**Task: Build a Modular API-Database Integration System with SQLite**

**Objective**:  
You will be creating a system that interacts with an API to fetch data, stores the data in an SQLite database, and provides a command-line interface for users to interact with the stored data. Your task is to build modular CRUD functions, a user-friendly menu system, and allow basic interaction with the database.

**Steps:**

**1. Choose an API**

* Select a public API that provides data you find interesting. Some examples of APIs you can use include:
  + **PokéAPI**: Information about Pokémon (types, abilities, stats).
  + **OpenWeatherMap**: Weather data for various cities.
  + **The Dog API**: Information about different dog breeds.
  + **The Cat API**: Data on different cat breeds.
* Familiarize yourself with the API's documentation and endpoints so that you can fetch and work with the data effectively.

**2. Create an SQLite Database**

* Design an SQLite database that will store the data from the API.
* Based on the API you choose, create relevant tables to store the information. For example, if you choose **PokéAPI**, you might need a table for Pokémon with columns like id, name, height, weight, type, etc.
* Your database should allow at least the following operations:
  + **Create**: Add new records to the database.
  + **Read**: Retrieve and display stored records.
  + **Update**: Modify existing records (e.g., updating a Pokémon's weight).
  + **Delete**: Remove records from the database based on their ID.
* **Extra Goal**: Implement search functionality by name, allowing users to find specific entries without using the ID.

**3. Build Modular CRUD Functions**

* Your system should have at least the following functions:
  1. **Create**: Insert new data into the database.
  2. **Read**: Retrieve data from the database (either by ID or by name).
  3. **Update**: Modify an existing record in the database.
  4. **Delete**: Remove a record by ID.
  5. **Search by Name** (Extra goal): Allow the user to search for entries by name instead of using IDs.

These functions should be modular and reusable throughout your code, ensuring clear and organized structure. They should be called in the menu system instead of open SQLite functions.

**4. Create a Command-Line Menu System**

* Build a command-line interface (CLI) where users can interact with the system and perform CRUD operations.
* The menu should allow the following options:
  + **View all records**: Display a list of all records in the database.
  + **Search records by ID**: Allow the user to input an ID and retrieve a specific record.
  + **Search records by name**: Implement a search option that allows users to input a name and find the corresponding record.
  + **Update a record**: Allow the user to update an existing record by specifying the record’s ID.
  + **Delete a record**: Allow the user to delete a record based on its ID.
  + **Exit**: Exit the program.

Your menu should loop, allowing the user to continue interacting with the system until they choose to exit.

**5. Fetching Data from the API**

* Use the **requests** library to fetch data from the chosen API.
* Parse the API response to extract the relevant information you want to store in the SQLite database.
* After successfully fetching and parsing the data, store it in the database using your CRUD functions but **only** if should the user want to. Remember data-management is key, if the user decides against storing useless information in the database that should also be an option. **The user drives the application!**

**6. Documentation**

* Write a brief report that explains:
  1. The API you chose and the type of data it provides.
  2. The structure of the SQLite database you created (e.g., table names, column types).
  3. How the CRUD functions work and how they interact with the database.
  4. How the user interface works and how the user can interact with the system (view records, search, update, delete).

**Additional Challenges (Optional):**

* **Error Handling**: Add error handling for cases such as:
  + Failed API requests.
  + Invalid user input (e.g., entering a non-existent ID or name).
* **File I/O**: Implement functionality to save API responses to a file (e.g., JSON or CSV) and later read them from the file to populate the database.
* **SQLAlchemy**: For students who want to challenge themselves further, try using SQLAlchemy to handle the database operations instead of using SQLite directly.

**Deliverables:**

1. A **Python script** that:
   * Fetches data from the API.
   * Stores data in an SQLite database.
   * Includes modular functions for Create, Read, Update, Delete, and Search.
   * Provides a simple command-line menu to interact with the data.
2. A **documentation report** that:
   * Describes the chosen API and how the data is structured in the database.
   * Explains the CRUD functionality and how users can interact with it through the menu system.

**Assessment Criteria:**

* **Functionality**: Does the system correctly fetch and store data? Are the CRUD operations functional?
* **Modularity**: Are the database operations well-organized into functions? Are they reusable and clear?
* **User Interface**: Is the command-line interface intuitive and easy to use?
* **Creativity**: Did the student add optional features like error handling, file I/O, or SQLAlchemy?
* **Documentation**: Is the report clear, concise, and informative?