THOMPSON RIVERS UNIVERSITY

Department of Engineering Faculty of Science

SENG 3210 – Applied Software Engineering
Inked System Testing Report

Isaac Latta T00239008

Matia Landry T00707575

March 24, 2025

1. List of Test Cases

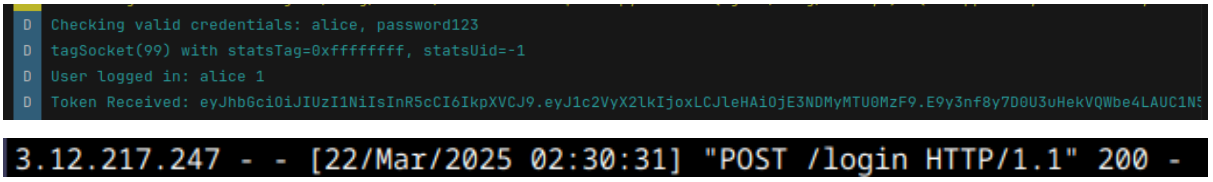
Test Case ID	**Title**
TC_01	Backend Functionality Testing
TC_02	Book Operations Testing
TC_03	Friend Operations Testing
TC_04	Reading Operations Testing
TC_05	Backend Isolation Testing
TC_06	Android Emulator End-to-End Integration Test

2. Detailed Report for the Test Cases

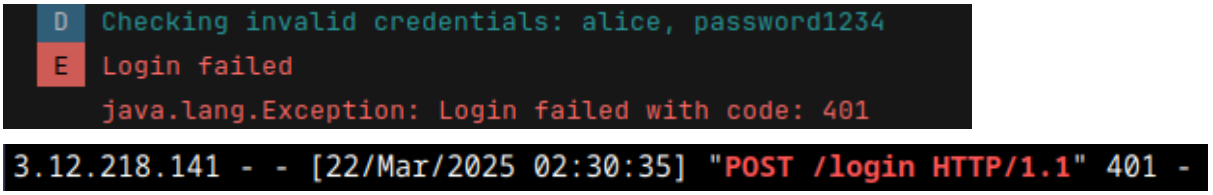
a. Test Case: TC_01 – Backend Functionality Testing

Test case ID	TC_01
Title	Backend Functionality Testing
Description	This test case validates the core backend services. It includes testing login functionality (both with valid and invalid credentials) and verifying that the book search function returns results for various queries.
Precondition	● The backend server must be up and running (or simulated via test_server.sh for isolation). ● The test user “alice” with password “password123” is registered.
Preconditions	● A valid token is returned upon successful login. ● Login with invalid credentials is rejected. ● Search results (or appropriate error messages) are returned for valid and invalid book search queries.

Postconditions	• A valid token is returned upon successful login. • Login with invalid credentials is rejected. • Search results (or appropriate error messages) are returned for valid and invalid book search queries.
Running Steps	• Start the backend. • Execute the BackendTest suite on the Android emulator. • Observe logs for “LOGIN_TEST” which indicate login attempts with both correct and incorrect passwords. • Run book search tests by submitting several search queries (e.g., “harry potter”, “Interesting books”, and invalid input). • Wait for each asynchronous operation (using CountDownLatch) to complete within the 5-second timeout.
Actual Output	• Log messages confirm a successful login with the valid credentials, and a token is received (as logged by subsequent tests). • Invalid login attempts are logged. • Book search results (or error responses) are printed in the log output.



Screenshot of successful login.

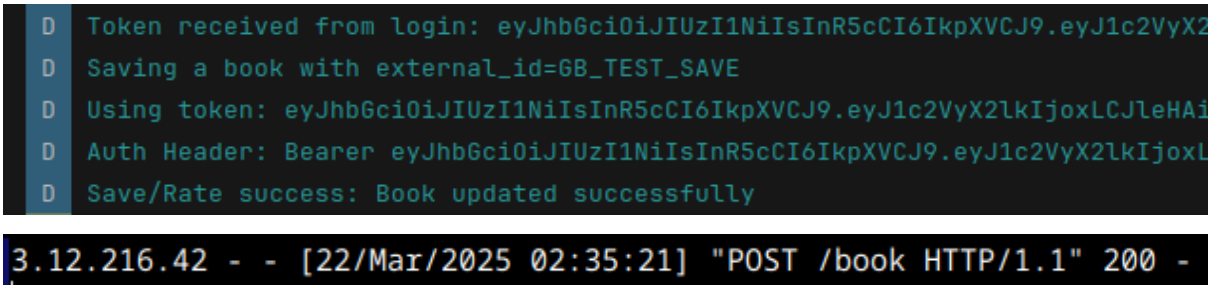


Screenshot of failed login, status code 401 Unauthorized.

