A report done by: Group 18

BOOKERY

REPORT

Group Members:

Alizada Fuad – Leader, Roles: Cyber Security and Database;

Mehdi Hasanli – Roles: Content Management and Code proofreading;

Toghrul Abdullazada – Roles: Report polishing and Code proofreading;

Tural Gadirov – Roles: Report polishing and Code proofreading;

Ilham Bakhishov – Roles: CLI design and debugging;

★ These roles were pre-defined, during the development every member contributed to every aspect of the project.

INTRODUCTION

"BOOKERY" – a bookshop management system, intended to make managing the total amount, addition and removal of extra and editing the details of already added books, sorting and filtering through by various categories and finally, writing sales and rent reports easier. When writing the code for this management system, we ensured that our objectives and goals of an intuitive text-based interface, optional quick-use numerical interface, security and safety of the implemented log-in system with various user privileges, and most importantly, shortest response time of the database were all up to our and future users' standards. Here, in this report, we will go through a quick rundown of how and why all the steps in and choices when writing the code were made.

- **PRIMARY OBJECTIVE:**

Write a working code of a Bookshop Management System in C. ✓

- **PRIMARY GOAL:**

Ensure that our management system remains unimpeded in speed even when the number of books is exceptionally high. ✓

- **SECONDARY GOALS:**
  - Implement a straightforward login mechanism for our book management system; ✓
  - Ensure that our management system is protected from malicious users and attempts to exploit it; ✓
  - Ensure that our interface is easy to understand and straightforward. ✓

SYSTEM DESIGN

System architecture and design

The system is carefully designed to accommodate a wide array of users, recognizing their diverse preferences and levels of proficiency in computing. Through providing the choice to engage through text-based commands or numerical inputs, it guarantees accessibility for individuals with varying levels of comfort and backgrounds in technology. Numerical command interface is intended for being used by new employees, while the advanced command line interface is intended for being used by experienced employees. This inclusive strategy demonstrates our dedication to ensuring technology is accessible and easy to use for everyone.

Data structure

```
// SQL statement to create books table.
const char *sql_books = "CREATE TABLE IF NOT EXISTS books ("
                        "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                        "title TEXT NOT NULL,"
                        "author TEXT NOT NULL,"
                        "genre TEXT,"
                        "price REAL,"
                        "quantity_available INTEGER,"
                        "quantity_rented INTEGER,"
                        "quantity_sold INTEGER,"
                        "quantity_rented_all INTEGER,"
                        "quantity_rented_days INTEGER"
                        ");";
```

Schema of books table in the database.

```
// SQL statement to create users table.
const char *sql_users = "CREATE TABLE IF NOT EXISTS users ("
                        "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                        "username TEXT NOT NULL,"
                        "password TEXT NOT NULL,"
                        "email TEXT NOT NULL,"
                        "role INTEGER NOT NULL"
                        ");";
```

Schema of users table in the database.

```
// SQL statement to create rents table.
const char *sql_rents = "CREATE TABLE IF NOT EXISTS rents ("
                        "id INTEGER PRIMARY KEY AUTOINCREMENT,"
                        "title TEXT NOT NULL,"
                        "Name TEXT NOT NULL,"
                        "Phone TEXT NOT NULL,"
                        "quantity_rented INTEGER,"
                        "rented_for_days INTEGER,"
                        "rent_date TEXT NOT NULL,"
                        "return_date TEXT"
                        ");";
```

Schema of rents table in the database.

