Practical - 2

AIM: Prepare Software Requirement Specification (SRS) document for Bug Management System.

Software Requirements Specification (SRS)

Bug Management System

---

1. Introduction

1.1 Purpose

The purpose of this document is to outline the software requirements for the **Bug Management System (BMS)**, which is designed to facilitate the tracking, reporting, and resolution of software bugs. The system will enable developers, testers, and project managers to efficiently manage bugs throughout the software development lifecycle. This system aims to enhance collaboration, maintain detailed records of issues, and provide a structured workflow for bug resolution.

1.2 Scope

The Bug Management System is a web-based application that allows users to log, track, prioritize, and resolve bugs efficiently. The system will support role-based access and provide functionalities such as bug reporting, status tracking, assignment to developers, and real-time notifications. The system aims to improve software quality and reduce the time required to fix defects. Additionally, the system will provide dashboards, reporting capabilities, and integration with third-party tools such as Jira, Slack, and GitHub.

1.3 Definitions, Acronyms, and Abbreviations

- **BMS**: Bug Management System
- **User**: Any individual who interacts with the system (Tester, Developer, Manager, Admin)
- **Bug**: A defect or issue in the software that needs to be fixed
- **Priority**: The urgency of resolving the bug (Low, Medium, High, Critical)
- **Severity**: The impact of the bug on the system (Minor, Major, Critical, Blocker)
- **Status**: The current state of the bug (New, Assigned, In Progress, Resolved, Closed, Reopened)
- **Dashboard**: A visual representation of statistics related to bugs and their resolution

1.4 References

- IEEE Standard for Software Requirements Specification (IEEE 830-1998)
- Agile Software Development Best Practices
- Software Testing Principles and Practices
- OWASP Guidelines for Secure Software Development

1.5 Overview
This document details the system’s functionalities, performance requirements, external interfaces, and constraints. It provides a structured breakdown of the system's features, user roles, and technical requirements.

---

2. Overall Description

2.1 Product Perspective

The BMS is an independent web application that integrates with development tools such as Git, Jira, and Slack for improved bug tracking and communication. It will be designed using a microservices architecture to ensure scalability and modularity.

2.2 Product Functions

- **User Authentication**: Role-based access for different users (Testers, Developers, Managers, Admins).
- **Bug Reporting**: Users can report a new bug by providing relevant details such as title, description, severity, priority, and steps to reproduce.
- **Bug Assignment**: Bugs can be assigned to developers by managers.
- **Status Tracking**: Bugs transition through various statuses (New, Assigned, In Progress, Resolved, Closed, Reopened).
- **Commenting System**: Users can discuss issues within the bug report.
- **Real-time Notifications**: Email and in-app notifications for status updates, comments, and assignments.
- **Search & Filters**: Users can filter bugs based on priority, severity, status, assigned developer, and date.
- **Dashboard & Reports**: Visual representation of bug trends, resolution time, and productivity.
- **Audit Logs**: Maintain records of all actions performed on a bug.
- **Role-based Permissions**: Different access levels for testers, developers, managers, and admins.
- **API for Third-party Integration**: RESTful API for integration with CI/CD pipelines and external tools.

2.3 User Characteristics

- **Testers**: Report bugs and track their status.
- **Developers**: View assigned bugs, update statuses, and provide fixes.
- **Managers**: Assign bugs, prioritize them, generate reports, and oversee resolution progress.
- **Admins**: Manage user accounts, permissions, and system settings.

2.4 Constraints

- Must be a web-based application accessible via modern browsers.
- Should support integration with third-party tools like Jira and GitHub.
- Secure authentication must be implemented (OAuth 2.0 preferred).
- The system should be GDPR and OWASP compliant for security and privacy.

2.5 Assumptions and Dependencies

- The system will be hosted on cloud-based infrastructure.
- Users will have an internet connection and an updated web browser.
- Developers will use version control tools for tracking bug fixes.
- The system will use a relational database (e.g., PostgreSQL, MySQL) for storing bug data.

