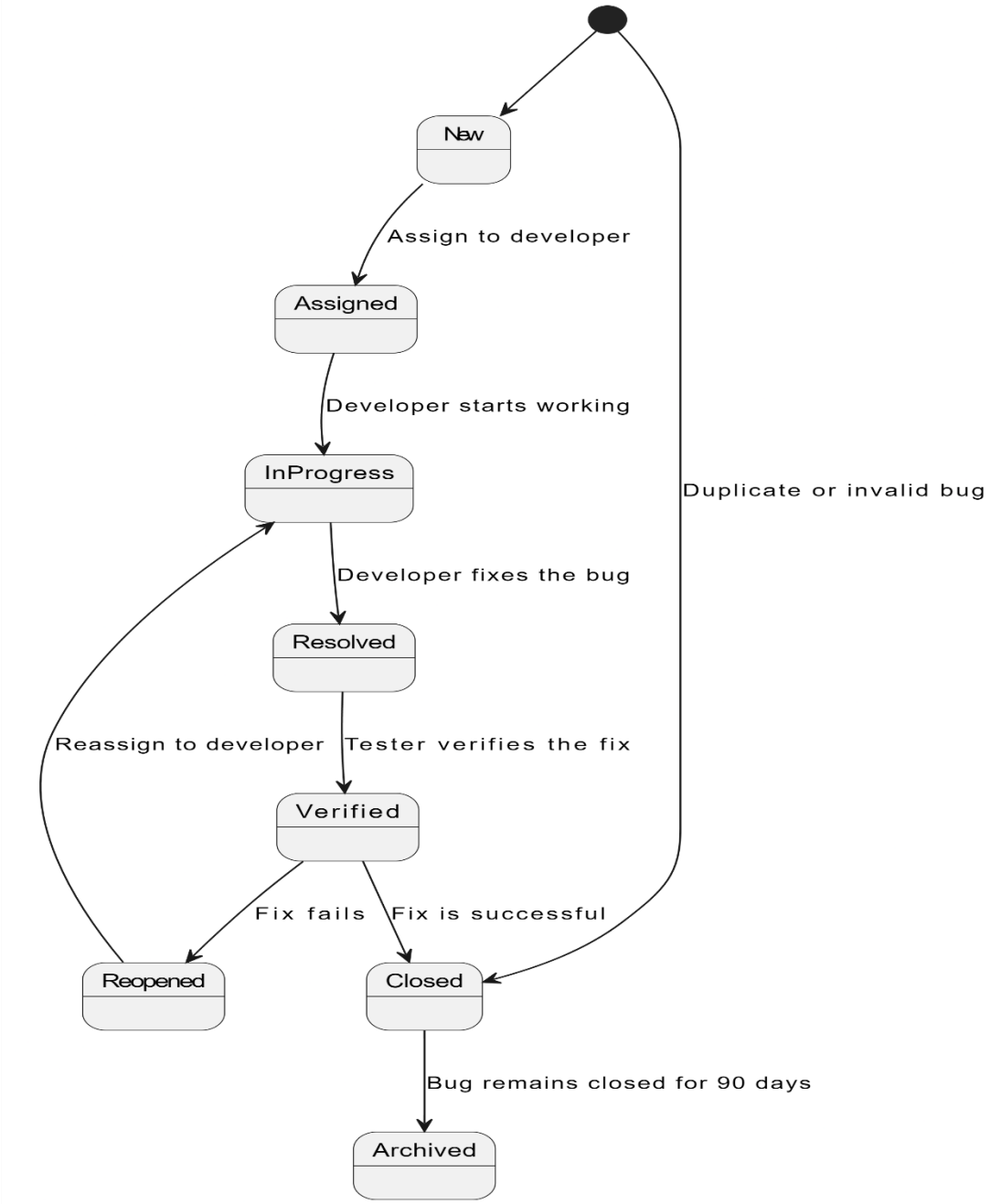
1. **Initial State**:
    - Represented by ● leading to the New state.
    - Indicates that the lifecycle begins when a bug is created.
2. **Final State**:
    - Bugs marked as Archived are considered in their final state.

Transitions Explanation

From New to Assigned:

- Triggered by assigning the bug to a developer.

2. **From Assigned to InProgress**:
    - Triggered when the developer begins working on the bug.
3. **From InProgress to Resolved**:
    - Triggered when the developer marks the bug as fixed.
4. **From Resolved to Verified**:
    - Triggered when the tester tests the fix.
5. **From Verified to Closed**:
    - Triggered when the tester confirms the fix is successful.
6. **From Verified to Reopened**:
    - Triggered when the fix is found to be incomplete or incorrect.
7. **From Reopened to InProgress**:
    - Triggered when the bug is reassigned to a developer for further fixing.
8. **From Closed to Archived**:
    - Triggered automatically after 90 days of inactivity in the Closed state.
9. **From New to Closed** (special case):
    - For duplicate or invalid bugs, the bug is directly marked as Closed.



(STATE DIAGRAM for BTS)

LAB-5

B4. ER Diagram

This ER (Entity-Relationship) Diagram for a Bug Tracking System shows the relationships between various entities involved in the system, focusing on bug tracking, user roles, and project management.

