Software Requirements

Specification

For

Mantis

Version 1
Prepared by Court of Owls
Faculty Of Information Technology
University Of Moratuwa
July 2020

Supervisor: Dr. Priyanga D. Thalagala

Team Members

184025L - Chandrasiri M.D.N.
184038E - Edirisinghe E.A.Y.R.
184116R - Pathirana W.P.C.P.
184152X - Sandarekha W.A.D.
184157R - Senanayake I.K.G.D.S.N.

Table of Contents

1.INTRODUCTION

1.1 Purpose

In mass systems a software bug is a failure or a drawback in a system that directs to an unexpected result or time-consuming effort. Purpose of this system is to solve those problems in a user friendly manner. Reporting these kinds of errors immediately is essential to get the best result out of the system. In this system we are developing a platform for all the user roles to interact which is an effective way of reporting issues/bugs.

This document is presented for further understanding of the open-source web-based bug tracker Mantis. This document contains the features of the software, purpose and the technologies used. This document has outlined the functional requirements, non functional requirements, design constraints and supporting factors related to the web application of the proposed system.

1.2 Product Scope

The purpose of this software is to enhance the user friendliness in the bug reporting process by providing various tools that are not available in most of the bug trackers currently available. Sub modules in this system have different areas to cover such as,

- User friendly UI
- Screen capturing capabilities
- Inbuilt sprint creating tools
- Bug solution pool for knowledge sharing
- Graphical Representation with summarized reports

1.3 Definitions, acronyms and abbreviations

Acronym/ Definition	Meaning
SRS	Software Requirement Specification
Route	A public transit path
RBAC	Role Based Access Control
GUI	Graphical User Interface
BCL	Bug Capture log
BMS	Bug Management System
BSP	Bug Solution Pool
BR	Bug Reporting

1.4 References

IEEE Template for System Requirement Specification Documents:
https://goo.gl/nsUFwy

GNU General Public License version 3:
http://www.gnu.org/licenses/gpl.html

CDDL Common Development and Distribution License:
https://opensource.org/licenses/CDDL-1.0

1.5 Overview

The rest of the document will first outline the proposed architecture of the system along with an abstract introduction to the usage of each module. Then the document will explain the detailed identified system requirements, non-functional requirements. The document will also provide details about performance requirements and design and implementation constraints. This document intends to provide a comprehensive idea to all the user roles regarding the overall architecture and functioning of the proposed system.

2.OVERALL DESCRIPTION

2.1 Product Perspective

This Multi-tenant Issue Tracking and Reporting System has some common features with other related systems such as Backlog, SpiraTeam, BugZilla and LightHouse which as listed below.

- Advanced filtering and sorting options
- Access to active and resolved issues
- Kanban boards for visual workflow
- Reporting module with visual aids
- Setting Bug priorities
- Effective tagging system
- Sprint Management

However, this system includes some unique features such as,

- Insert supporting files without third party tools
- Role Based Access Control(RBAC)
- Maintain a bug pool in order to provide knowledge to junior developers.

In conclusion, this system provides an overall solution for the bug tracking process in an organization.

