**Student:** Travis Uhrig

**Capstone Project Proposal:** Command Line Kanban Board

### Description (What)

A minimalist task management kanban tool, like Trello, written in C++ for the windows command line interface. It allows the user to create and manage boards and tasks. This tool offers a simple alternative to complicated GUI tools.

### Intended User (Who and Why)

Designed for an individual's personal task management, allowing them to track the details and progress of multiple tasks across several projects.

### Data

The program accepts task-related data like task title, description, and stage. It also accepts board data like name, allowing each board to represent a different project.

### Advanced Concepts

- Classes: Utilized to define tasks and kanban boards.
- Constructors and Destructors: For creating tasks, boards, and the database, and destructing when tasks and boards are deleted, or the database is closed.
- Exception Handling: Database interactions can fail so the project will need to handle those failures and errors gracefully.

### Algorithm

The kanban program starts with the user creating or selecting a board, and in that board the user can create or manage tasks. The boards and tasks are all saved and loaded from a database. The database is updated every time a change is made to a task or board.

Tasks can be updated and moved through different stages. The transition to each new stage triggers checks to ensure the task meets the necessary criteria, with missing requirements causing an error message.

Boards, and tasks can be deleted. When a board is deleted all of its associated tasks are also deleted.

### Functionality Summary

- Two main entities exist in the program, each having its own class: Boards and Tasks. There is also a 3rd class called Database to handle saving and loading data to the SQLite DB.
- Task:
    - Tasks are added to the current board's task list when created.
    - Tasks are created with a title and start in the "To Do" stage.
    - The variable difficultyRating on tasks can be an int from 1 to 5.

    - Tasks move through, in order, the stages "To Do", "In Progress", and "Done". Tasks cannot skip a stage.
    - Tasks can be deleted.
    - Task must meet the next stage’s requirements to move to the next stage:
        - Tasks need a description and difficulty rating to enter the "In Progress" stage.
        - Tasks have no additional requirements to move to the “Done” stage.
- Board:
    - Boards can be deleted.
    - The user can switch between boards, working with one board at a time.
    - Tasks are ordered in their stages on their board by their id.
    - Deleted Boards get their Tasks deleted as well.
- Database:
    - Created Boards and Tasks are saved to a SQLite database.
    - Any updates are also saved to the SQLite database.

**Exception Handling**

Detailed exception handling protects the user experience, the program data, and the stability of the program even when unexpected errors occur. Cases handled include:

- Null Input: Stops processing and requests valid input for null user input.
- Stage Requirements: Verifies if a task meets the necessary requirements to move to a new stage.
- Database Error: Checks for successful data read/write operations to the database. In case of a failure, it notifies the user of a database error.
- Unknown Exception: In case of an unforeseen issue, it outputs error info and terminates to prevent potential damage.

**Search, Sort, Storage**

Search, sort, and storage are crucial for task management. I am using sqlite for storage, with a Boards and a Tasks table. Boards can contain many Tasks, and Tasks can belong to one Board. Using a relational database like sqlite lets me persistently store this one-to-many relationship directly with foreign keys.

Storage Relationships

```
CREATE TABLE Tasks (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        …,
        board_id INTEGER NOT NULL,
        FOREIGN KEY(board_id) REFERENCES Boards(id)
);
CREATE TABLE Boards (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        …
);
```

Sorting also happens at the database level. I order requests for tasks by stage and id, and requests for boards by title and id. When the order might change, like adding or removing tasks or boards, changing a task’s status, or changing a board’s title, I save the changes and then reload from the database with the same order by requirements. The order stays consistent, letting me correctly match user selection with selected items.

SQL Sorting

```
SELECT * FROM Boards ORDER BY title, id;

SELECT * FROM Tasks WHERE board_id = ?
        ORDER BY CASE
                WHEN stage = 'To Do' THEN 1
                WHEN stage = 'In Progress' THEN 2
                WHEN stage = 'Done' THEN 3
        END, id;
```

The UML Sequence Diagram shows how this order is maintained, with DB reloads after each change that could affect the order.

Search also utilizes the database, with the foreign key relationship between Boards and Tasks letting me search for all Tasks associated with a specific Board.

SQL Search

```
SELECT * FROM Tasks WHERE board_id = ?;
```