Entities and Attributes:

1. **Customer**:
    - **Attributes**: username, password, projectName, Email, Name
    - **Relationships**:
        - Customers **own** projects.
        - They **send** bugs to the system via the Requested Bugs entity.
2. **Project**:
    - **Attributes**: projectName, details, categoryStaff
    - **Relationships**:
        - Projects are **owned** by customers.
        - Projects **generate** bugs.
3. **Requested Bugs**:
    - **Attributes**: projectName, errorCategory, errorDetails, status
    - **Relationships**:
        - Linked with the Customer entity (customer reports bugs).
        - Associated with the project and its details.
4. **Employee**:
    - **Attributes**: Name, Username, Password, Email, Category
    - **Relationships**:
        - Employees **consist of** different categories of staff.
        - Employees are responsible for **solving** bugs.
5. **Staff**:
    - **Attributes**: Category
    - **Relationships**:
        - Staff members **receive** the project category and details.
        - Consists of specific categories, and assigned to projects.
6. **Bugs**:
    - **Attributes**: TicketNum, projectName, errorCategory, errorDetails, status, assignedToStaff, solvedByEmployee, solution
    - **Relationships**:
        - Bugs are **assigned** to staff members for resolution.

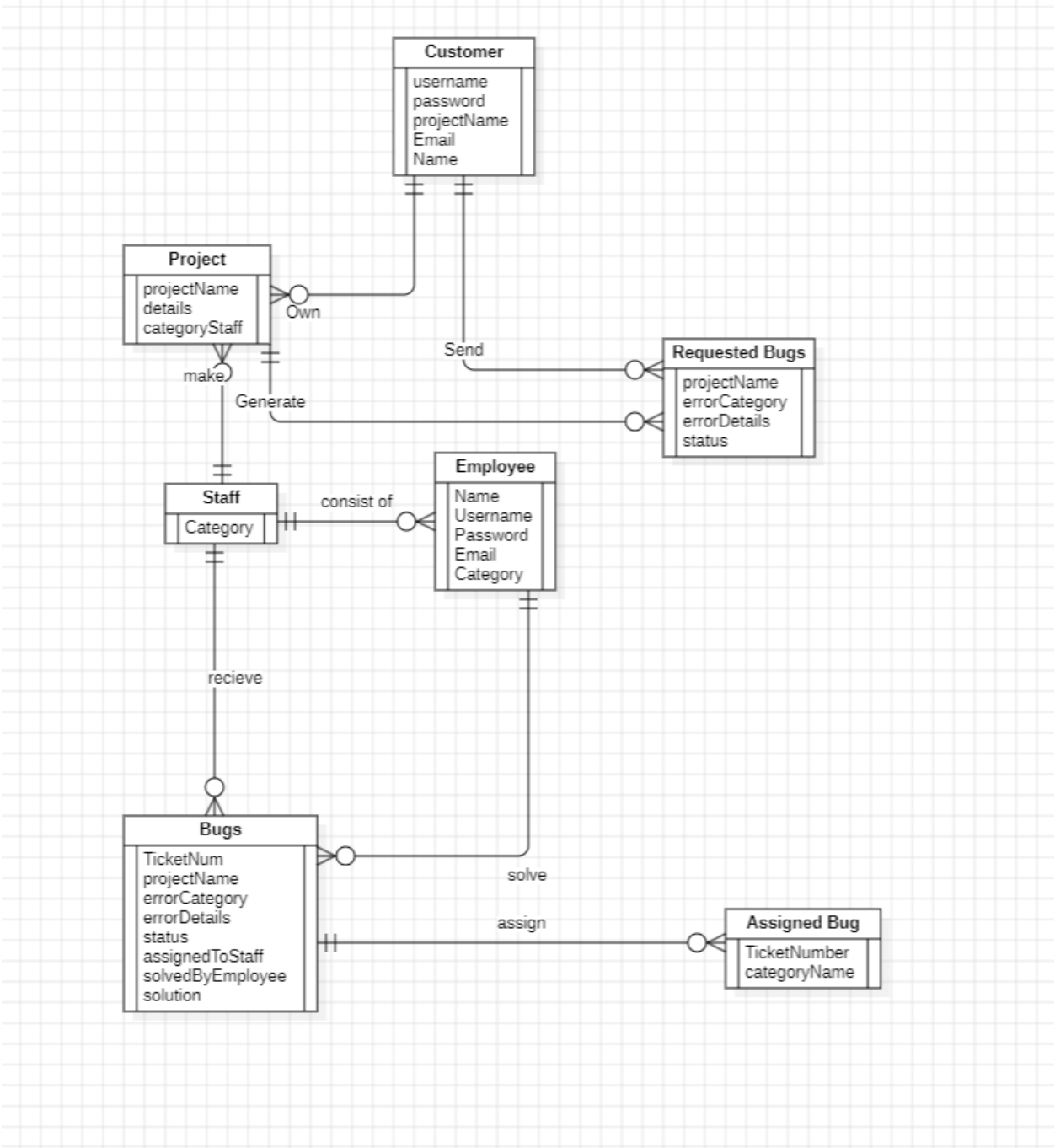
        - Bugs are **solved** by employees.
        - Bugs are related to projects via their projectName.

7. **Assigned Bug**:
    - **Attributes**: TicketNumber, categoryName
    - **Relationships**:
        - Bugs are **assigned** to a staff member based on their category.

Summary:

- **Customer**: Customers report bugs related to their projects. They own the projects, which generate bugs.
- **Project**: Projects are related to bugs and consist of categories.
- **Requested Bugs**: These represent bug reports sent by customers, detailing the error category and status.
- **Employee and Staff**: Employees solve the bugs. The staff consists of different categories, and bugs are assigned based on these categories.
- **Bugs**: Central to the system, bugs are tracked with ticket numbers, and they have details such as error categories and statuses. They are assigned to staff members and solved by employees.
- **Assigned Bug**: Once a bug is reported, it is assigned to an appropriate staff member based on its category.