2.2 Product Functions

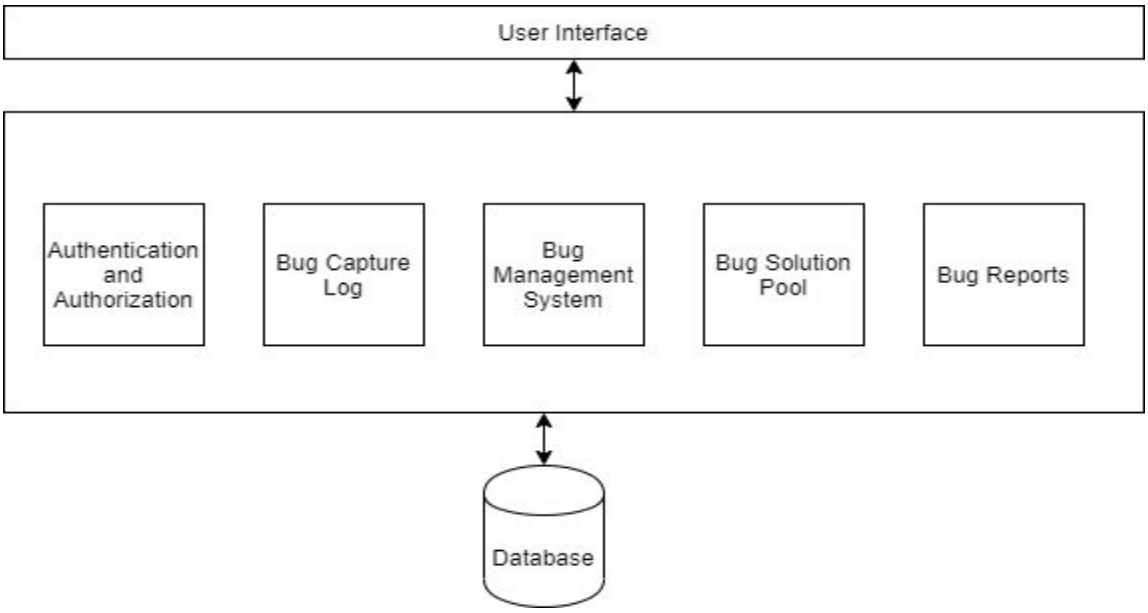

Figure 1: Architecture Diagram

This product can be categorized into different subcategories/functions like

- Authentication and Authorization
- Bug Capture Log
- Bug Management System
- Bug Solution Pool
- Bug Reporting System

Each subcategory has its unique functionalities and different access points for user roles.

2.2.1 Authentication and Authorization

This follows the Role Based Access Control (RBAC) method to provide permissions to users.This should have below multi-tenant sub functions,

- Users can login to the system.
- Users can make a password request.
- Allow defining a super admin
- Allow admin to create users
- Allow admin to create and provide user roles.
- Allow admin to define and provide permissions for user roles.
- Allow admin to create projects and assign users to the projects.

2.2.2 Bug Capture Log

BCL is an interface where customers can engage with the system. Customers can submit their issues through this interface. Customers can login to the system with provided credentials and the following sub functions will be available in this sub module.

- Customers can view the project where he/she is a part of.
- Customers can select a project and view previously filed issues.
- Customers can select a project and file an issue.
- Issue filing process can be listed down as follows,
    - Customer selects “Create new Issue” button
    - Issue window allows input of a description
    - Issue window allows uploading files
    - Issue window allows screen recordings

Recording screen is the most vital feature in issue creation. Customers should be able to turn on-screen recording and record the bug. Existing libraries can be used to achieve this feature or can use built-in screen capturing options. Once the recording is done, it should be attached as a video to the issues.(Customers can comment on created issues if necessary.)

2.2.3 Bug Management System

This sub module will be only available for the internal users. After creating an issue, it will be visible in the ‘Backlog’ which is a pool of all the bugs. From this pool, managers can sort and filter issues in order to add them into sprints. Each sprint will consist of a Kanban board with four lanes. The four lanes will be,

- Open
- In progress
- Review
- Completed

Following sub functions will be available in BMS

- Internal users can login to the system.
- Managers can create sprints with due dates and add issues to each sprint.
- Managers can assign QAs and developers to issues.
- QAs can,
    - Change tags of the bug.
    - Assign himself/herself to the bug
    - Assign someone else to the bug
    - User can add time spent for a bug (time tracking)
- Developers can,
    - Resolve issues
    - Assign himself/herself to the bug
    - Assign someone else to the bug
    - Attach files to the bug
    - Add comments to the bug
    - User can add time spent for a bug (time tracking)

2.2.4 Bug Solution Pool

This platform will be used as a knowledge base which includes solved bugs for reference. Here Managers/QAs/Developers can add comments to the bugs. The system should allow full-text search in order to provide ease of searching.

2.2.5 Bug Report

Reporting section is the graphical representation and overall summary of the bugs and updates about current progress. Developer work breakdown, developer KPIs, project KPIs, sprint performance monitoring and KPIs, full bug summary can be recognized as the sub functionalities in this section.