Functionality

Explanation of features, with pseudocode

- **Adding new books to the inventory;**

1. Open database connection
2. If connection fails, print error message and exit
3. For each attribute (title, author, genre, price, quantity available):
   a. Prompt user to enter the attribute
   b. Validate the attribute until it's valid
4. Set quantity rented and sold to 0
5. Construct SQL query to insert new book into the database
6. Execute the SQL query
7. Close the database connection

- **Displaying all books in the inventory;**

1. Open database connection
2. If connection is successful:
   - Prepare SQL statement to select all columns from the 'books' table
   - If statement preparation is successful:
     - Initialize variables to store maximum widths and values for each column
     - Iterate over each row fetched by the SQL statement:
       - Update maximum widths and values for each column
     - Print a line of dashes for formatting

        - Print column headers
        - Print another line of dashes for formatting
        - Reset the SQL statement to re-execute
        - Iterate over each row fetched by the SQL statement:
            - Print book data with appropriate formatting
            - Print a line of dashes for formatting
        - Finalize the SQL statement
    - Close the database connection

- **<u>Searching for books by title, author and/or genre;</u>**

1. Open database connection
2. Prompt user to enter search term (title, author, or genre)
3. Prepare SQL statement to select title, author, genre, price, quantity_available, quantity_rented, and quantity_sold from books table based on search term
4. Initialize variables for maximum widths of title, author, genre, price, quantity_available, quantity_rented, and quantity_sold
5. Bind search term to the prepared statement
6. Fetch data to calculate maximum widths for each column
7. Reset the statement to re-execute
8. Print search results header with aligned columns
9. Iterate through search results:
    - If title matches search term:
        - Print row with title highlighted in green, aligned with other columns

- If author matches search term:
  - Print row with author highlighted in green, aligned with other columns
- If genre matches search term:
  - Print row with genre highlighted in green, aligned with other columns

10. Finalize statement and close database connection

- **Updating book details;**

1. Open database connection
2. If opening database fails, print error message and exit
3. Prompt user to enter the title of the book to update until a valid title is entered
4. Initialize a struct to hold the updated book details
5. Prompt user to enter new title until a valid title is entered
6. Prompt user to enter new author, genre, price, and quantity available
7. Construct SQL UPDATE statement to update book details
8. Prepare SQL statement with placeholders for parameters
9. Bind values to the prepared statement placeholders
10. Execute the statement
11. If execution fails, print SQL error message
12. If execution succeeds, print success message
13. Finalize statement and close database connection

- **Selling books to customers;**

1. Open database connection

2. If opening database fails, print error message and exit
3. Prompt user to enter the title of the book to sell
4. Prompt user to enter the quantity to sell
5. Construct SQL UPDATE statement to update quantity_sold and quantity_available for the specified book
6. Execute the SQL statement using sqlite3_exec
7. If execution fails, print SQL error message
8. If execution succeeds, print success message
9. Close database connection

- **Viewing sales reports.**

1. Open database connection
2. If opening database fails, print error message and exit
3. Print header for sales report
4. Prepare SQL statement to select book details from the database
5. Calculate maximum widths for each column based on fetched data
6. Print header row with aligned columns
7. Reset the statement to re-execute
8. Initialize totalRevenue variable to 0
9. Iterate through fetched data:
    a. Extract title, author, genre, price, and quantity_sold for each book
    b. Calculate revenue for each book (price * quantity_sold)
    c. Print book details with aligned columns, including revenue

d. Add revenue to totalRevenue

e. Print horizontal line separator

10. Print total revenue

11. Finalize statement and close database connection

Implementation Details

Overview of the programming languages, libraries, and tools used

- **Programming Languages:**
  - C
- **Libraries:**
  - sqlite3.h
    - SQLite - a C-language library that provides a lightweight disk-based database. It doesn't require a separate server process and allows access to the database using a nonstandard variant of the SQL query language.
  - openssl/evp.h
    - OpenSSL - its EVP (Envelope) library provides a high-level interface for cryptographic operations, perfect for our needs. It supports a wide range of cryptographic algorithms and functions, from which we used SHA-256 hashing algorithm.

```
 * @param password The password to be hashed.
 * @param hash     The array to store the resulting hash.
 *
 * @return void
 */
void hashPassword(const char *password, unsigned char *hash) {
    EVP_MD_CTX *mdctx;       // Message digest context.
    const EVP_MD *md;        // Message digest type.
    unsigned int md_len;     // Length of the message digest.

    // Select the SHA-256 message digest.
    md = EVP_sha256();

    // Create a new message digest context.
    mdctx = EVP_MD_CTX_new();

    // Initialize the message digest context with the selected message digest.
    EVP_DigestInit_ex(mdctx, md, NULL);

    // Update the message digest context with the password data.
    EVP_DigestUpdate(mdctx, password, strlen(password));

    // Finalize the message digest and store the resulting hash in the 'hash' array.
    EVP_DigestFinal_ex(mdctx, hash, &md_len);

    // Free the message digest context.
    EVP_MD_CTX_free(mdctx);
}
```

