| **Coursework #3** | |
| --- | --- |
| **Module’s Information:** | |
| Module | PRG4203 MOBILE APPLICATION DEVELOPMENT  IBM4203 MOBILE APPS DEVELOPMENT USING ANDROID |
| Session | JAN 2025 |
| Programme | BCSI / BITI |
| Lecturer | Lai Mei Yoon |
| Coursework Type | Project (Group of 3 or 4) |
| Percentage | 40% |
| Hand-out Date | Week 7 |
| Due Date | Week 14 |
| **Student’s Declaration:** | |
| ***We declare that:***   * ***We understand what is meant by plagiarism*** * ***This assignment is all our own work and we have acknowledged any use of the published or unpublished works of other people.*** * ***We hold a copy of this assignment which we can produce if the original is lost or damaged***  | **Name** | **ID** | **Signature** | **Marks** | | --- | --- | --- | --- | | 1. **Thanuja Sekuri** | **I25032161** | **thanuja** |  | | 1. **Bhulakshmi Ainala** | **I25032054** | **bhulakshmi** |  | | 1. **Shrey Rastogi** | **I25032149** | **shrey** |  | |  |  |  |  | | |

| **Learning Outcomes Assessed:** | |
| --- | --- |
| LO2 | Demonstrate an appropriate user interface design and logic using variety of Android programming techniques. (P5, PLO3) |
| **Penalty for late submission:** | |
| 1 day – minus 20% of total mark awarded.  2 days – minus 50% of total mark awarded.  3 days – 0 mark for this piece of coursework. | |

**Rubric**

| **Criteria** | **Excellent** | **Good** | **Satisfactory** | **Needs Improvement** | **Your Score** |
| --- | --- | --- | --- | --- | --- |
| **Functionality (40)** | All required features are implemented and work flawlessly. **(31-40)** | Most features are implemented and work with minor issues. **(21-30)** | Some features are implemented, but there are significant issues. **(11-20)** | Few features are implemented, and there are major issues. **(1-10)** |  |
| **UI/UX Design (16)** | The app has an intuitive, user-friendly interface with excellent visual design. **(13-16)** | The app has a good interface with minor usability issues. **(9-12)** | The app has a basic interface with several usability issues. **(5-8)** | The app has a poor interface with major usability issues. **(1-4)** |  |
| **Creativity and Innovation (8)** | Demonstrates creativity and innovation in app design and functionality. **(7-8)** | Shows some creativity and innovation. **(5-6)** | Basic app design with limited creativity. **(3-4)** | Lacks creativity and innovation. **(1-2)** |  |
| **Code Quality (8)** | Code is clean, well-organized, and follows best practices. **(7-8)** | Code is mostly clean and organized, with minor deviations from best practices. **(5-6)** | Code is somewhat organized but has several issues and does not follow best practices. **(3-4)** | Code is poorly organized and does not follow best practices. **(1-2)** |  |
| **Documentation (20)** | Well-written and well-organized report. It contains all the required information. **(16-20)** | Clear and organized report with minor details missing. **(11-15)** | Basic report with some organization. Some required information is missing. **(6-10)** | Incomplete or poorly organized report. **(1-5)** |  |
| **Presentation (8)** | Engaging and well-structured presentation. Effectively communicates design. **(7-8)** | Clear and structured presentation. Communicates design well. **(5-6)** | Basic presentation with some structure. Communicates design adequately. **(3-4)** | Incomplete or poorly structured presentation. Fails to communicate design. **(1-2)** |  |
|  |  |  |  | **Total** |  |

**Notes:**

* **Excellent:** Exceeds expectations in all aspects.
* **Good:** Meets expectations with minor improvements needed.
* **Satisfactory:** Meets basic requirements but lacks depth.
* **Needs Improvement:** Does not meet basic requirements.

| **Lecturer’s Overall Comments** |
| --- |
|  |

**Warning:**

It is important to include citation and references in the assignment. You are required to upload your document to Turnitin for plagiarism checks which advisable not more than 25% out of 100%. Please read the following statement properly to avoid penalty:

**“Plagiarism & Copying”**

PLAGIARISM AND COPYING ARE STRICTLY PROHIBITED AND CONSIDERED A SERIOUS OFFENCE. THE FACULTY HAS THE FULL RIGHTS TO TAKE DISCIPLINARY ACTIONS AGAINST ANY STUDENT WHO PLAGIARISE OR COPY THE WORK OF OTHERS. AN ACCEPTABLE TURN IT IN / SAFE ASIGN SIMILARITY REPORT MUST BE INCLUDED AS APPENDIX FOR ALL SUBMITTED ASSIGNMENTS. THE TOTAL PERCENTAGE IN THE REPORT SHOULD LESS THAN 25% and EACH INDIVIDUAL ITEM IN THE REPORT SHOULD LESS THAN 3%. Your assignment will NOT BE GRADED if it does not meet the above threshold.