2.3 User Classes and Characteristics

The user categories of the proposed system are as follows.

- Internal Users:
    - Administrators
    - Managers
    - Quality Assurance Engineers
    - Developers
- External Users:
    - Customers

2.3.1 Administrators

Super administrator can be created once at the system setup. Here system allows admin to create users and their user roles. Admins are the ones who assign roles to users and define permissions for different roles. Creating projects and Assigning users to projects are the main tasks in this user role

2.3.2 Managers

Managers' task is to handle the issues of each and every user. So manger can create a sprint and can add bugs from bug log to sprint which is a kanban board based platform to review that particular issue

2.3.3 Quality Assurance Engineers

QA role is assigned in the Bug management System who can access the dashboard. QA Engineer can only view filed bugs and change status. (Mainly focuses on testing and debugging)

2.3.4 Developers

Developers have the access to the bug management dashboard. Developer along with the manager can create sprints and one developer is assigned to one particular sprint.

2.3.5 Customer

Customer is the one who files the issue. So in the issue filing process they allow input of a description, file uploading , screen recording options and comment section tc. After filing the issue it will be added to the capture log.

2.4 Dependencies

- Working with specific web browsers (outdated) may cause problems.

2.5 Apportioning of Requirements

Due to some limitations in the project, some parts of the project are expected to be developed in the future versions. Here are some of the ones available for future development.

1.Uploading media files such as videos and screenshots with resizing features (crop method) may be introduced in the future.

3.SPECIFIC REQUIREMENTS

3.1 External interfaces

3.1.1 Inputs and Outputs

A software bug is an error, failure, flaw, or fault in a computer program or system that causes wrong or unpredictable results or unexpected behaviors. So when they get an error they can put their bug as an input to our application. Clients can attach media files, descriptions related to the bug. Applications get them as user inputs and generate the bug report as the output. Also output in this software is called as a 'ticket'.

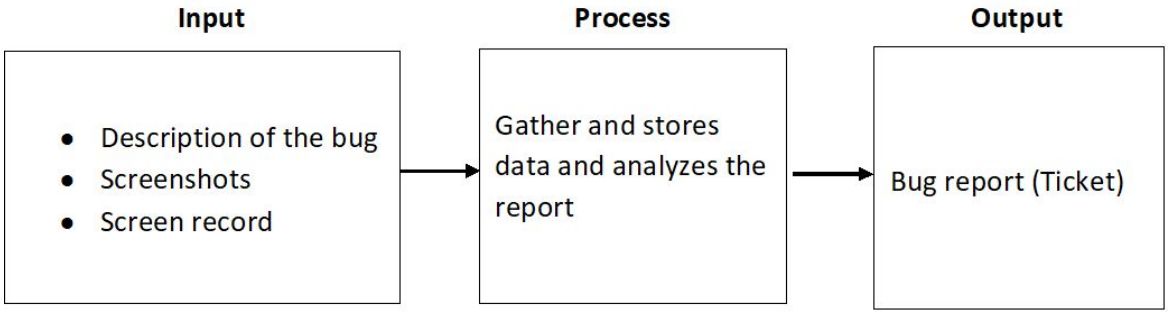

Figure 2: IPO Diagram

3.2 Functional requirements

3.2.1 Authentication and Authorization

3.2.1.1 User Verification and Validation

3.2.1.1.1 Description and Priority

The system will read the user credentials in order to verify the details and provide a token.
Priority: High

3.2.1.1.2 Stimulus/ Responses Sequence

Stimulus: Provide user credentials.
Responses: Verification,validation and provide access.

3.2.1.1.3 Functional Requirements

Requesting authentication API (READ Credentials)

3.2.1.2 Role Based Access Control

3.2.1.2.1 Description and Priority

The system will have multi-tenant capabilities.There should be different user roles with different permissions.
User Roles: Administrator, Manager, QA, Developer
Priority: High

3.2.1.2.2 Stimulus/ Responses Sequence

Stimulus: Create, read, update and delete internal users.
Responses: Return user object.