b. Test Case: TC_02 – Book Operations Testing

Test case ID	TC_02

Title	Book Operations Testing
Description	This test case covers operations related to saving and rating books. It ensures that the system correctly handles saving a book, rating a book, and performing both actions in a single operation.
Precondition	• The user “alice” must be logged in successfully (as handled in the @Before setup method). • A valid token is obtained and stored. • The start_backend script must be executed on the server to properly seed the database.
Postconditions	• For a “save” operation, the book with the given external ID is stored in the user’s list. • For a “rate” operation, the correct rating is applied to the book. • For combined operations, both saving and rating are executed.
Running Steps	• Run the BookOperationsTest suite on the emulator. • Check the logs for “BOOK_OPS_TEST” to confirm the actions: saving a book with external_id=GB_TEST_SAVE, rating a book with external_id=GB_TEST_RATE (rating=5), and saving & rating with external_id=GB_TEST_BOTH (rating=3). • Allow each operation up to 5 seconds to complete.
Actual Output	• Log outputs display confirmation messages for each action (e.g., “Saving a book…”, “Rating a book…”). • A token is logged during the setup phase, indicating a successful login.



Screenshot of book operation test logs

c. Test Case: TC_03 – Friend Operations Testing

Test case ID	TC_03
Title	Friend Operations Testing
Description	This test case evaluates the functionality associated with friend-related operations. It tests sending friend requests, removing friends, searching for friends, listing current friends, listing incoming friend requests, and handling (e.g., approving) friend requests.
Precondition	● The user “alice” is logged in and holds a valid session token. ● The system contains other user profiles (e.g., “bob”, “frank”) for performing friend operations. ● The start_backend script must be executed on the server to properly seed the database
Postconditions	● Friend requests are sent or removed as indicated. ● Search operations return the expected user(s). ● Lists of friends and friend requests are updated accordingly.
Running Steps	Execute the FriendOperationsTest suite on the emulator. Monitor log outputs: ● For sending a friend request to “bob”. ● For removing friend “bob”. ● For searching using query “alice” (which should return a set of related user profiles). ● For listing friends and friend requests. ● For handling (approving) a friend request from user “frank”.
Actual Output	● Log entries confirm that a friend request is sent and that removal is attempted. ● Search and list operations display the expected friend data. ● Handling of the friend request (approval) is logged with appropriate user identification.

```
D  User logged in: alice 1
D  Token Received: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjox
D  Found 1 groups.
D  Total friends: 0
D  Pending requests: 2
```

```
D  Approving friend request from frank
D  Request handled: Friend request approved
D  Total friends: 1
D  Pending requests: 2
```

```
D  Removing friend bob
D  Friend removed: Friend removed successfully
```

```
3.12.217.148 - - [22/Mar/2025 02:40:06] "POST /friend/request HTTP/1.1" 200 -
3.12.216.113 - - [22/Mar/2025 02:40:06] "GET /friend/list HTTP/1.1" 200 -
3.12.216.187 - - [22/Mar/2025 02:40:06] "GET /friend/requests HTTP/1.1" 200 -
```