The Faculty reserves the right to use any means to identify plagiarism and copying. If you reference the work of other authors, you must ensure that you properly reference them to avoid the charge of plagiarism. Copying the work of others (current students, past students or the works of others) automatically earns you a Fail grade. If two or more students copied from each other, both students (regardless of who copies from whom) will earn a Fail grade.”

# 

# **Table of contents**

[**Table of contents**](#_3jbzkadpj82n) **4**

[**1. Introduction**](#_gyjt3haearwx) **8**

[1.1 Project Overview](#_156htvvzfprc) 8

[1.2 Purpose of the Report](#_u7tf0u39do75) 8

[**2. Background**](#_8816gvvqzxuu) **9**

[2.1 Problem Statement](#_nd4vry7vezbt) 9

[2.2 Proposed Solution](#_xpzme3qshg6) 9

[2.3 Target Audience](#_2lcmvbb5ju6v) 10

[**3. Requirements Analysis**](#_je6pfzv6a50x) **11**

[3.1 Functional Requirements](#_vxbsnv7u0pr) 11

[3.2 Non-Functional Requirements](#_abbhfyei9dvl) 15

[**4.Application Title Objective, and Storyboard**](#_eep8g9yms3lc) **22**

[4.1 Application Title](#_5c1duk9ptzyp) 22

[4.2 Application Objective](#_tu2a0r7syoov) 22

[4.3 Storyboard](#_njz7f1a3ty4c) 23

[1. Splash Screen](#_l60wndfdelsl) 23

[2. Home Screen (Book List View)](#_bdikfy64047e) 24

[4. Edit Book Screen](#_w5vh17urtngf) 24

[5. Delete Confirmation Dialog](#_spymhiov9hd1) 25

[6. Book Detail Screen](#_b83zqvb3gtup) 25

[7. Share Screen](#_gjz51lzc9k8i) 25

[8. Notification Screen](#_7rap0dp40e6k) 25

[**5. Features and Functions**](#_zf6olf7034js) **26**

[5.1 Core Inventory Management Features](#_wp57y98pvnp1) 26

[5.2 Search Functionality](#_vru0sw8hruqd) 28

[5.3 Underlying Technical Functions](#_iu5buxelxlgc) 29

[**6. Key Classes and Components**](#_1od6zph482hr) **30**

[6.1 Key classes](#_ko5lj8thx1m4) 30

[6.2 Feature Implementation Details](#_eqo22slo3tog) 36

[**7. Screenshots of App**](#_5ych01emku1h) **41**

[**8. Contribution, Lessons Learned, and Conclusion**](#_2zn0qyt9uew1) **55**

[8.1 Contribution](#_sm3xgrfhgju) 55

[8.2 Lessons Learned](#_fwqstw28yjqw) 56

[8.3 Conclusion](#_m0afz1i4zbqf) 57

# **1. Introduction**

## **1.1 Project Overview**

“Shelf Stock” is a nicely created software application that can be used in containing all the problems associated with book inventory management. This application provides the core functionalities of creating, reading, editing and deleting book entries, and all of them should be done in a way that a user can use this application easily and intuitively. In addition to the ‘main’ features, ‘Shelf Stock’ has more, to help with the management of your books from a user experience value perspective and features beyond ‘essential’ standard book management features. They also include the abilities to share book details with others and to get notice of changes in the book inventory. The application is a modern Android app developed with modern Android development principles, one of which is to use Jetpack Compose to create a responsive user interface and the Room Persistence Library to store local data verily and reliably. The application has a maintainable, testable and scalable codebase based on the Clean Architecture principles. Asynchronous software techniques and coroutines are used by the application to provide the best smooth and responsive behavior regardless of how much data and how complex an operation it is performing. The user interface has a fresh and intuitive design for newcomers as well as a very efficient one for the experienced users; easy navigation and standard design components will do the trick seamlessly.

