BugiNova

The Bug Tracker

Software Requirements Specification

AGENDA

1. Introduction
2. User Classes and Characteristics
3. Product Features
4. Activity Diagram
5. Workflow
6. Class Diagram
7. Programming Technologies
8. Third Party Dependency
9. Deployment Strategy
10. Cloud Architecture Diagram

1. Introduction

This Software Requirements Specification (SRS) document outlines the requirements for the development of BugiNova.

BugiNova is a web-based bug tracking application that allows users to create, track, and manage bugs or issues in software development projects. It will provide a centralized platform for developers, testers, and project managers to collaborate and efficiently resolve software bugs. The application will be developed using Python language, Django framework, and PostgreSQL and Redis databases, along with other tools and technologies. The application will be secure, efficient, reliable, and scalable, with a user-friendly interface and comprehensive functionality.

2. User Classes and Characteristics

BugiNova will have three types of users:

- **Test Manager: A user who is responsible for assigning the bug to the Developer..**
- **Tester: A user who is responsible for Testing the bugs and verifying it.**
- **Developer: A user who is responsible for resolving the software bugs.**
- **Reporter : Developer, Tester or Test Manager.**

3. Product Features

User Management:

BugiNova must include user management functionality, allowing users to register and create accounts. The registration process must be secure and user-friendly, requiring a unique username and password. Once registered, users should be able to log in to the application and access their dashboard or home page. The application must allow for different user roles and permissions, such as administrator, project manager, or developer. Each role must have different levels of access and permissions, which must be defined and managed by the application administrator.

Bug Tracking:

The core feature of BugiNova is bug tracking. Users must be able to create and track bugs or issues in the software. The application must allow users to create a new bug and provide details about the issue, including steps to reproduce it, the severity of the issue, and the priority level. The interface for creating and editing bugs must be clear and intuitive, including the ability to attach screenshots or other files. The application must provide a list of all bugs, as well as the bugs assigned to users or their teams. Users must be able to filter and sort the bug list based on various criteria, such as severity or status. The application must also allow users to search for bugs based on keywords or other criteria.

Bug Assignment:

Bug assignment is a critical feature of BugiNova, allowing users to assign bugs to specific team members or groups. Users must be able to assign bugs to themselves, or to other team members, and reassign bugs as needed. The application must provide clear notifications when a bug is assigned or reassigned, and users must be able to view the bugs assigned to them or their team. The application must also allow users to add notes or comments to a bug, which can be viewed by other team members.

Status Tracking:

Status tracking is another important feature of BugiNova, allowing users to track the status of bugs, including their current state (e.g., open, in progress, resolved) and the person or team responsible for resolving them. The application must provide a clear and intuitive interface for viewing the status of bugs, including a dashboard or summary view. Users must be able to update the status of bugs as they are resolved, and the application must provide notifications to other team members when a bug's status changes. The application must also allow users to view the history of a bug, including all status changes and comments.

Notification and Communication:

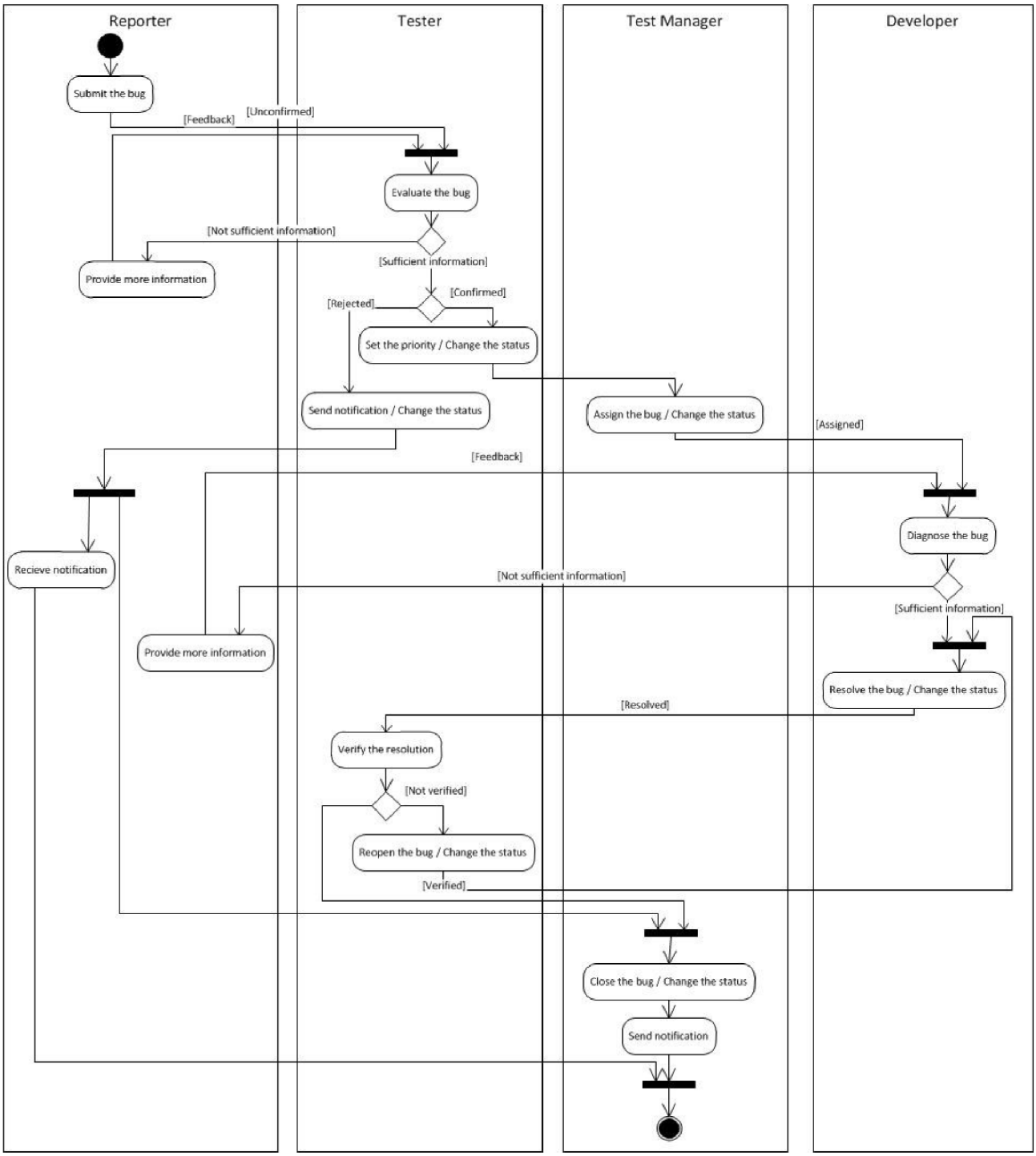
Notification and communication are critical features of BugiNova, allowing users to stay informed about bugs and communicate with other team members. The application must provide notifications to users when bugs are assigned to them or when their status changes. Users must be able to receive notifications via email or through the application interface. The application must also allow users to communicate with each other about bugs, either through comments or through an

integrated messaging system. Users must be able to view and respond to comments or messages, and the application must provide notifications when new comments or messages are added.

Reporting and Analytics:

Reporting and analytics are important features of BugiNova, allowing users to track bug trends, measure team performance, and generate reports. The application must provide comprehensive reporting and analytics functionality, including the ability to generate reports

4. Activity Diagram



Bug activity diagram.

5. Workflow

0. The reporter opens a bug report containing the following information:

- **A short but clear and explanatory description of the issue**
- **Environment details including:**

  - Hardware configuration

  - Operating system

  - Dependent software

  - Browser version

  - Product version

  - Component which failed

- **Steps to reproduce the problem: a concise and minimal set of steps which can be followed to reproduce the problem.**
- **Expected results: this should describe the expected behavior or expected results which will readily explain why the problem is being reported when compared to the actual results.**
- **Actual results: this should describe what actually happened, complete with any error messages, stack traces, screenshots, or log files that show the outcome. It may be the case that the behavior is correct but has**

been misunderstood when compared to the expected results. In this case, the resolution may be that clearer documentation is required.