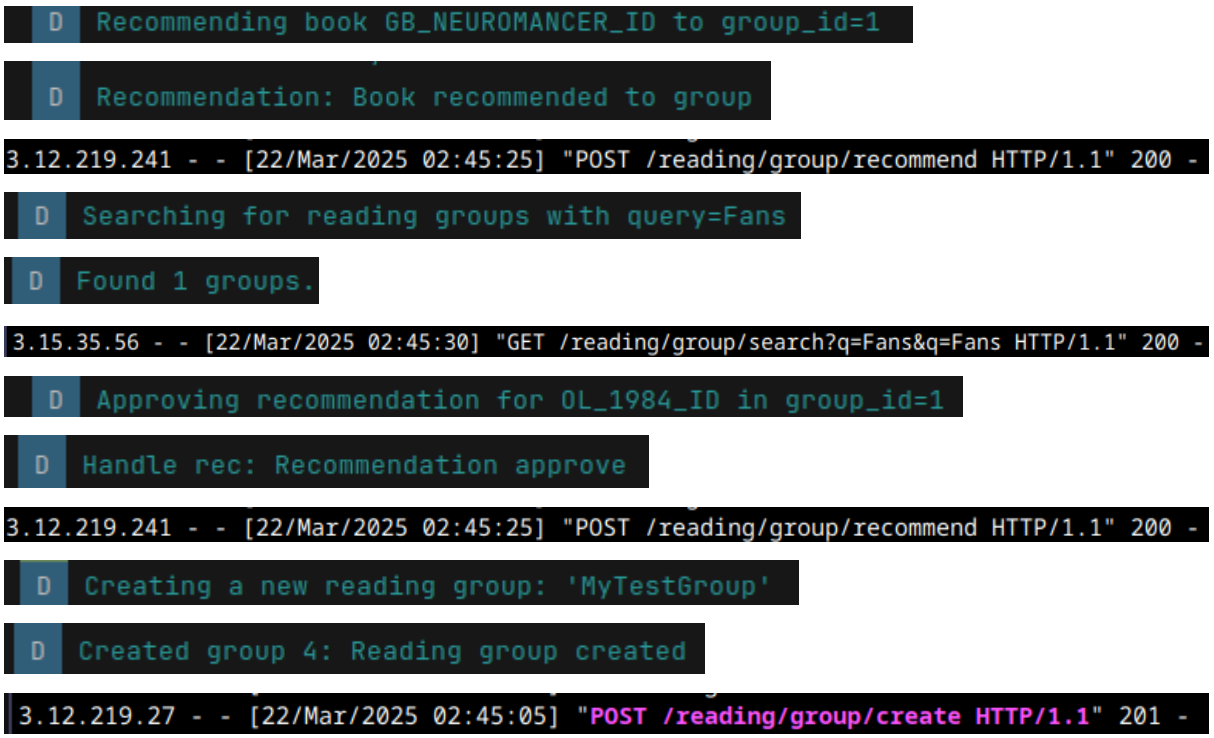
Screenshots of server logs and logcat showing both client and server side friend operation results.

d. Test Case: TC_04 – Reading Group Operations Testing

Test case ID	TC_04
Title	Reading Group Operations Testing
Description	This test case verifies the reading group features of the application. It covers creating a reading group, joining a group, recommending books to a group, searching for groups, promoting a group member, handling book recommendations, and listing both user groups and group recommendations.
Precondition	• User “alice” is authenticated. • The backend has pre-configured reading group data (or groups are created during testing). • The start_backend script must be executed on the server to properly seed the database
Postconditions	• A new reading group is created and its details recorded. • The user is successfully added to an existing group. • Recommendations are processed (approved or rejected) as required. • Updated lists for groups and recommendations are available.
Running Steps	• Run the ReadingOperationsTest suite on the emulator.

	• Observe the following steps in sequence:
	  ○ Creation of a reading group named “MyTestGroup”.
	  ○ Joining an existing group (using groupId 3).
	  ○ Recommending a book.
	  ○ Searching for groups using the query “Fans”.
	  ○ Promoting a member (user_id 3) within group (groupId 1).
	  ○ Handling a book recommendation (approval for externalId “OL_1984_ID”).
	  ○ Listing user groups and group recommendations.
	• Each step uses a CountDownLatch to ensure asynchronous operations complete (timeout set to 5 seconds).

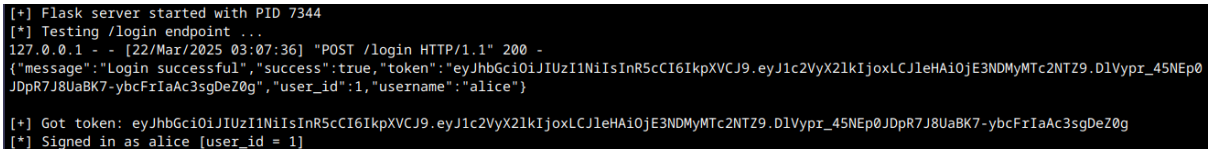
Actual Output	• Log messages confirm each operation: group creation, joining, recommending, searching, promotion, handling recommendations, and listing operations. • Console logs provide evidence of successful execution and state changes in the system.



Screenshots of logcat and server logs showing the results of the tests.