This diagram outlines the key relationships between customers, employees, staff, projects, and bugs, emphasizing how bugs are reported, assigned, and resolved.



(ER Diagram for BTS)

LAB-6

B5. Class and Object Diagram

1. Class Diagram

The class diagram presents a conceptual model of a bug tracking system, illustrating the relationships and interactions between various entities involved in the process.

Key Entities:

- **Customer:** Represents users who report bugs or issues within the system.
- **User:** A broader category encompassing both customers and staff members.
- **Staff:** Employees responsible for handling and resolving bug reports.
- **Bug:** Represents a reported issue or defect within the system.
- **Admin:** Has administrative privileges to manage the system, including adding/updating categories, employees, and projects.
- **Database:** Handles data storage and retrieval for the system.

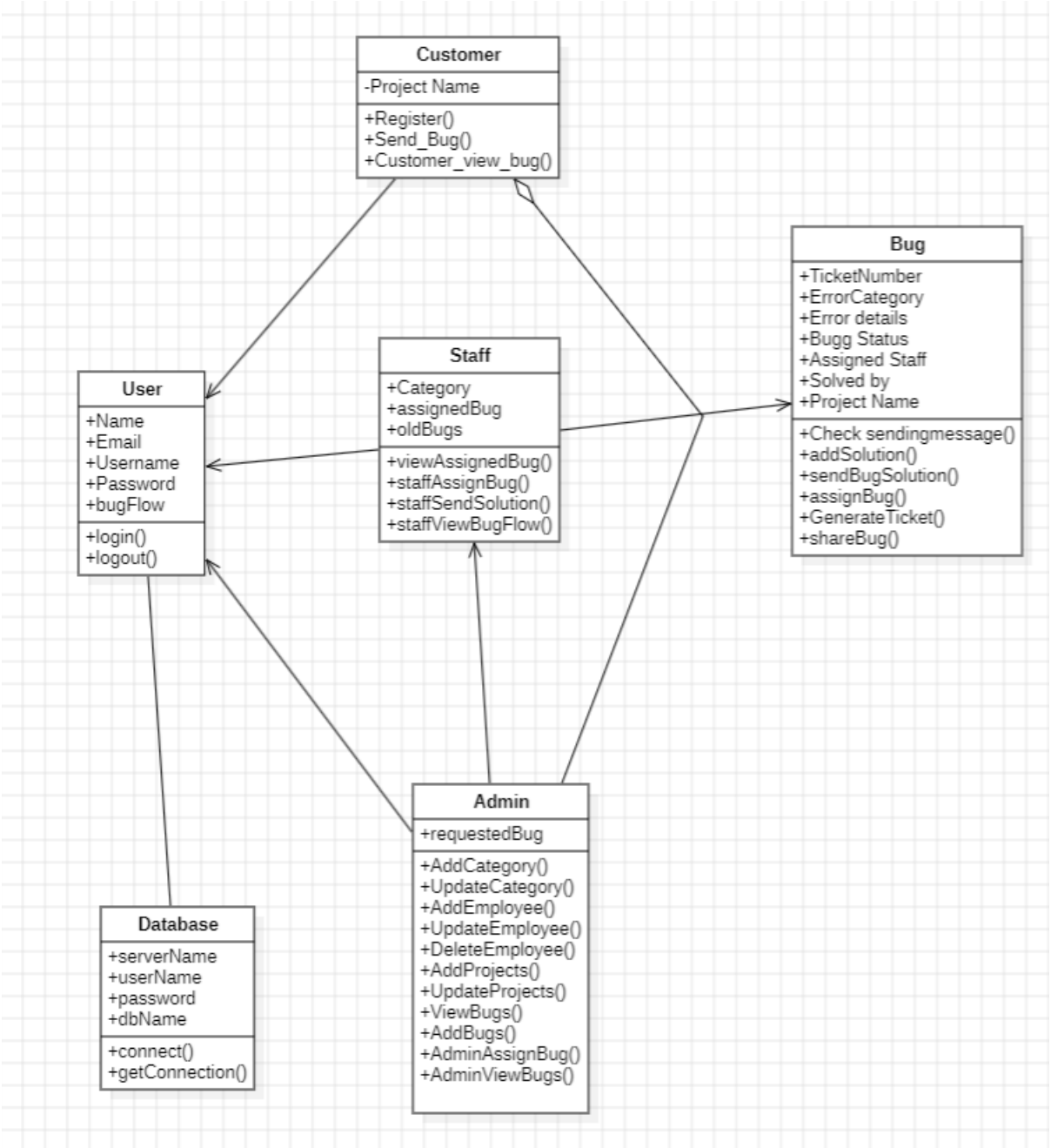
Relationships and Interactions:

- **Customer-Bug:** Customers can report bugs, creating new bug instances.
- **Bug-Staff:** Staff members are assigned to handle bugs, updating their status and providing solutions.
- **Staff-Admin:** Admins can assign bugs to staff members and view their progress.
- **Admin-Database:** Admins interact with the database to manage system data, such as adding/updating employees, categories, and projects.
- **User-Database:** Users (customers and staff) interact with the database to access and update information.

Key Operations:

- **Customer:** Report bugs, view bug status.
- **Staff:** Assign bugs, view assigned bugs, send solutions, check if a message has been sent.
- **Admin:** Add/update categories, employees, projects, view bugs, assign bugs.
- **Database:** Connect to the database, get a connection, perform CRUD operations (create, read, update, delete) on various entities.