3.2.1.2.3 Functional Requirements

Requesting Internal User API (CREATE Users)
Requesting Internal User API (READ Users)
Requesting Internal User API (UPDATE Users)
Requesting Internal User API (DELETE Users)

3.2.1.3 Entity Management

3.2.1.3.1 Description and Priority

The administrator should be able to enroll external users to the system.
Priority: High

3.2.1.3.2 Stimulus/ Responses Sequence

Stimulus: Create, read, update and delete entities.
Responses: Return entity object.

3.2.1.3.3 Functional Requirements

Requesting Entity API (CREATE Entity)
Requesting Entity API (READ Entity)
Requesting Entity API (UPDATE Entity)
Requesting Entity API (DELETE Entity)

3.2.1.4 Project Management

3.2.1.4.1 Description and Priority

The system should be able to manage multiple projects simultaneously. Moreover, the administrator should be able to assign users and entities to the project.
Priority: High

3.2.1.4.2 Stimulus/ Responses Sequence

Stimulus: Create, read, update and delete projects.
Responses: Return project object.

3.2.1.4.3 Functional Requirements

Requesting Project API (CREATE Project)
Requesting Project API (READ Project)
Requesting Project API (UPDATE Project)
Requesting Project API (DELETE Project)

3.2.2. Bug Capture Log

3.2.2.1 Issues

3.2.2.1.1 Description and Priority

The external entity should have the capability to file issues under the projects assigned to them. After the issue form submission, a ticket will be created including all the details of the form.
Priority: High

3.2.2.1.2 Stimulus/ Responses Sequence

Stimulus: Create, read, update and delete issues.
Responses: Return ticket object.

3.2.2.1.3 Functional Requirements

Requesting Project API (READ Projects)
Requesting Ticket API (CREATE Ticket)
Requesting Ticket API (READ Tickets)
Requesting Ticket API (UPDATE Ticket)
Requesting Ticket API (DELETE Ticket)
Requesting Comment API (CREATE Comment)
Requesting Comment API (READ Comment)
Requesting Comment API (UPDATE Comment)
Requesting Comment API (DELETE Comment)

3.2.2.2 Screen Capturing

3.2.2.2.1 Description and Priority

Capture screen upon the external user's request.
Priority: High

3.2.2.2.2 Stimulus/ Responses Sequence

Stimulus: Request screen capturing.
Responses: Capturing the screen.

3.2.2.2.3 Functional Requirements

Request Use-screen-recording npm module

3.2.3. Bug Management System

3.2.3.1 Backlog

3.2.3.1.1 Description and Priority

Maintain a list of tickets with filtering and sorting options.
Priority: High

3.2.3.1.2 Stimulus/ Responses Sequence

Stimulus: View tickets.
Responses: Return ticket object.

3.2.3.1.3 Functional Requirements

Requesting Ticket API (READ Tickets)
Requesting work state API (CREATE work state)

3.2.3.2 Sprint Management

3.2.3.2.1 Description and Priority

Manage sprints to ease the process of solving issues.
Priority: High

3.2.3.2.2 Stimulus/ Responses Sequence

Stimulus: Create, read, update and delete sprints.
Responses: Return sprint object.

3.2.3.2.3 Functional Requirements

Requesting sprint API (CREATE Sprint)
Requesting sprint API (READ Sprint)
Requesting sprint API (UPDATE Sprint)
Requesting sprint API (DELETE Sprint)

3.2.3.3 Workflow Management

3.2.3.3.1 Description and Priority

Manage the workflow inside the sprint.
Priority: Medium

3.2.3.3.2 Functional Requirements

Requesting work state API (READ work state)
Requesting work state API (UPDATE work state)
Requesting work state API (DELETE work state)
Requesting Comment API (CREATE Comment)
Requesting Comment API (READ Comment)
Requesting Comment API (UPDATE Comment)
Requesting Comment API (DELETE Comment)

3.2.4. Bug Solution Pool

3.2.4.1 Bug Pool

3.2.4.1.1 Description and Priority

Maintaining a record of tickets in order to provide details to future beneficiaries.
Priority: high

3.2.4.1.2 Stimulus/ Responses Sequence

Stimulus: View bug pool.