Challenges Encountered During Implementation and Solutions

1. **Challenge: Storing the data**
    - Storing large amounts of data in a text file is very inefficient

    Solution:
    - Use the sqlite3 database
    - Data stored in a categorized database allowed more speed, ease of use and efficiency
2. **Challenge: Implementing the renting system**
    - Book shop needs to be able to contact customers if their rent is overdue
    - Rent records and available book number should be updated when the book is returned

    Solution:
    - Customers' names and phone numbers are noted in the rent records
    - Create rent recall function to delete the rent record and update available book information
3. **Challenge: Implementing a secure user log-in system**
    - Storing usernames and passwords without any hashing algorithm is risky
    - Not all employees should have admin privileges over the system

    Solution:
    - Use OpenSSL's hashing algorithm

- With this measure taken, we can now store not the plain text passwords, but their hashed forms
- Implement least privilege principle which suggests that users should only have the minimal level of access or permission to perform their tasks
- Implement role-based access control (RBAC) method to restrict access based on roles of individual employees

Sample Usage

```
bms-> bms-> search rent
Enter search term (title, name, or phone): 0773127845

***** Search Results ******
-------------------------------------------------------------------------------------------------------------
 Id      | Title        | Name      | Phone      | Quantity Rented | Rented for Days | Rent Date          | Return Date |
-------------------------------------------------------------------------------------------------------------
3        | The Stranger | Ilham Bak | 0773127845 | 1               | 10              | 12/05/2024         | 22/05/2024  |
-------------------------------------------------------------------------------------------------------------
8        | Fiction      | Ilham Bak | 0773127845 | 1               | 6               | 12/05/2024         | 18/05/2024  |
-------------------------------------------------------------------------------------------------------------
17       | Don Quixote  | Ilham Bak | 0773127845 | 1               | 40              | 12/05/2024         | 21/06/2024  |
-------------------------------------------------------------------------------------------------------------
bms-> bms->
```

```
bms-> search book
Enter search term (title, author, or genre): Fiction

***** Search Results ******
-----------------------------------------------------------------------------------------------------------
Title           | Author          | Genre   | Price      | Quantity Available | Quantity Rented    | Quantity Sold |
-----------------------------------------------------------------------------------------------------------
The Jungle Book | Rudyard Kipling | Fiction | $30.00     | 81                 | 0                  | 19            |
-----------------------------------------------------------------------------------------------------------
The Gadfly      | Ethel Lilian    | Fiction | $9.00      | 90                 | 1                  | 9             |
-----------------------------------------------------------------------------------------------------------
Fiction         | Alice Munro     | Story   | $5.00      | 9                  | 1                  | 0             |
-----------------------------------------------------------------------------------------------------------
bms-> bms->
```

```
bms-> bms-> search rent
Enter search term (title, name, or phone): Ilham Bak

***** Search Results ******
-------------------------------------------------------------------------------------------------------------
 Id      | Title        | Name      | Phone      | Quantity Rented | Rented for Days | Rent Date          | Return Date |
-------------------------------------------------------------------------------------------------------------
3        | The Stranger | Ilham Bak | 0773127845 | 1               | 10              | 12/05/2024         | 22/05/2024  |
-------------------------------------------------------------------------------------------------------------
8        | Fiction      | Ilham Bak | 0773127845 | 1               | 6               | 12/05/2024         | 18/05/2024  |
-------------------------------------------------------------------------------------------------------------
17       | Don Quixote  | Ilham Bak | 0773127845 | 1               | 40              | 12/05/2024         | 21/06/2024  |
-------------------------------------------------------------------------------------------------------------
bms-> bms->
```

```
bms-> bms-> search book
Enter search term (title, author, or genre): Robert Greene

***** Search Results ******
-----------------------------------------------------------------------------------------------------------
Title               | Author        | Genre     | Price      | Quantity Available | Quantity Rented    | Quantity Sold |
-----------------------------------------------------------------------------------------------------------
The 48 Laws of Power | Robert Greene | Self-help | $20.00     | 59                 | 0                  | 41            |
-----------------------------------------------------------------------------------------------------------
The Art of Seduction | Robert Greene | Self-help | $18.99     | 31                 | 0                  | 0             |
-----------------------------------------------------------------------------------------------------------
bms-> bms->
```

```
bms-> bms-> add book
Enter title: The Art of Seduction
Enter author: Robert Greene
Enter genre: Self-help
Enter price: 18.99
Enter quantity available: 31
Book added successfully.
```

```
bms-> bms-> del book
You don't have permission for this action!
 This incident will be reported.
bms-> whoami
mehdi : You are a user.
bms-> 
```

```
bms-> login
Enter username: admin
Enter password:
Authentication successful!
bms-> bms-> whoami
admin : You are an admin.
bms-> 
```