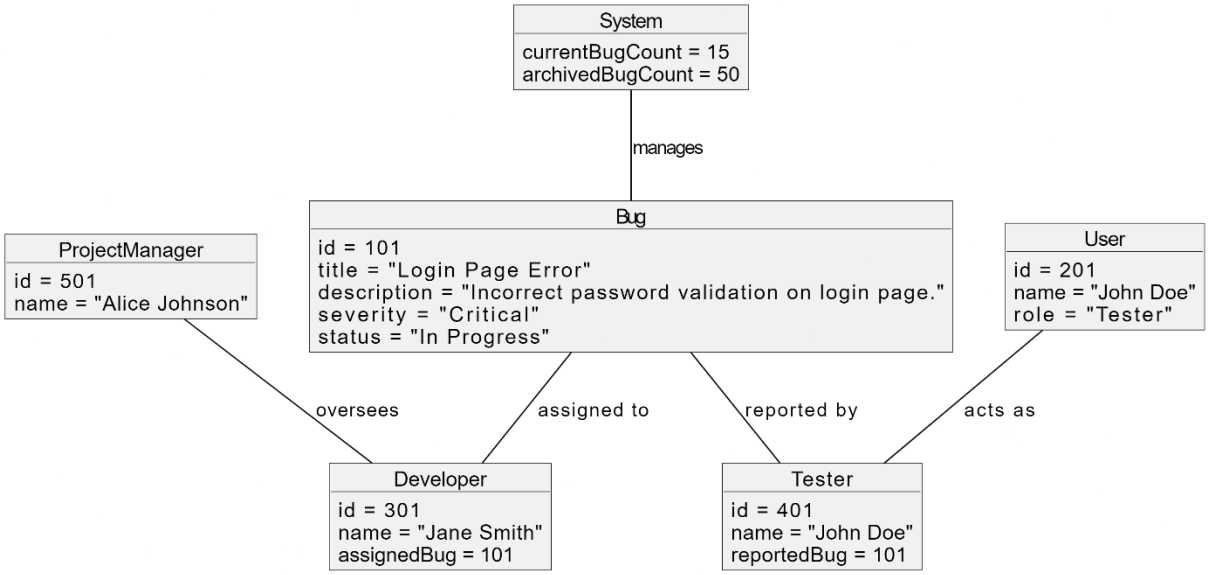
Overall, the class diagram outlines a well-structured system for tracking, managing, and resolving bugs within an application or system. It provides a clear visualization of the entities involved and their interactions, facilitating understanding and development of the bug tracking system.



(**Class Diagram for BTS**)

2. Object Diagram

1. **Objects Represented**:
    - **Bug**: Represents a bug with attributes such as id, title, description, severity, and status.
    - **User**: Represents a user of the system with attributes like id, name, and role.
    - **Developer**: Represents the developer assigned to fix the bug.
    - **Tester**: Represents the tester who reported the bug.
    - **ProjectManager**: Represents the project manager overseeing the system.
    - **System**: Represents the Bug Tracking System itself, with attributes like the current bug count and archived bug count.
2. **Relationships**:
    - A User acts as a Tester and can report bugs.
    - A Bug is assigned to a Developer.
    - A ProjectManager oversees the work of Developer.
    - The System manages bugs, including current and archived ones.
3. **Example Objects**:
    - The Bug object has specific details: ID = 101, title = “Login Page Error”, severity = “Critical”, and status = “In Progress”.
    - The Developer object (Jane Smith) is assigned to fix the bug with ID = 101.



(**Object Diagram for BTS**)

LAB-7

B6. Sequence Diagram

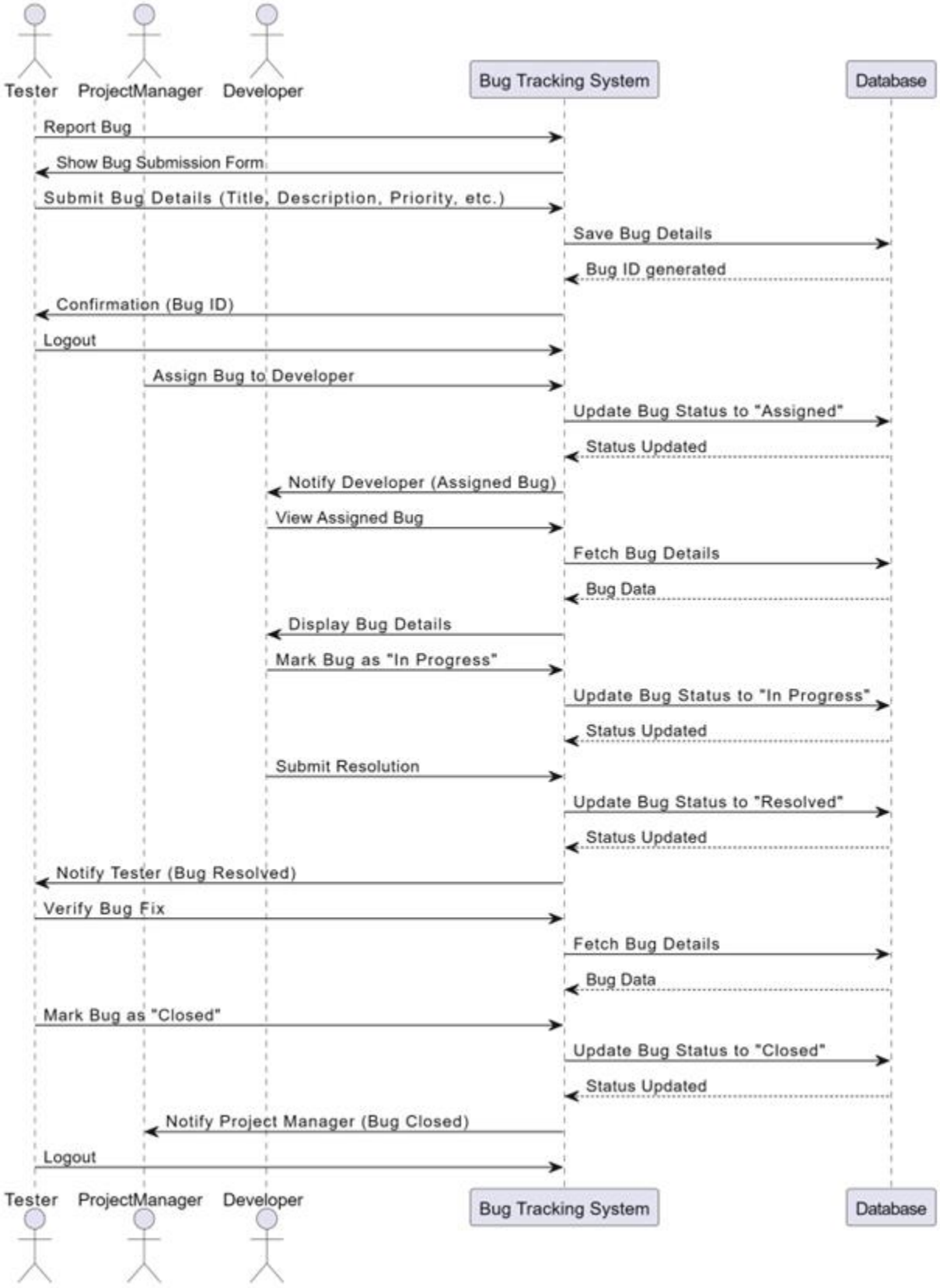
This sequence diagram represents the flow of interactions in a **Bug Tracking System (BTS)** involving three main actors: **Tester**, **Project Manager**, and **Developer**, along with the system itself and the **Database**.