1. The reporter submits the bug as unconfirmed to the tester.

2. The tester evaluates the bug's information and determines whether it is sufficient or not.

2.1. If the information is not sufficient, the tester notifies the reporter to provide more information.

2.2. The reporter provides more information as feedback to the tester.

2.3. The tester evaluates the bug again.

3. If there is sufficient information, the tester decides whether the bug is confirmed or rejected.

4. If the bug is rejected, the tester changes the status of the bug to "Rejected" and sends a notification to the reporter and the test manager.

4.1. The reporter receives the notification.

4.2. The test manager receives the notification, closes the bug, changes its status to "Closed," and sends a notification to all participants.

5. If the bug is confirmed, the tester:

Tests the bug.

Sets the priority of the bug according to the following standards:

- Immediate (5): The bug blocks development or testing work and should be fixed, or is a security issue in

a released version of the software.

- Urgent (4): The bug blocks usability of a large portion of the product and should be fixed before the

next planned release.

- High (3): Seriously broken, but not as high impact. Should be fixed before the next major release.

- Normal (2): Either a fairly straightforward workaround exists or the functionality is not very important

and/or not frequently used.

- Low (1): The bug is not all that important.

Sets the severity of the bug according to the following standards:

- Blocker: The bug is so important that it prevents development or testing on the affected product.

- Critical: The bug causes the product to crash or a function does not work at all.

- Major: The bug affects the product's ability to be operational.

- Normal: The bug impacts the product to work improperly.

- Minor: The bug causes the product to not work optimally, but it is possible to workaround such a bug.

- Trivial: The bug irritates the user, something is clearly not right, but it does not affect usability.

- Enhancement: A proposal of new functionality that improves the product in the future.

Change the status to "New."

Pass the bug to the test manager.

6. The test manager:

Review the bug.

Acknowledges the bug.

Changes the status to "Assigned."

Assigns the bug to the developer.

7. The developer diagnoses the bug.

7.1. If the information is insufficient, the Developer notifies the Reporter to provide more information.

7.2. The Reporter provides more information as feedback to the Developer.

7.3. The Developer diagnoses the bug again and resolves the bug if the information is sufficient.

7.3.1. The Developer changes the status to "Resolved."

- Fixed: The bug has been checked and tested and is now resolved.

- Wontfix: The bug is described as one that will never be fixed.

- Duplicate: The bug has been repeated more than once.

- Worksforme: All attempts at reproducing this bug were futile, and reading the code produces no clues

as to why the described behavior would occur.

- Invalid: The bug is in some way not valid.

7.3.2. The Developer passes the bug back to the Tester.

8. The Tester verifies the resolution.

8.1. If the resolution is not verified, the Tester:

- Reopens the bug.

- Changes the status to "Reopened."

- Pass the bug back to the Developer.

8.2. If the resolution is verified, the Tester:

- Changes the status to "Verified."