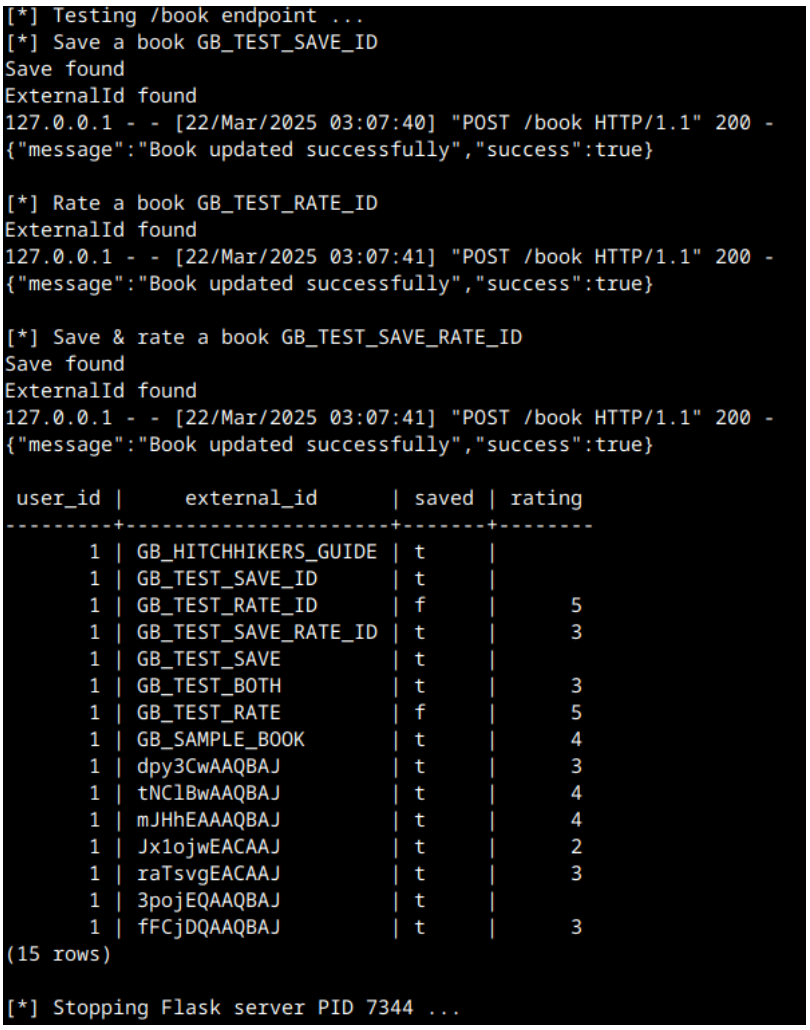
e. Test Case: TC_05 – Backend Isolation Testing