Responses: Return a list of tickets which belongs to the bug pool.

3.2.4.1.3 Functional Requirements

Requesting Ticket API (READ Tickets)

Requesting Ticket API (UPDATE Ticket)

Requesting Comment API (CREATE Comment)

Requesting Comment API (READ Comment)

Requesting Comment API (UPDATE Comment)

Requesting Comment API (DELETE Comment)

3.2.4.2 Full Text Search

3.2.4.2.1 Description and Priority

The system should provide the ability to search tickets using the Full text search method.

Priority: Medium

3.2.4.2.2 Stimulus/ Responses Sequence

Stimulus: Search a user given text.

Responses: Return ticket object.

3.2.4.2.3 Functional Requirements

Requesting Ticket API (READ Tickets)

Requesting Comment API (READ Comment)

3.2.5. Bug Reports

3.2.5.1 Developer Work Breakdown

3.2.5.1.1 Description and Priority

The system should create a monthly report of every developer with details of the bugs he worked on and the total time spent.

Priority: High

3.2.5.1.2 Stimulus/ Responses Sequence

Stimulus: View developer timesheet.

Responses: Return developer timesheet report object.

3.2.5.1.3 Functional Requirements

Requesting Internal User API (READ Users)
Requesting sprint API (READ Sprint)
Requesting Ticket API (READ Tickets)

3.2.5.2 Project KPIs

3.2.5.2.1 Description and Priority

The system should create a monthly report for every project with details of time spent on debugging issues related to the project.
Priority: High

3.2.5.2.2 Stimulus/ Responses Sequence

Stimulus: View project timesheet.
Responses: Return project timesheet report object.

3.2.5.2.3 Functional Requirements

Requesting Ticket API (READ Tickets)
Requesting Entity API (READ Entity)
Requesting Project API (READ Projects)

3.2.5.3 Sprint Performance Monitoring and KPIs

3.2.5.3.1 Description and Priority

At the end of every sprint, a report should be created detailing the sprint issues management.
Priority: High

3.2.5.3.2 Stimulus/ Responses Sequence

Stimulus: View sprint report.
Responses: Return sprint report object.

3.2.5.3.3 Functional Requirements

Requesting sprint API (READ Sprint)
Requesting work state API (READ work state)
Requesting Ticket API (READ Tickets)
Requesting Project API (READ Projects)
Requesting Entity API (READ Entity)
Requesting Internal User API (READ Users)

3.2.5.4 Full bug summary

3.2.5.4.1 Description and Priority

The system should provide a monthly report on new additions to the issues and number of resolved issues.
Priority: Medium

3.2.5.4.2 Stimulus/ Responses Sequence

Stimulus: View full bug summary charts.
Responses: Return full bug summary report object.

3.2.5.4.3 Functional Requirements

Requesting work state API (READ work state)
Requesting Ticket API (READ Tickets)

3.3 Performance Requirements

Hardware

- 1 GB RAM
- 1 CPU Core
- 24 GB SSD Storage
- 2 TB Transfer
- 40 Gbit Network In
- 125 Mbit Network Out

Web Server

- 1.6 GHz CPU
- 1.75 GB RAM
- 40GB HDD

3.4 Interfaces

3.4.1 User Interfaces

3.4.1.1 Login page

3.4.1.1.1 *Function*

Login or signup to the system.

3.4.1.1.2 *Description*

This interface obtains inputs from the user to access the system by entering a valid email and password into the given text boxes. If a user is not already registered, the system will get some information from the user and allow the user to get signed up. After that process, the user can access the system by clicking the login button.

3.4.1.2 Bug capture log

3.4.1.2.1 *Function*

Mainly filing an issue and commenting on created issues.

3.4.1.2.2 *Description*

Bug capture log interface consists of a dashboard with relevant user projects. A window will appear when a user clicks on a project. There is a plus button to create an issue in that window. On click of the plus button, the user can input a description and add media files. Screen record button is used for on screen recording. Customers can add above media files with the ‘attached’ button at the bottom of the window. In addition there may be issues with pre-filed. When clicking on the comment, the user can view or comment on it.

