Problem Solving and Lab: C++

Final Project

Bug Tracker using C++ & SQLite

2025.06.04

22011808 Sia Lee

Motivation

1

Lack of Lightweight Bug Tracking Tools

2

Need for Local, Dependency-Free Tracking

3

Educational Purpose and Systems-Level Practice

This project aims to provide a minimal, efficient alternative that's easy to deploy and use.

1

Add bug

- Purpose: Displays all reported bugs in the system.
- Key Features:
  - Retrieves all records from the bugs table.
  - Outputs each bug’s ID, title, description, status, priority, and date.

Main Functions

2

List Bugs

- Purpose: Displays all reported bugs in the system.
- Key Features:
  - Retrieves all records from the bugs table.
  - Outputs each bug’s ID, title, description, status, priority, and date.

3

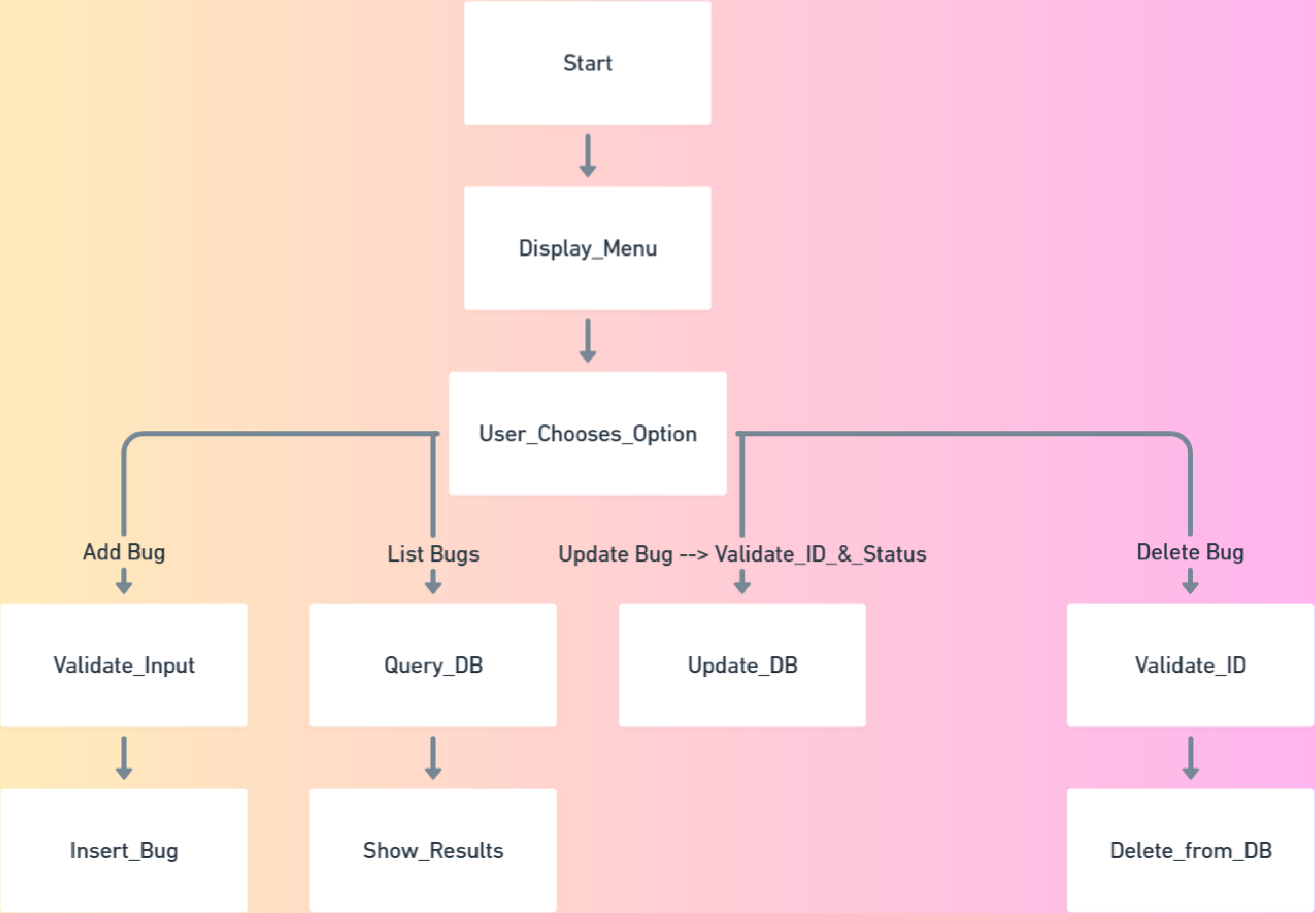
Update Bug

- Purpose: Updates the status of an existing bug.
- Key Features:
    - Validates the bug ID and new status.
    - Updates the status field of the selected bug in the database.

4

Delete Bug

- Purpose: Permanently removes a bug from the tracker.
- Key Features:
  - Verifies the bug ID exists.
  - Deletes the bug record from the database.


Start
Display_Menu
User_Chooses_Option
Add Bug
List Bugs
Update Bug --> Validate_ID_&_Status
Delete Bug
Validate_Input
Query_DB
Update_DB
Validate_ID
Insert_Bug
Show_Results
Delete_from_DB

Data Validity Checks

- Title Validation:
  - Must not be empty.
  - Maximum length: 100 characters.
- Description Validation:
  - Must not be empty.
  - Maximum length: 1000 characters.
- Priority Validation:
  - Only accepts: "Low", "Medium", or "High" (case-insensitive).

Data Validity Checks

- Status Validation:
  - Only accepts: "Open", "In Progress", or "Resolved" (case-insensitive).
- Bug ID Validation:
  - Must be a positive integer.
  - Must correspond to an existing record in the database before update/delete.
- Centralized Input Validation:
  - All user inputs are filtered through a reusable getValidInput() function for consistency and reliability.

Design Choices

- SQLite:
    - ○ Lightweight, file-based database — no server required; ideal for small-scale applications.
- C++ with SQLite C API:
    - ○ Allows fine-grained control over database interactions and teaches low-level database access.
- Console-Based UI:
- Simple input/output through terminal — keeps the focus on logic and database manipulation.

SQL Injection & Prevention

What is SQL Injection?

- SQL Injection is a security vulnerability where an attacker manipulates input to inject malicious SQL statements.

How this project prevents it:

- Uses prepared statements like sqlite3_prepare_v2() instead of raw SQL strings.
- Binds input safely with sqlite3_bind_text() — user input is treated as data, not executable SQL.
- Avoids string concatenation in queries, eliminating the risk of injection.

Future Extensions

- Accessibility to multiple users
  - Add login functionality with roles (admin, developer, tester).
  - Only authorized users can edit or delete bug reports.
  - Allow bugs to be assigned to specific developers or team members.
  - Track who is responsible for resolving each issue.
- Filter & Search Capabilities
  - Implement filtering by priority, status, or date.
  - Add keyword-based search in title or description.

Future Extensions

- Bug History / Audit Log
  - Track changes to bug status or priority over time.
  - Useful for debugging and accountability.
- GUI Interface
  - Replace the console UI with a graphical user interface using Qt or another framework for better usability.
- Email Notifications
  - Send alerts when a new bug is reported or when status changes.