Bug Tracker

Developed by Edith Molda

Table of Contents

1. Introduction

1.1 Purpose

The purpose of the bug tracker is to build a software application that will record facts about a program's bugs into a database. The facts will include the time the bug was reported, its severity, the program behaviour caused by the bug, how to reproduce the bug, the identity of the person who reported the bug and programmers that may be working on fixing the bug.

1.2 Document Conventions

This document uses the following conventions.

CGI	Computer-generated Imagery
DB	Database
DBMS	Database Management System
ER	Entity Relationship
HTML	HyperText Markup Language

1.3 Intended Audience and Reading Suggestions

This project is intended for developers and project managers.

1.4 Project Scope

The purpose of the bug tracker is to provide an accurate record of bugs throughout the development of a project. The system will have a database to record the time a bug was reported, its severity, the program behaviour caused by the bug, how to reproduce the bug, the identity of the person who reported the bug and programmers that may be working on fixing the bug. The system will also allow administrators to update the status of a bug. Overall, the bug tracker will be a useful tool for developers and project managers to use in the process of developing a project.

1.5 References

https://en.wikipedia.org/wiki/Bug_tracking_system
https://krazytech.com/projects/sample-software-requirements-specificationsrs-report-airline-database

2. Overall Description

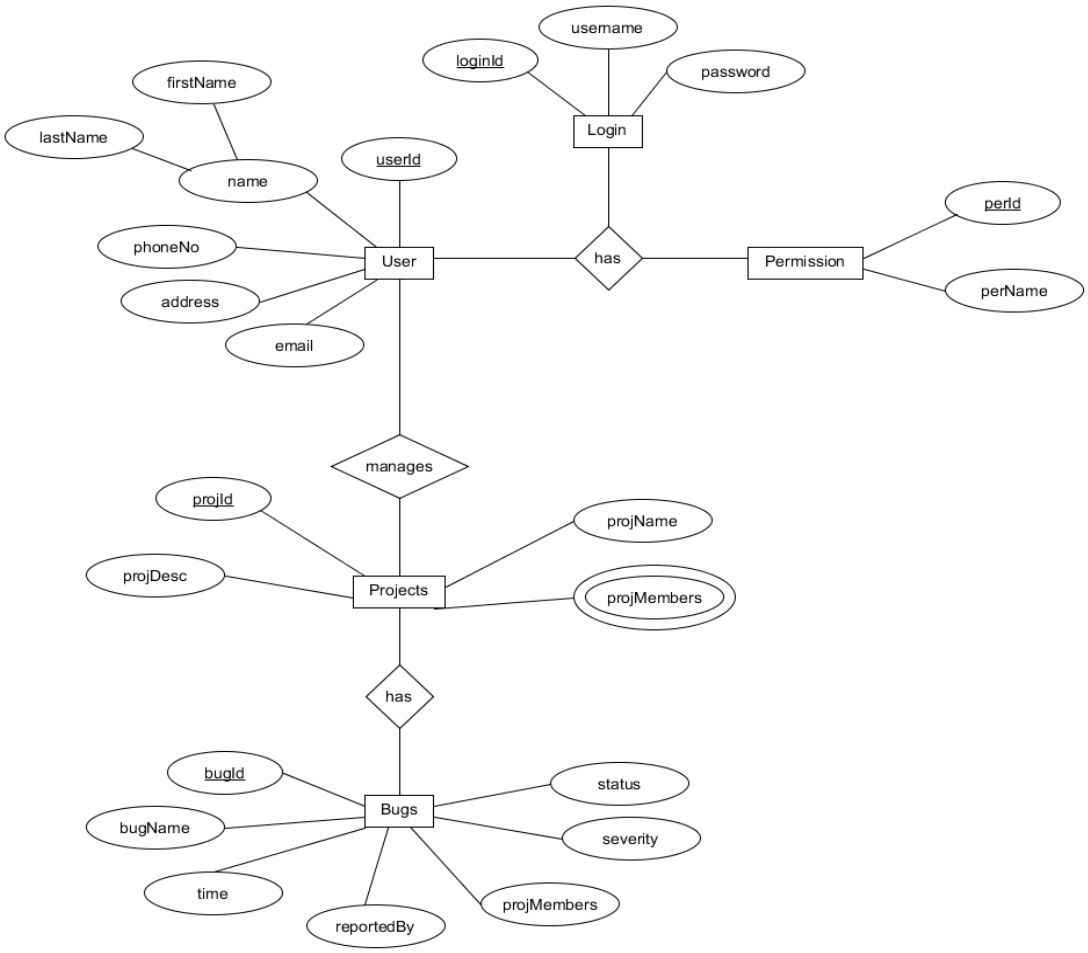
2.1 Product Perspective

The bug tracker database system stores the following information.