Key Steps:

1. **Bug Reporting (Tester)**:
    - The **Tester** initiates the process by reporting a bug, which triggers the system to display a form for bug submission.
    - After submitting details like the title, description, and priority, the system saves the bug details in the database and generates a unique Bug ID.
    - The system then confirms the bug submission by providing the Tester with the Bug ID.
2. **Bug Assignment (Project Manager)**:
    - The **Project Manager** assigns the reported bug to a **Developer**.
    - The system updates the bug's status to "Assigned" in the database and notifies the Developer about the new assignment.
3. **Bug Fixing (Developer)**:
    - The **Developer** views the assigned bug by fetching the bug details from the database.
    - Once the Developer begins working on the bug, they mark it as "In Progress", and the system updates this status in the database.
    - After resolving the issue, the Developer submits the resolution, and the system changes the status of the bug to "Resolved" in the database.
4. **Bug Verification and Closure (Tester)**:
    - The **Tester** is notified of the resolution and verifies the bug fix by reviewing the details again from the database.
    - If the fix is satisfactory, the Tester marks the bug as "Closed", and the system updates the status in the database accordingly.
    - Finally, the system notifies the **Project Manager** that the bug has been closed.

Interactions:

- Each actor interacts with the **Bug Tracking System**, which, in turn, communicates with the **Database** to update or retrieve bug status and details.



(**SEQUENCE DIAGRAM of the BTS**)

LAB-8

B7. Data Flow Diagram

A **Data Flow Diagram (DFD)** is used to visually represent the flow of data through a system. It highlights the system's processes, external entities, data inputs, outputs, and storage points. For the **Bug Tracking System**, the DFD captures the interactions between users, administrators, developers, and the processes within the system.

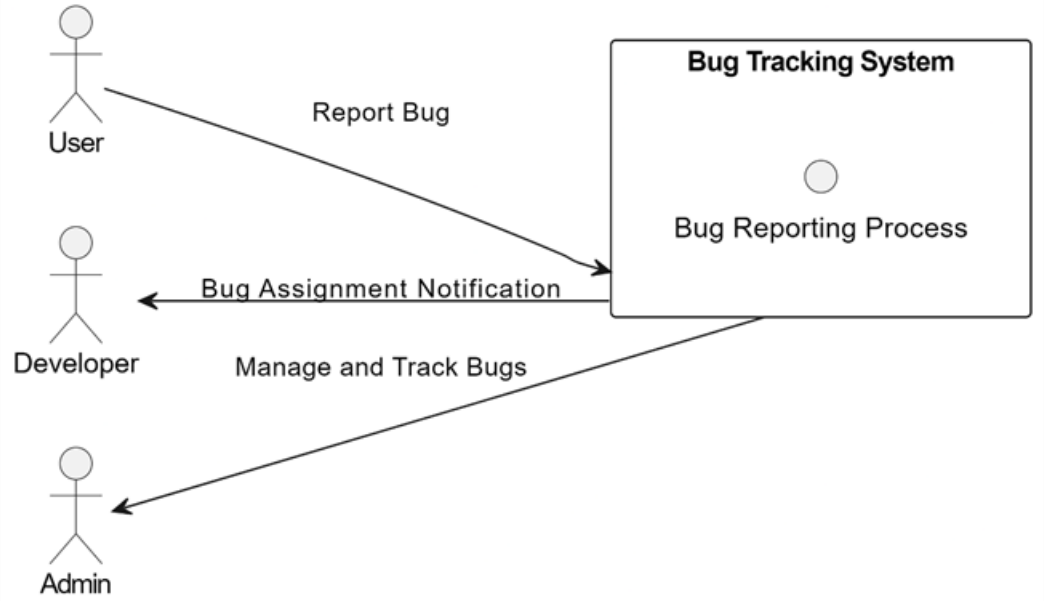
Level-0 DFD:

The **Level 0 DFD** (also called a **Context Diagram**) provides a high-level view of the system. It treats the system as a **single process** and identifies the external entities that interact with it.