Testing

We had a to-do list while developing Bookery. We put all of the code's shortcomings into the list, and periodically checked the list to try and fix the errors and vulnerabilities as we continued developing the code.

```
sqlite3:

https://www.tutorialspoint.com/sqlite/sqlite_c_cpp.htm


# TODO:

SQLI in sellBook() <--- fix this vulnerability.
Fix updateUser() function : --> segmentation error ---> DONE
recall rented books                                ---> DONE
validate phone number
SQLI in rentBook <--- Fix this vulnerability.
Update sellBook() and rentBook() functions , if quantity_available is 0 , shouldnt be able to sell or rent.
```

Additionally, we had a tests folder where we tested different functions before incorporating them into the project.

```c
#include <stdio.h>
#include <time.h>

int main() {
    char date_string[11];
    time_t t = time(NULL);
    struct tm *today = localtime(&t);

    strftime(date_string, sizeof(date_string), "%d/%m/%Y", today);

    printf("Today's date is: %s\n", date_string);

    return 0;
}
```

Additional materials and documentation

User manuals

- SQLite user manual
- OpenSSL user manual

Code samples

```c
// Who am i

/**
 * @brief Prints the role of the current user.
 *
 * @details This function prints whether the current user is an admin or a regular user.
 *          It uses ANSI escape codes for colored output.
 *
 * @param None
 *
 * @return None
 */
void whoami(){
    // Check if the user role is admin (0) or not.
    if(userRole == 0){
        // Print the user name and indicate that they are an admin.
        printf("%s : You are an %sadmin.%s\n",userName,GREEN,RESET);
    } else {
        // Print the user name and indicate that they are a regular user.
        printf("%s : You are a %suser%s.\n",userName,GREEN,RESET);
    }
}
```

```c
void addUser() {
        password2Ptr = getpass("Enter password again: ");
        strcpy(password2, password2Ptr);
        if(strcmp(password,password2) != 0){
            printf("%sPasswords don't match!%s\n",RED,RESET);
        } else {
            // If passwords match, store the password in newUser struct.
            strcpy(newUser.password, password);
        }
    } while (!validatePassword(newUser.password));

    // Prompt the admin to enter the email.
    do {
        printf("Enter email: ");
        scanf("%99s", newUser.email);
    } while (!validateEmail(newUser.email));

    // Prompt the admin to enter the role (0 for admin, 1 for regular user).
    do {
        printf("Enter role (0 for admin, 1 for regular user): ");
        scanf("%d", &newUser.role);
    } while (!validateRole(newUser.role));

    // Hash the password using SHA-256 algorithm.
    unsigned char hashed_password[SHA256_DIGEST_LENGTH];
    hashPassword(newUser.password, hashed_password);

    // Convert hashed password to hexadecimal string.
    char hashed_password_str[SHA256_DIGEST_LENGTH * 2 + 1];
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++) {
        sprintf(&hashed_password_str[i * 2], "%02x", hashed_password[i]);
    }

    // Construct SQL query to insert new user into the database.
    char sql[1000];
    sprintf(sql, "INSERT INTO users (username, password, email, role) VALUES (?, ?, ?, ?);");

    // Prepare SQL statement
    sqlite3_stmt *stmt;
```

```c
    // Input validation loop for title.
    do {
        printf("Enter title: ");
        scanf(" %[^\n]", newBook.title);
    } while (!validateTitle(newBook.title));

    // Input validation loop for author.