- Time a bug was reported
- Severity of the bug
- The program behaviour caused by the bug
- How to reproduce the bug
- The identity of the person who reported the bug
- Programmers that are working on fixing the bug
- Status of the bug

2.2 Product Features

The major features of the bug tracker system shown in the following entity-relationship model (ER model).



2.3 User Classes and Characteristics

Users of the system will be able to retrieve bug information from the secure database. Users will have access to reports of bugs that were posted by other users. Each report will provide access to more information about the bug from the database. Each project will have permissions that only some users will be able to access.

Given that the user has permission to access a project, they will have the following functionalities:

- Create a bug report.
- Add time of bug.
- Recruit project members to fix the bug.
- Label the severity of the bug.
- Update the status of the bug.

The admin will also have the following functionalities:

- Add/Delete permission to a user.

2.4 Operating Environment

The operating environment for the bug tracker system is as listed below.

- distributed database
- client/server system
- Operating system: Windows
- database: Microsoft SQL Server
- platform: ASP.Net MVC, C# and Bootstrap
- security: Auth0

2.5 Design and Implementation Constraints

1. Will follow an MVC design pattern.
2. SQL commands for queries.
3. Implement the database using a centralized database management system.

2.6 Assumption Dependencies

Let us assume that this is a distributed bug tracker system and it is used in the following application:

- A request for reporting a bug from a project, having the user fill out further bug information through a form.
- Updating the status of the bug fix.

Assuming both of the transactions are single transactions, we have designed a distributed database that is dispersed between all users for each project.

3. System Features

3.1 Functional Requirements

Description and Priority
The bug tracker system maintains bug information for each project. The bug information includes time, severity, program behaviour, how to reproduce bug, who reported bug, who is/are working on fix, and status of bug fix. This bug tracker project is of high priority because it is very difficult to complete a project if bugs are present and not fixed.

Stimulus/Response Sequences

- Browse through bug list.
- Assign user or self to a bug to fix.
- Update status of bug fix.

Functional Requirements
Other system features include:

- Distributed Database
  A distributed database implies that a single application will be able to operate transparently on data that is spread across a variety of project databases and connected by a communication network.
- Client/Server System
  The term client/server refers primarily to an architecture or logical division of responsibilities, the client is the application (also known as the front-end), and the server is the DBMS (also known as the back-end).
  A client/server system is a distributed system in which all the data resides at the server sites and all applications execute at the client sites.

4. External Interface Requirements

4.1 User Interfaces

- Front-end software: C#, ASP.Net MVC, and Bootstrap
- Back-end software: Microsoft SQL Server

4.2 Hardware Interfaces

- Windows
- A browser which supports CGI, HTML & JavaScript

4.3 Software Interfaces

Software used	**Description**
Operating system	I have chosen Windows operating system for its best support and user-friendliness.
Database	To save all of the bug information, I have chosen Microsoft SQL Server.
C#	To implement the project I have chosen the C# language for its more interactive support with ASP.Net MVC.
ASP.Net MVC	To create the basis of the system for the users.
Bootstrap	For its vast collection of templates.
Auth0	For its ease of providing a secure user interface.

4.4 Communication Interfaces

This project supports all types of web browsers. I will be using simple electronic forms for the bug reporting.

5. Nonfunctional Requirements

5.1 Performance Requirements

The steps involved to perform the implementation of the bug tracker database are as listed below.

1. ER Diagram
   The ER Diagram constitues a technique for representing the logical structure of a database in a visual way. This analysis is then used to organize data as a relation, normalizing relation and obtaining a relational database.
   - Entities: Specifies distinct real-world items in an application.
   - Attributes: Specifies properties of an entity.
   - Relationships: Connects entities and represents meaningful dependencies between them.

   Refer to 2.2 for the ER Diagram of the bug tracker database.

2. Normalization
   The basic objective of normalization is to reduce redundancy in order to have information stored only once. Storing information several time leads to wasted storage space and will increase the total size of the data stored.
   If a database is not properly designed it can give rise to modification anomalies. Modification anomalies arise when data is added to, changed or deleted from a database table. This can be elimated by normalizing a database.
   Normalization is the process of breaking down a table into smaller tables. Each table will deal with a single theme. This should be considered only after a thorough analysis and complete understanding of the implications.

5.2 Safety Requirements

If a wide portion of the database experiences extensive damage due a failure, such as a disk crash, the recovery method restores a past copy of the database that was backup up in the cloud storage. A more current state is then redone by completing the operations of committed transactions from the backed up log up to the time of failure.

5.3 Security Requirements

A secure login will permit only registered users to access the bug tracker system and to their permitted projects. This will help to prevent fraudulent attempts to spam the system with unneccessary data.

5.4 Software Quality Attributes

- **Availability**
  The bug information and status updates will be available right after they are reported by a user.
- **Correctness**
  The bug information and status updates will reach every registered project member.
- **Maintainability**
  The users should maintain correct information and status updates of the bug(s).
- **Usability**
  The bug information and status updates should help to complete a working project.