Description

- **Entities**:
  - **User**: Submits bugs to the system.
  - **Developer**: Receives assigned bugs for resolution.
  - **Admin**: Oversees the system's functionality, manages bugs, and generates reports.
- **Process**:
  - **Bug Tracking System**: A single abstract process that handles all operations.
- **Data Flows**:
  - User submits bug details to the system.
  - Developers receive bug assignments and notifications from the system.

o Admin manages bugs and retrieves system reports.



(Level-0 DFD)

Level-1 DFD:

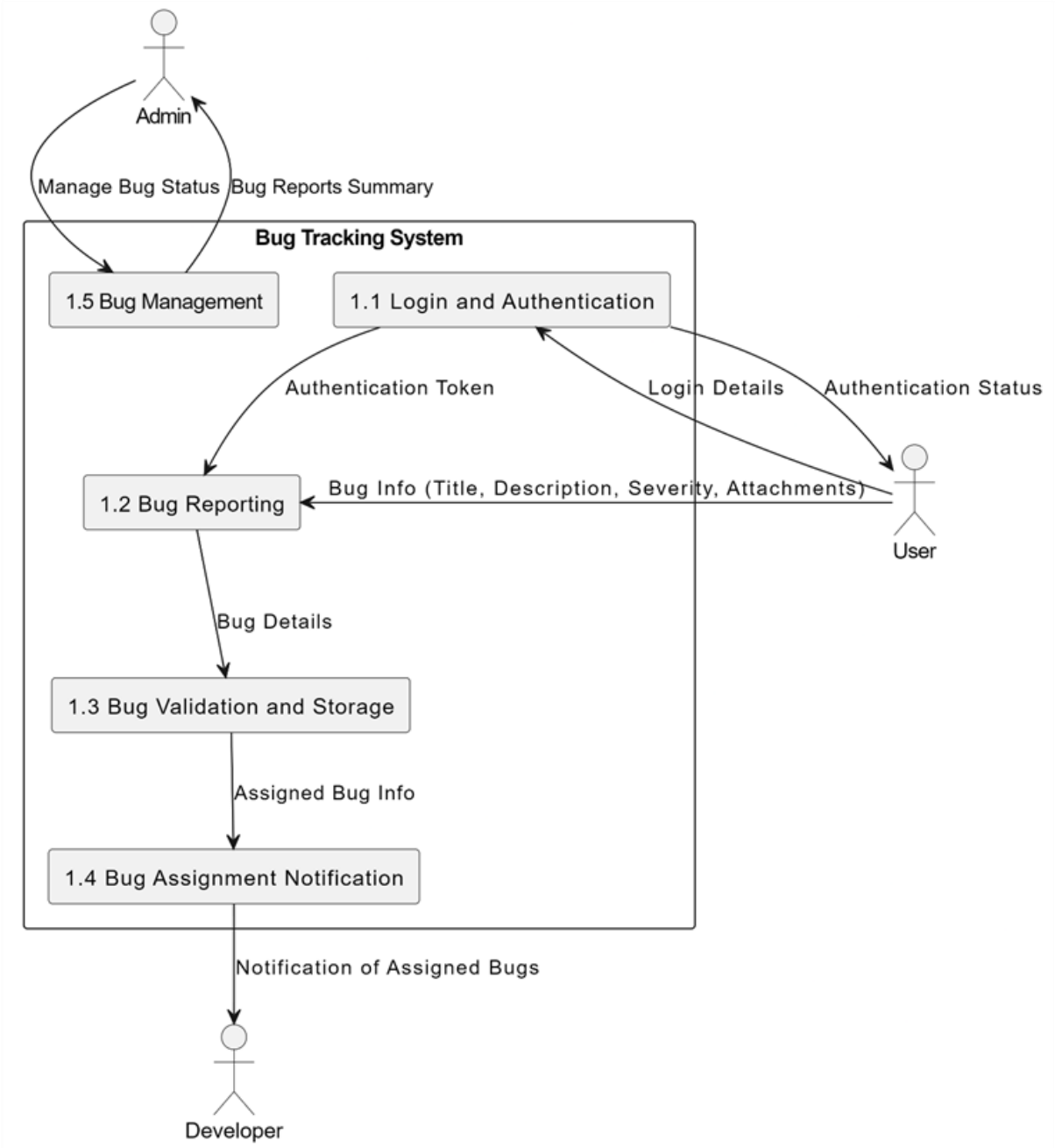
The Level 1 DFD provides a more detailed view, breaking down the system into five main processes. It focuses on the logical flow of data between processes, external entities, and data stores.

Processes:

1. Login and Authentication (1.1):
    - Verifies the credentials of users (e.g., admins, developers, or general users).
    - Receives login details and provides authentication status.
2. Bug Reporting (1.2):
    - Allows users to submit bugs with relevant details (e.g., title, description, severity, and optional attachments).
    - Passes validated bug data to the storage process.
3. Bug Validation and Storage (1.3):
    - Validates bug inputs for completeness and correctness.
    - Stores bug details in the database for tracking.
4. Bug Assignment Notification (1.4):
    - Assigns validated bugs to appropriate developers based on certain criteria (e.g., workload, expertise).
    - Sends notifications to developers with details of the assigned bugs.
5. Bug Management (1.5):
    - Allows the admin to manage the lifecycle of bugs (e.g., updating statuses, reassigning bugs, or resolving duplicates).
    - Generates detailed reports on the system's activities and bug statistics.