3.4.1.3 Bug Management Interface

3.4.1.3.1 *Function*

Admin panel view bug reports and create sprints.

3.4.1.3.2 *Description*

Only Developers, Managers, QA can access this interface. After login to the system they can view the backlog. Users can identify the bugs related to a project while searching the project code in the search bar. There is a plus button on the left side of the bug report. The user can create a new sprint by clicking on 'Create New Sprint'. After creating a sprint, they can add bugs to it with the plus button described above. Also users can view the bug report and a window containing the comment section, 'Attach file' button, time tracking section and the tags associated with the report. The Sprint window (sub interface) contains 5 kanban lanes such as Open, In-progress, QA, Done. Users can drag and drop from one lane to the other.

3.4.1.4 Bug Solution Pool Interface

3.4.1.4.1 *Function*

List solved bugs for future reference

3.4.1.4.2 *Description*

This interface can be viewed by both admin and clients. They can view the old bug's information for future reference. If a user needs to get some relevant details about a bug, the system shows some bug reports while searching some keywords on the search bar.

3.4.1.5 Interface of Reports

3.4.1.5.1 *Function*

Monitor project progress

3.4.1.5.2 Description

Automatically generate some reports such as developer timesheet, project timesheet, sprint summary, Gant charts. Users can also view the reports.

3.4.2 Hardware Interfaces

No need for any special hardware requirements for this application.

3.4.3 Software Interfaces

Frontend is developed by Reactjs and backend is developed by Python/Django. Also linked to the Postgresql database.

3.4.4 Communication Interfaces

This application requires an internet connection. Because this is an online bug tracking system. Application will communicate over HTTP. Notifications could be sent by email or sms using SMTP,SMPPP.

3.5 Design and Implementation Constraints

3.5.1 Software Development Kits and Libraries

Overall system will be developed by using below technologies,

In the frontend,

Bootstrap - Use to build the basic structure of a particular user interface.

React 16 - Use as the main frontend framework.

Redux - Use to manage state.

Formik + Yup - Use to build forms.

In the backend,

Django - Use as the main backend framework.

Django Rest Framework - Use as the main API control module in the backend.

Postgresql - Use as the main database.

VsCode,PyCharm,pgAdmin are used as basic IDEs to develop the system.

3.5.2 Development Languages

Python,Javascript are used as main programming languages.
HTML/CSS are used as markup languages.
SQL is used as a query language.

3.5.3 Development Tools

The project will be hosted for source and version control in github using git.

3.5.4 Frontend Server

Web application would act as a front-end access user to the system and thus users are to be equipped with a compatible web browser and internet connection for the use of the service. No offline services will be available.

3.5.5 Backend Server

Django backend with postgresql database will work as a backend server

3.5.6 Other Constraints

- Endpoint naming convention to be used in the system can be described as follows,
    - http://api.example.com/api/users/ - GET request
    - http://api.example.com/api/users/id - POST/DELETE/UPDATE request

- The service will be available 24 hrs everyday.
- Users must have their correct credentials to enter into their user account.
- Only administrators can log in to the admin dashboard.
- External users can only log in to the bug capture log.
- Internal users can log in to the backlog,sprint,reports and workflow management according to their permissions.
- The system can handle as many as users concurrently.

3.6 Non Functional Requirements

3.6.1 Usability

3.6.1.1 Graphical User Interface

- The GUI will be consistent looking among all web pages.
- GUI of application will load within maximum time of 3 seconds
- GUI will be modified according to the latest designing and styling standards
- In Bug Reports GUI will contain different kind of graphical representations (Gantt Charts etc)
- Font size of readable text would be maintained at 22px.

3.6.1.1 Orientation TIme

- Users does not need any specific time for training
- Users who have some technical knowledge can fulfill their requirements without any external interaction

3.6.2 Reliability

- The data regarding the routes and transits will be entered to the system only through the approval of relevant authorities.

3.6.3 Availability

- Web applications will be available at any time for user access.
- Any user who has unique login credentials can access the system.