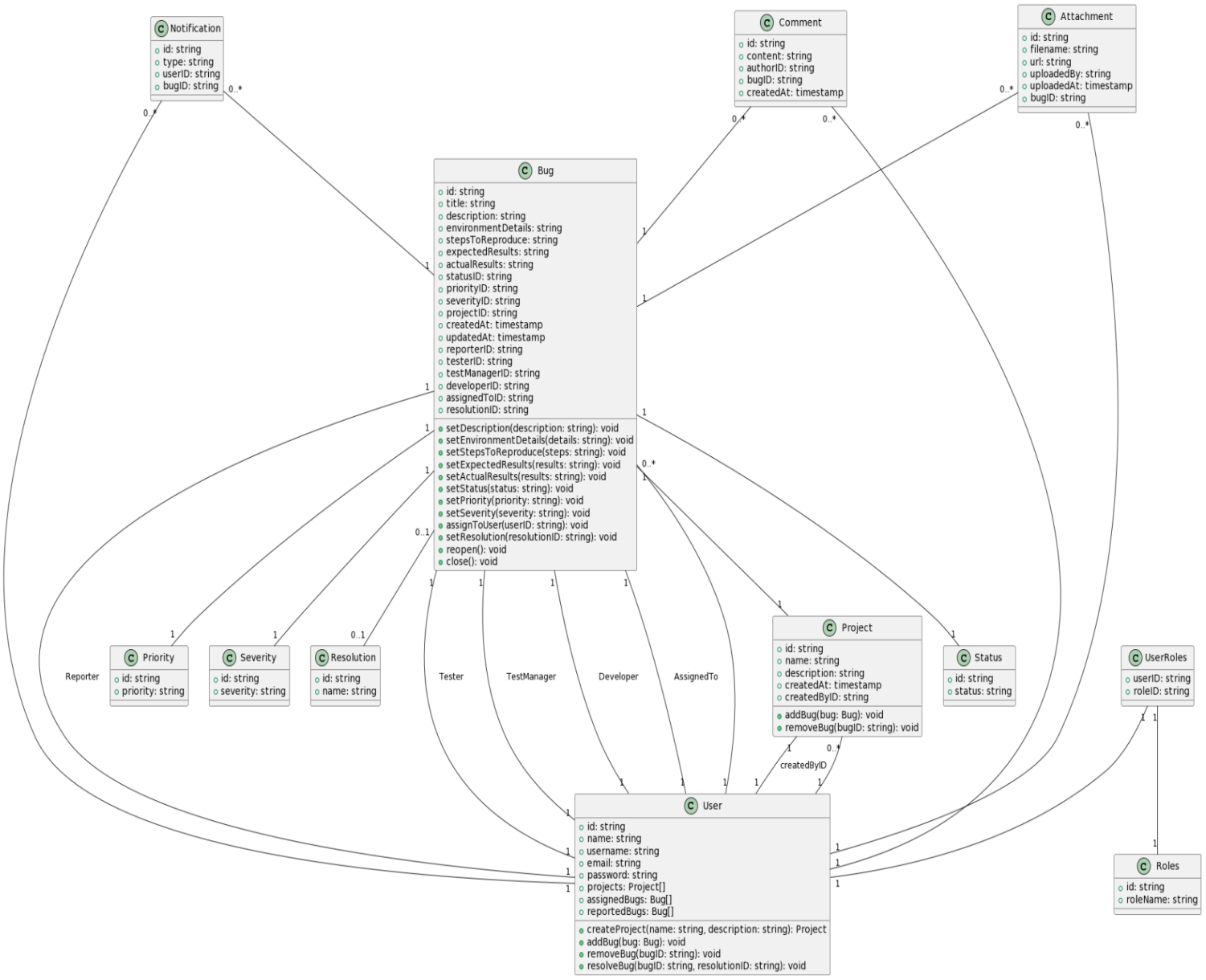
- Pass the bug to the Test Manager.

9. The Test Manager closes the bug.

Change the status to "Closed."

Send notifications to all participants.

6. Class Diagram



7.Programming Technologies

- **Programming Language: Python**
- **Web Framework: Django**
- **Database: PostgreSQL + Redis**

8. Third Party Dependency

- **Security: Auth0**
- **UI: Bootstrap**

9. Deployment Strategy

The BugiNova's Cloud Architecture combines MVC and Microservices, utilizing Kubernetes and Docker for improved scalability, portability, and management:

1. Microservices Containerization: Docker allows packaging microservices with their dependencies into isolated containers, enabling consistent development, testing, and deployment across different environments.

2. Orchestrating Microservices with Kubernetes: Kubernetes offers container orchestration and management capabilities, facilitating efficient deployment, scaling, and high availability of microservices.

3.MVC within Microservices: Each microservice can implement its own MVC pattern internally. For example, you can have separate microservices for user management, bug tracking, notifications, and project management, where each microservice follows the MVC pattern independently.

4. Scaling and Load Balancing: Kubernetes supports horizontal scaling of microservices based on resource utilization, automatically balancing loads across multiple instances to maintain performance during high traffic.

5. Service Discovery and Communication: Kubernetes provides service discovery mechanisms, allowing microservices to dynamically communicate. The MVC applications (within Microservices) can seamlessly interact with each other using Kubernetes service names or endpoints.

10. Cloud Architecture Diagram