Test case ID	TC_05
Title	Backend Isolation Testing
Description	This test case focuses on testing the backend services in isolation using the provided shell script (test_server.sh). The goal is to validate that core backend functionalities (such as API endpoints) work correctly without the Android
Precondition	• The backend environment is set up in isolation(done in test_server.sh). • Necessary environment variables and configuration parameters (API keys, endpoints) are correctly set.
Postconditions	• API endpoints return the expected responses for the given inputs. • Errors or failures are correctly logged and handled.
Running Steps	• Execute the test_server.sh script from the command line. • Monitor the output for messages related to API testing (e.g., authentication, data retrieval, or error handling). • Verify that the responses and logs indicate proper handling of both valid and invalid requests.
Actual Output	• The command-line output provides a detailed log of the backend's behavior during testing. • Successful responses and error messages (where applicable) are logged.



```
[*] Beginning tests for group operations.
[*] Testing create_group ...
127.0.0.1 - - [22/Mar/2025 03:07:39] "POST /reading/group/create HTTP/1.1" 201 -
{"group_id":4,"message":"Reading group created","success":true}

[*] Testing search_groups ...
127.0.0.1 - - [22/Mar/2025 03:07:39] "GET /reading/group/search?q=Fans HTTP/1.1" 200
{"groups":[{"created_by":1,"group_id":1,"group_name":"Sci-Fi Fans"}],"success":true}

[*] Testing join_group ...
127.0.0.1 - - [22/Mar/2025 03:07:39] "POST /reading/group/join HTTP/1.1" 200 -
{"message":"Reading group joined","success":true}

[*] Testing recommend_book_to_group...
127.0.0.1 - - [22/Mar/2025 03:07:39] "POST /reading/group/recommend HTTP/1.1" 200 -
{"message":"Book recommended to group","success":true}

[*] Testing promote_group_member...
127.0.0.1 - - [22/Mar/2025 03:07:39] "POST /reading/group/promote HTTP/1.1" 200 -
{"message":"Member promoted to admin","success":true}

[*] Testing recommend_book_to_group...
127.0.0.1 - - [22/Mar/2025 03:07:39] "POST /reading/group/recommend HTTP/1.1" 200 -
{"message":"Book recommended to group","success":true}
```

```
[*] Friend operation tests begin ...
[*] Testing /friend/list ...
127.0.0.1 - - [22/Mar/2025 03:07:40] "GET /friend/list HTTP/1.1" 200 -
{"friends":[{"user_id":2,"username":"bob"}],"success":true}

[*] Testing /friend/requests ...
127.0.0.1 - - [22/Mar/2025 03:07:40] "GET /friend/requests HTTP/1.1" 200 -
{"requests":[{"user_id":5,"username":"heidi"},{"user_id":4,"username":"dave"}],"success":true}

[*] Approving friend request from heidi (user_id=5) ...
127.0.0.1 - - [22/Mar/2025 03:07:40] "POST /friend/request HTTP/1.1" 200 -
{"message":"Friend request approved","success":true}

[*] Denying friend request from dave (user_id=4) ...
127.0.0.1 - - [22/Mar/2025 03:07:40] "POST /friend/request HTTP/1.1" 200 -
{"message":"Friend request denied","success":true}

[*] Testing /friend/search ...
127.0.0.1 - - [22/Mar/2025 03:07:40] "GET /friend/search?q=char HTTP/1.1" 200 -
{"success":true,"users":[{"user_id":3,"username":"charlie"}]}

[*] Removing friend bob (user_id=2) ...
127.0.0.1 - - [22/Mar/2025 03:07:40] "POST /friend/remove HTTP/1.1" 200 -
{"message":"Friend removed successfully","success":true}

[*] Listing updated friends after removal...
127.0.0.1 - - [22/Mar/2025 03:07:40] "GET /friend/list HTTP/1.1" 200 -
{"friends":[{"user_id":5,"username":"heidi"}],"success":true}

[*] Friend operation tests complete.
```



```
[*] Testing /book endpoint ...
[*] Save a book GB_TEST_SAVE_ID
Save found
ExternalId found
127.0.0.1 - - [22/Mar/2025 03:07:40] "POST /book HTTP/1.1" 200 -
{"message":"Book updated successfully","success":true}

[*] Rate a book GB_TEST_RATE_ID
ExternalId found
127.0.0.1 - - [22/Mar/2025 03:07:41] "POST /book HTTP/1.1" 200 -
{"message":"Book updated successfully","success":true}

[*] Save & rate a book GB_TEST_SAVE_RATE_ID
Save found
ExternalId found
127.0.0.1 - - [22/Mar/2025 03:07:41] "POST /book HTTP/1.1" 200 -
{"message":"Book updated successfully","success":true}

 user_id |     external_id      | saved | rating
---------+----------------------+-------+--------
       1 | GB_HITCHHIKERS_GUIDE | t     |
       1 | GB_TEST_SAVE_ID      | t     |
       1 | GB_TEST_RATE_ID      | f     |      5
       1 | GB_TEST_SAVE_RATE_ID | t     |      3
       1 | GB_TEST_SAVE         | t     |
       1 | GB_TEST_BOTH         | t     |      3
       1 | GB_TEST_RATE         | f     |      5
       1 | GB_SAMPLE_BOOK       | t     |      4
       1 | dpy3CwAAQBAJ         | t     |      3
       1 | tNClBwAAQBAJ         | t     |      4
       1 | mJHhEAAAQBAJ         | t     |      4
       1 | Jx1ojwEACAAJ         | t     |      2
       1 | raTsvgEACAAJ         | t     |      3
       1 | 3pojEQAAQBAJ         | t     |
       1 | fFCjDQAAQBAJ         | t     |      3
(15 rows)

[*] Stopping Flask server PID 7344 ...
```

Screenshots of TC_05, test_server.sh scripts outputs. Verifying the api updates the database and responds correctly. I have left out some test screenshots as the output of the script is quite verbose, and would result in this report being much too long.

Note: All tests (aside from TC_05) require the start_backend script to be run on the server first. This script seeds the database, starts the backend api, and listens for app connections.
Due to our need for rapid development, instead of unit testing we jumped straight to instrumented tests, and to avoid the complexities of mock implementation, we just simply inspected the logcat/server logs and database state to verify the tests. These tests are designed to verify the end to end—client to server actions. Our ui was tested manually(i.e. by us using it). TC_05 is done by simply running test_server.sh, which seeds the database, starts the flask api, and sends curl requests to the various api endpoints over localhost.