Entities and Data Flows:

1. User:
    - Inputs login details into the authentication module.
    - Submits bug information to the system.
    - Receives authentication status or error messages.
2. Developer:
    - Receives notifications for assigned bugs, including details such as severity and deadline.
3. Admin:
    - Sends bug management commands (e.g., update bug status).
    - Retrieves system-generated reports summarizing bugs.
4. Data Stores:
    - Bug Database: Stores all bug details and their lifecycle statuses.


Admin
Manage Bug Status
Bug Reports Summary
Bug Tracking System
1.5 Bug Management
1.1 Login and Authentication
Authentication Token
Login Details
Authentication Status
1.2 Bug Reporting
Bug Info (Title, Description, Severity, Attachments)
User
Bug Details
1.3 Bug Validation and Storage
Assigned Bug Info
1.4 Bug Assignment Notification
Notification of Assigned Bugs
Developer

(Level-1 DFD)

LAB-9

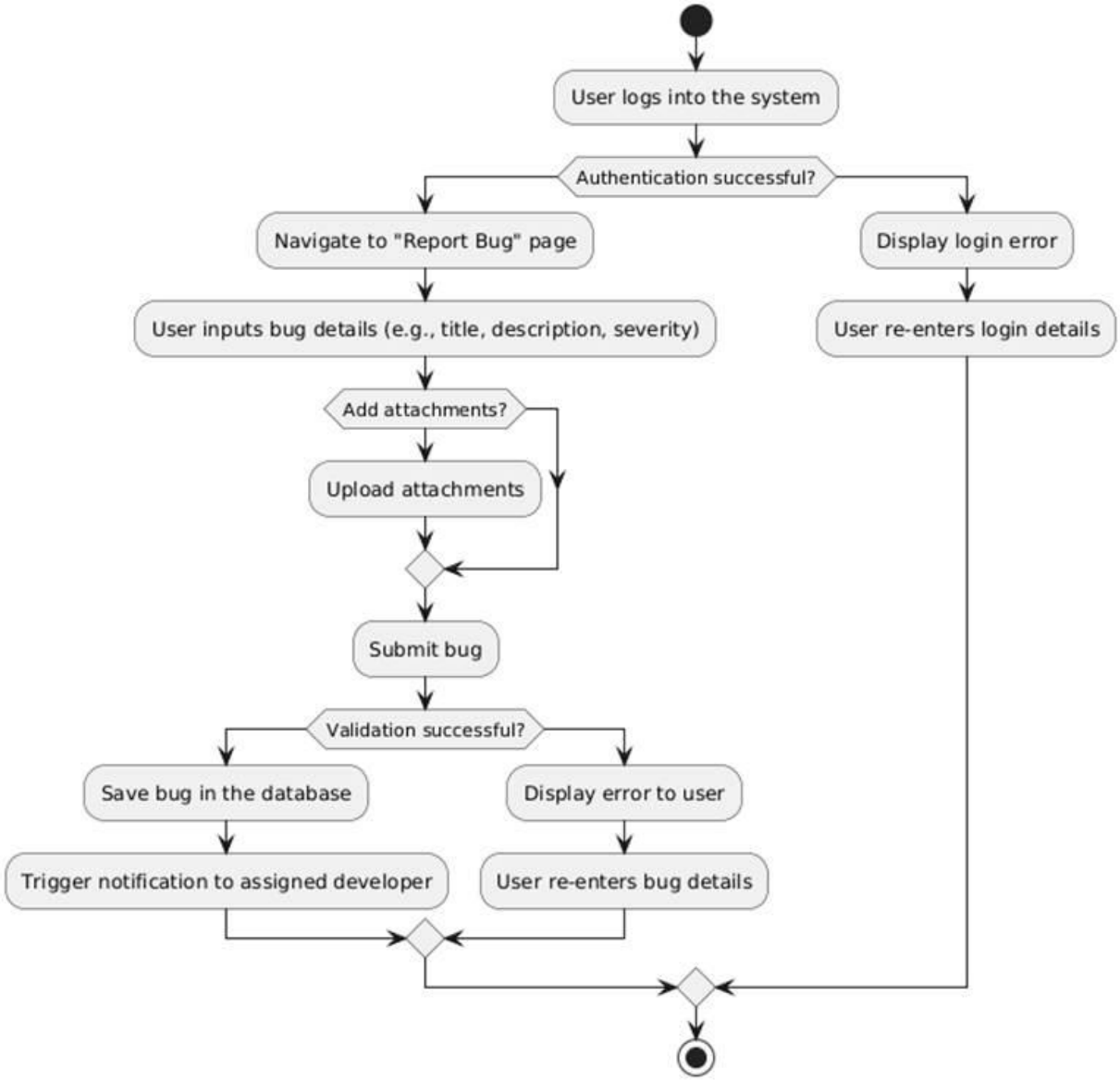
B8. Estimation of Test Coverage Metrics and Structural Complexity

Control Flow Graph:

A control flow graph (CFG) is a directed graph where the nodes represent different instructions of a program, and the edges define the sequence of execution of such instructions. Let's write a piece of code of the Bug-Reporting Module of the Bug Tracking System.
Pseudo Code:

```
Start
    DISPLAY "Please enter username and password"
    INPUT username, password
    IF Authentication is successful THEN
        DISPLAY "Navigate to Report Bug page"
        DISPLAY "Please enter bug details:"
        INPUT title, description, severity
        DISPLAY "Do you want to upload attachments? (Yes/No)"
        INPUT choice
        IF choice is "Yes" THEN
            DISPLAY "Please upload attachments"
        END IF
        DISPLAY "Submit bug"
        IF Bug details are valid THEN
            DISPLAY "Saving bug to database"
            DISPLAY "Notification sent to assigned developer"
        ELSE
            DISPLAY "Error: Please fix the bug details"
            DISPLAY "Re-enter bug details"
        END IF
    ELSE
        DISPLAY "Error: Invalid username or password"
        DISPLAY "Re-enter login details"
    END IF
End
```

Let's now draw the Control Flow Graph from the above Piece of Pseudo-code:



(CONTROL FLOW DIAGRAM of the BTS)

Computing Cyclomatic Complexity

Let *G* be a given CFG. Let *E* denote the number of edges, and *N* denote the number of nodes. Let *V(G)* denote the Cyclomatic complexity for the CFG. *V(G)* can be obtained in either of the following three ways, out of which, we considered 2:

- **Method #1:**

  V(G) = E - N + 2
  Here, we have a total of 13 Nodes (N) and 11 Edges (E).
  Edges (E) = 13; Nodes (N) = 11 PP = 1
  V(G)=E−N+2P
  V(G)=13−11+2(1)
  V(G)=13−11+2
  V(G)=4
- **Method #2**:

V(G) could be directly computed by a visual inspection of the CFG:
V(G) = Total number of bound areas + 1
Here, from the graph, we have Total Bound Areas as 3
So, V(G) = 3+1 = 4.

Result:

The **Cyclomatic Complexity** of the Bug Reporting Module's control flow is **4**.

This means there are 4 independent paths through the code, indicating the number of test cases you might need to fully cover all possible scenarios in this module. Generally, the higher the cyclomatic complexity, the more complex the code is, and the more test cases are needed for full coverage.

Optimum Value of Cyclomatic Complexity

A set of threshold values for Cyclomatic complexity has been reproduced below.

V(G)	Module Category	Risk
1-10	Simple	Low
11-20	More complex	Moderate
21-50	Complex	High
> 50	Unstable	Very high

It has been suggested that the Cyclomatic complexity of any module should not exceed. Doing so would make a module difficult to understand for humans. If any module is found to have Cyclomatic complexity greater than 10, the module should be considered for redesign.

Merits

McCabe's Cyclomatic complexity has certain advantages:

- Independence of programming language
- Helps with risk analysis during development or maintenance phase
- Gives an idea about the maximum number of test cases to be executed (hence, the required effort) for a given module

Demerits

Cyclomatic complexity doesn't reflect on cohesion and coupling of modules.

McCabe's Cyclomatic complexity was originally proposed for procedural languages. One may investigate to get an idea of how complexity calculation could be modified for object-oriented languages. In fact, one may also wish to make use of Chidamber-Kemerer metrics (or any other similar metric), which have been designed for object-oriented programming.

LAB-10

B9. Designing Test Suites

Test Suite Design for the BUG REPORTING MODULE

The **Bug Reporting Module** allows users to report bugs by providing details such as the title, description, severity, and optional attachments. It ensures the bug is correctly validated and stored in the system for further processing. Below is a comprehensive test suite designed for this module.

Test Suite for Bug Reporting Module

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result	Pass/Fail
TC-001	Verify mandatory fields in the bug reporting form	1. Navigate to the "Report Bug" page. 2. Leave mandatory fields (title, description) blank. 3. Click "Submit".	The error message displayed: "Mandatory fields cannot be empty."	Error "Mandatory fields cannot be empty."	**Pass**
TC-002	Verify successful bug submission with valid inputs	1. Navigate to the "Report Bug" page. 2. Fill in all mandatory fields. 3. Click "Submit".	Bug is successfully submitted, and a success message is displayed.	Successful	**Pass**
TC-003	Verify attachment upload functionality	1. Navigate to the "Report Bug" page. 2. Attach a valid file (e.g., .png, .pdf). 3. Submit the form.	File is successfully uploaded, and bug submission proceeds.	Successful	**Pass**
TC-004	Validate file size limit for attachments	1. Navigate to the "Report Bug" page. 2. Attach a file larger than the allowed size. 3. Submit the form.	Error message displayed: "File size exceeds the allowable limit."	Error "File size exceeds the allowable limit."	**Pass**
TC-005	Validate attachment file type	1. Navigate to the "Report Bug" page. 2. Attach an unsupported file type (e.g., .exe). 3. Submit.	Error message displayed: "Unsupported file type."	Error "Unsupported file type."	**Pass**

TC-006	Verify bug severity input	1. Navigate to the "Report Bug" page. 2. Select severity (e.g., "Critical"). 3. Submit the form.	Selected severity is recorded and displayed in the bug tracking system.	SEVERE	**Pass**
TC-007	Verify duplicate bug detection	1. Navigate to the "Report Bug" page. 2. Enter details of an existing bug. 3. Submit the form.	The system displays a warning: "A similar bug already exists in the database."	Warning	**Pass**
TC-008	Validate system response to invalid inputs	1. Navigate to the "Report Bug" page. 2. Enter special characters in the title/description fields. 3. Submit.	Error message displayed: "Invalid characters in input fields."	Error "Invalid characters in input fields."	**Pass**

Appendix C: To Be Determined List

- TBD-1: Integration of details with other tools.