---

3. Specific Requirements

3.1 Functional Requirements

1. **User Authentication**
   - Users must log in using email and password.
   - Admins can manage user roles and permissions.
   - Support for Single Sign-On (SSO) authentication.

2. **Bug Reporting**
   - Users can submit a bug report with relevant details.
   - Bugs should have fields: Title, Description, Priority, Severity, Status, Steps to Reproduce, Attachments.

3. **Bug Tracking**
   - Users can update the status of bugs.
   - Managers can reassign bugs if necessary.

4. **Notifications**
   - Email and in-app notifications for status changes, assignments, and comments.
   - Configurable notification preferences for users.

5. **Search & Filters**
   - Search bugs using keywords, status, priority, severity, assignee.
   - Filters should be available for better sorting.

6. **Reports & Analytics**
   - Dashboard with visual statistics on bug reports and resolution time.
   - Exportable reports in CSV and PDF formats.

7. **Audit Logs**
   - Maintain a history of all bug-related actions.

8. **Third-party Integrations**
   - Support for Jira, Slack, and GitHub integration.
   - REST API for external access.

3.2 Non-functional Requirements

1. **Performance**
   - System should handle at least 1000 concurrent users.
   - Response time for loading pages should be under 3 seconds.
2. **Security**
   - Encrypted user data storage.
   - Secure API communication using HTTPS.
   - Role-based access control (RBAC) implementation.
3. **Usability**
   - Responsive design for mobile and desktop.
   - Intuitive UI with minimal learning curve.
4. **Scalability**
   - Should be able to scale horizontally with increasing users.
5. **Availability**
   - 99.9% uptime with automatic failover.

3.3 External Interface Requirements

3.3.1 User Interface

- Web-based interface accessible via modern browsers (Chrome, Firefox, Edge, Safari).
- Key components:
  - **Dashboard**: Displays analytics and quick actions.
  - **Forms**: Login, signup, feedback, and data entry.
  - **Notifications**: In-app and email alerts for updates.
  - **Responsive Design**: Optimized for desktops, tablets, and mobiles.

3.3.2 Hardware Interface

- Hosted on **cloud infrastructure** with:
  - Virtual machines or containers for deployment.
  - Load balancing for performance optimization.
  - Secure storage and backup mechanisms.
- Accessible from desktops, laptops, tablets, and smartphones.

3.3.3 Software Interface

- Integrates with third-party tools:
  - **GitHub** (version control), **Jira** (project management), **Slack** (team communication).
- Supports authentication via **OAuth 2.0** (Google, GitHub).
- Uses **relational (MySQL/PostgreSQL) or NoSQL (MongoDB) databases** for storage.

3.3.4 Communication Interface

- RESTful API for seamless integration with third-party services.
- API capabilities:
    - **User Authentication** (secure login/signup).
    - **Data Management** (CRUD operations).
    - **Integration Support** (external tool interaction).
- Uses **JSON format** for responses and **HTTPS with token-based authentication (JWT/OAuth 2.0)** for security.

3.4 Other Requirements

3.4.1 Legal and Regulatory Requirements

- The system must comply with relevant data protection regulations (e.g., GDPR, CCPA).
- The system must ensure that user data is stored securely and that users have the right to access and delete their data.

3.4.2 Performance Metrics

- The system shall be able to handle a minimum of 1000 bug reports per day.
- The system shall generate reports within 5 seconds of the request being made.
- The system shall support a minimum of 100 concurrent users without performance degradation.

3.4.3 Backup and Recovery

- The system shall implement daily backups of the database to prevent data loss.
- The system shall provide a recovery process that allows for restoration of data from backups within 24 hours of a data loss incident.

3.4.4 Accessibility Requirements

- The system shall comply with WCAG 2.1 Level AA accessibility standards to ensure usability for users with disabilities.
- The system shall provide keyboard navigation and screen reader compatibility.