    do {
        printf("Enter author: ");
        scanf(" %[^\n]", newBook.author);
    } while (!validateAuthor(newBook.author));

    // Input validation loop for genre.
    do {
        printf("Enter genre: ");
        scanf(" %[^\n]", newBook.genre);
    } while (!validateGenre(newBook.genre));

    // Input validation loop for price.
    do {
        printf("Enter price: ");
        scanf("%f", &newBook.price);
    } while (!validatePrice(newBook.price));

    // Input validation loop for quantity available.
    do {
        printf("Enter quantity available: ");
        scanf("%d", &newBook.quantity_available);
    } while (!validateQuantity(newBook.quantity_available));
    newBook.quantity_rented = 0;
    newBook.quantity_sold = 0;

    char sql[1000]; ///< SQL query string.
    sprintf(sql, "INSERT INTO books (title, author, genre, price, quantity_available, quantity_rented, quantity_sold, quantity_rented_
            newBook.title, newBook.author, newBook.genre, newBook.price, newBook.quantity_available, newBook.quantity_rented, newBook.
```

```c
void displayBooks() {
    sqlite3 *db; // SQLite database connection.
    sqlite3_stmt *stmt; // SQLite statement.
    int return_code; // Return code for SQLite operations.

    // Open the SQLite database.
    return_code = sqlite3_open(DATABASE_FILE, &db);
    if (return_code != SQLITE_OK) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return;
    }

    printf("\n********** List of Books **************\n");

    // SQL query to select book information.
    const char *sql = "SELECT title, author, genre, price, quantity_available, quantity_rented, quantity_sold FROM books;";

    // Prepare the SQL statement.
    return_code = sqlite3_prepare_v2(db, sql, -1, &stmt, 0);
    if (return_code != SQLITE_OK) {
        fprintf(stderr, "Failed to execute statement: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return;
    }

    // Calculate maximum widths for each column.
    int max_title_width = 6;
    int max_author_width = 6;
    int max_genre_width = 0;
    double max_price = 0.0;
    int max_qty_available = 0;
    int max_qty_rented = 0;
    int max_qty_sold = 0;

    // Iterate through the result set to find maximum widths.
    while ((return_code = sqlite3_step(stmt)) == SQLITE_ROW) {
```

```c
void searchBook() {
    sqlite3 *db; // SQLite database connection.
    sqlite3_stmt *stmt; // SQLite statement.
    int return_code; // Return code for SQLite operations.

    // Open the SQLite database.
    return_code = sqlite3_open(DATABASE_FILE, &db);
    if (return_code) {
        fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return;
    }

    char searchTerm[MAX_TITLE_LENGTH];
    printf("Enter search term (title, author, or genre): ");
    scanf(" %[^\n]s", searchTerm);

    // SQL query to search for books based on the search term.
    const char *sql = "SELECT title, author, genre, price, quantity_available, quantity_rented,\
        quantity_sold FROM books WHERE title LIKE ? OR author LIKE ? OR genre LIKE ?;";

    // Prepare the SQL statement.
    return_code = sqlite3_prepare_v2(db, sql, -1, &stmt, 0);
    if (return_code != SQLITE_OK) {
        fprintf(stderr, "Failed to execute statement: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return;
    }

    // Calculate maximum widths for each column.
    int max_title_width = 0;
    int max_author_width = 0;
    int max_genre_width = 0;
    double max_price = 0.0;
    int max_qty_available = 0;
    int max_qty_rented = 0;
```