3.6.4 Security

- The system will authenticate each API call before processing
- All the data will be stored

3.6.5 Maintainability

- Mean Time To Repair will be 12 hours to 36 hours depending on the severity of the repairment.
- The system will provide an easily accessible interface for managing routing data.

4.APPENDIXES

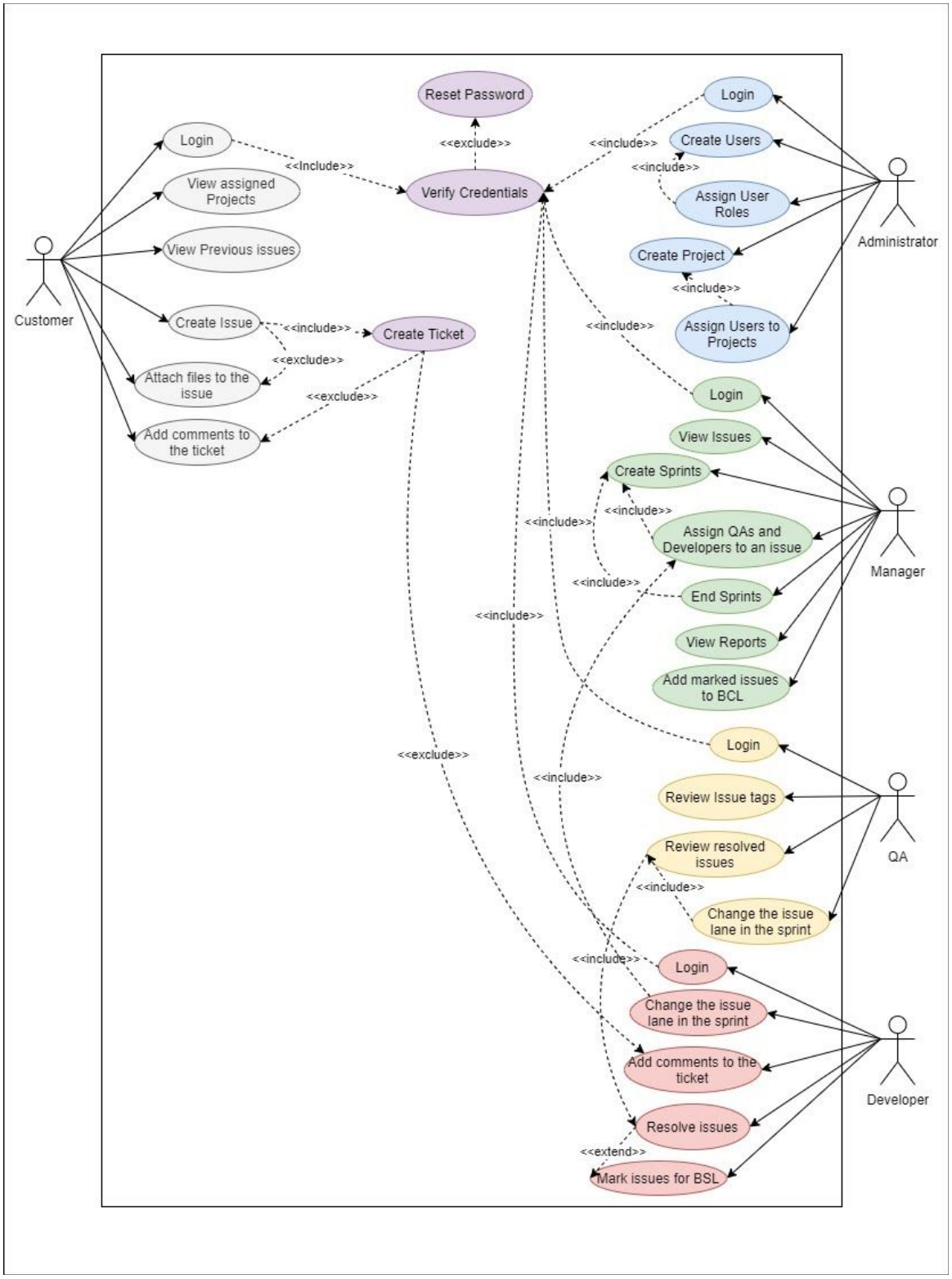

Figure 3: Use Case Diagram

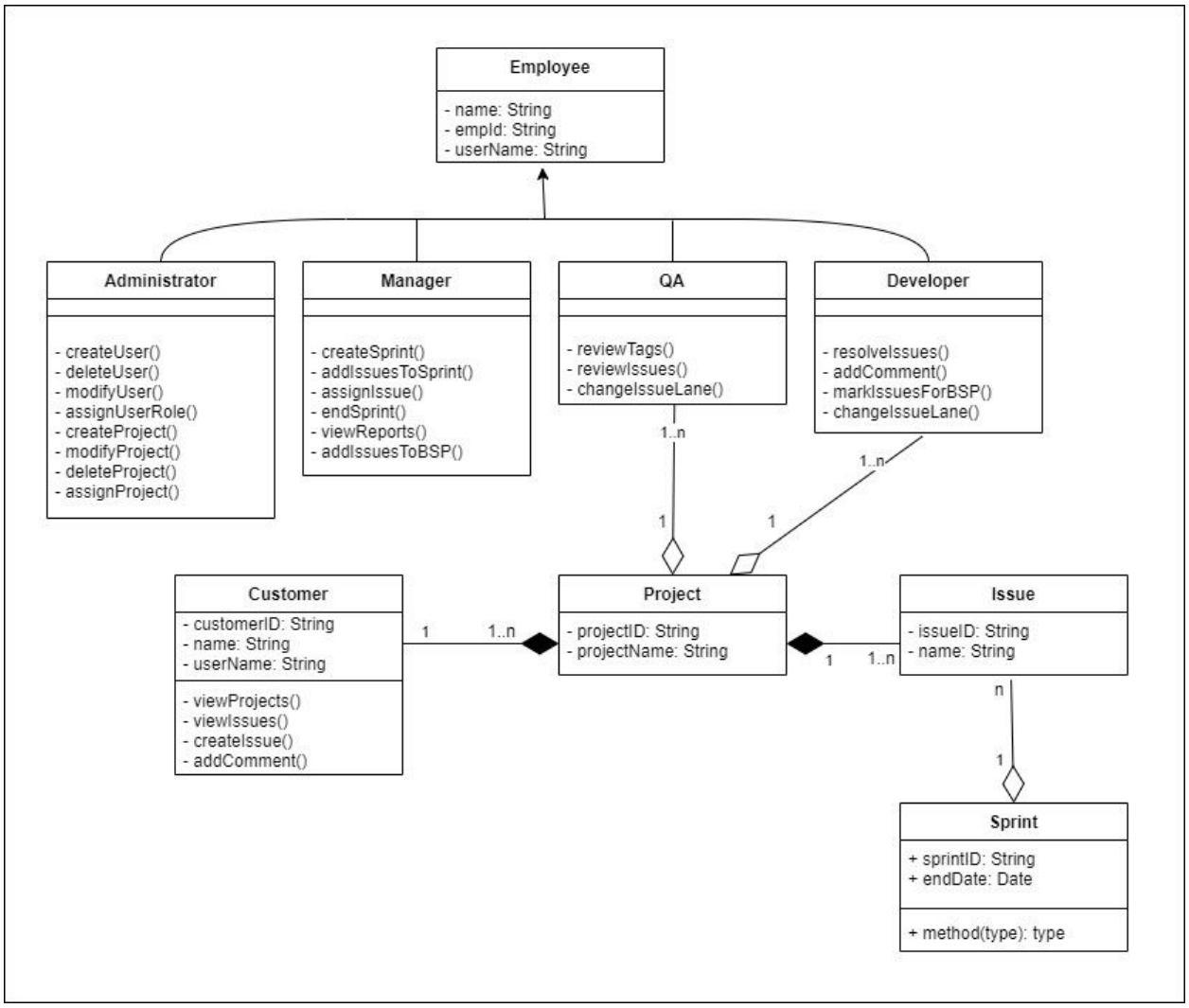

Figure 4: Class Diagram