3.4.5 Localization and Internationalization

- The system shall support multiple languages, allowing users to select their preferred language from a dropdown menu.
- The system shall store all user-facing text in a resource file to facilitate easy translation.

3.5 System Models

- **Use Case Diagram**: Shows interactions between Testers, Developers, Managers, and Admins.
- **ER Diagram**: Represents the relationships between users, bugs, and statuses.
- **Sequence Diagram**: Illustrates the flow of bug reporting and resolution.

4. Appendices

4.1 Glossary

- **Bug:** A defect in software that causes unintended behavior.
- **Bug Severity:** A classification of the impact of a bug on the system (e.g., Critical, Major, Minor).
- **Bug Priority:** A classification of the urgency of fixing a bug (e.g., High, Medium, Low).
- **User Roles:** Different levels of access and permissions within the system (e.g., Admin, Developer, Tester, Project Manager).
- **Test Case:** A predefined set of conditions to test software behavior.
- **Sprint:** A fixed development period in Agile methodology.

4.2 other

- **Appendix A**: User Roles and Permissions Table
- **Appendix B**: Sample Bug Report Form
- **Appendix C**: API Documentation for Integration

5. System Architecture

5.1 Architectural Design

The Bug Management System will follow a client-server architecture, where the client is a web application and the server handles business logic, data storage, and integration with third-party services.

5.2 Technology Stack

- **Frontend:** HTML, CSS, JavaScript (React or Angular)
- **Backend:** Node.js or Python (Django/Flask)
- **Database:** PostgreSQL or MongoDB
- **Hosting:** AWS, Azure, or similar cloud service

5.3 Architecture Overview

The system is divided into three main layers:

- **Presentation Layer (Frontend):** Provides the user interface for bug reporting and tracking.
- **Application Layer (Backend):** Processes business logic, handles bug assignments, and manages user roles.
- **Data Layer (Database):** Stores bug reports, user details, and status updates.

5.4 System Components

- **User Interface:** A web-based dashboard for reporting and managing bugs.
- **Authentication System:** Ensures secure access for developers, testers, and managers.
- **Bug Tracking Module:** Manages the lifecycle of bugs from reporting to resolution.
- **Notification System:** Alerts users about bug status changes and assignments.
- **Database:** Maintains structured records of bug reports, user actions, and system logs.

5.5 Communication Flow

1. Users submit a bug report through the UI.
2. The backend validates and stores the bug details in the database.
3. The bug is assigned to a developer based on priority and severity.
4. The developer updates the bug status as they work on the issue.
5. Once fixed, the bug is marked as resolved, and stakeholders are notified.

---

6. Future Enhancements

The Bug Management System can be further improved with additional features and optimizations. Some possible future enhancements include:

6.1 AI-Based Bug Classification

Implementing machine learning algorithms to automatically categorize and prioritize bugs based on severity and impact.

6.2 Integration with Version Control Systems

Connecting the system with GitHub, GitLab, or Bitbucket to link bugs with code changes and commits.

6.3 Automated Bug Assignment

Using AI-based recommendations to assign bugs to developers based on expertise, workload, and past performance.

6.4 Advanced Reporting and Analytics

Providing detailed insights into bug trends, resolution time, developer efficiency, and project stability through visual dashboards.

6.5 Mobile Application Support

Developing a mobile version of the system to allow users to report and track bugs on the go.

6.6 Multi-Language Support

Adding localization features to enable global usage by supporting multiple languages.

6.7 Automated Testing Integration

Integrating automated testing tools to detect potential issues before deployment and linking test results to bug reports.

---

7. Conclusion

The **Bug Management System** is designed to streamline bug tracking, assignment, and resolution in software development. By ensuring efficient collaboration among testers, developers, and managers, the system aims to enhance software quality, reduce debugging time, and provide better project insights.

Signature of Faculty: ________________