## **1.2 Purpose of the Report**

It is a final product of a paper which explains in detail what has occurred in 'Shelf Stock' application. It seeks to provide full knowledge of what work was made for and why the application was created and the functional and non functional applications that helped the application to be ready to use. Next, in the rest of the report, we will detail how the application features were implemented and attempted to represent the code implementation which was used together with documentation of the architectural decisions and design patterns that were considered. It will also specifically give relevant screenshots and diagrams which will demonstrate, by the way it’ll look, and how it should be used, a visual walkthrough of the application’s user interface. Additionally, it would reflect on how the learning from a development process as a preaching lesson, how they bumped into the pitfalls, how they responded to them, and what the impingement of these into their final result of project. Finally, the report will end with a short summary of the project’s outcomes, the ability of the project bringing in a negative impact on inventory management on the books, and potential future developments and expansion in the social book market.

# **2. Background**

## **2.1 Problem Statement**

Today it becomes a problem for people or small companies to keep their record of physical books of collections because the world is moving at a fast pace. Manual Log and Spreadsheets is cumbersome and time consuming and highly prone to errors. None of these methods provide the flexibility such as quickly search desired books, update info or sharing details with one or more people. Also, it is impossible to receive notifications of additions or removals in the collection with a little struggle. This lack of efficient and user friendly tools will on many occasions lead to organization, loss of books and making it impossible to manage the books inventories effectively. In particular, the following problems are identified.

* Inefficient Tracking: The difficulty in keeping an up-to-date record of books.
* Time-Consuming Management: This was the effort involved adding, editing or deleting book information.
* Limited Searchability: The problem of quickly finding a particular book in the collection.
* Lack of Sharing Capabilities: Lack of easy leeching of the book details to another data manager.
* Absence of Notifications: Users will never be notified when there is a change in the book inventory.
* Data Inconsistency: It increases the risk of errors and inconsistencies in the data that is manually entered.
* Scalability Issues: Difficulties with managing growing and large collections.

## **2.2 Proposed Solution**

The 'Shelf Stock' application, so designed as to be able to solve these problems of managing book inventories by giving out a complete and very easy mobile application. The application features the following that streamline the process of organizing and accessing book information.

* **CRUD Operations**:The system allows instant data entry, reading, modification and book entry deletion to maintain current inventory records.
* **User-Friendly Interface**: Users can easily navigate through their collections because the application features an uncomplicated interface that is both friendly and attractive to view.
* **Local Data Storage**: The Room Persistence Library enables users to have quick access to book data stored on their devices through a local database for offline functionality.
* **Sharing Functionality**:Users have the capacity to share book information across multiple platforms which helps promote joint work and data sharing.
* **Notification System**: The application sends real-time notifications to users for everything ranging from new book records to altered book inventory or deleted books.
* **Clean Architecture**: The app implements Clean Architecture principles which leads to high levels of maintenance, testability and scalability.
* **Responsive Design**: Jetpack Compose enables this app to deliver an adaptive UI which responds perfectly on all screen dimensions and mobile platforms.
* **Asynchronous Operations**: The application uses coroutines to process database operations which provides a non-blocking operation that maintains smooth user interface responsiveness.
* **Filter by Category**: Users enjoy filtering book options by category which helps them seek books from specific genres or classifications.
* **Category Selection with Custom Input**: Users can choose an already defined category from a dropdown menu during the process of adding or updating books. A new input field becomes accessible when users select "Other" which allows them to create their own category in the inventory management system.

## **2.3 Target Audience**

`The 'Shelf Stock' application has been developed with the many people and small organisations in mind which have a necessity to coordinate and manage their book collections in an obvious and easy form.

* Individual Book Collectors: Enthusiast and hobbyist to catalog and organize their personal libraries. It includes people with a wide variety of collections from fiction and non-fiction, to specialized genres and collections of rare books and academic texts.
* Small Libraries and Reading Groups: It is used by organizations with limited resources that require a simple and cost effective solution to inventory their books. Community libraries, church libraries, book clubs, etc. would all fall into this category.
* Educators: Classroom teachers and professors who are in charge of classroom libraries or resource collections. The app can assist them to track down books that they lent to scholars, arrange perusing materials and impart assets with companions.
* Researchers and Academics: For those individuals who have to manage a collection of research papers, journals, and books. With the app they can quickly check for material that is relevant and cite it.
* Small Businesses: Small business with a small library, book, or manuals or training materials. Law firms, medical offices or design studios could all be listed here.
* Bookstores (Small): Fundamental inventory system from basic inventory management system that empowers independent bookstores.

# **3. Requirements Analysis**

The "Shelf Stock" application's functional along with non-functional needs receive detailed description in this section to establish a comprehensive framework of expected system capabilities.

## **3.1 Functional Requirements**

The "Shelf Stock" application needs to perform certain specified actions according to its functional requirements. The system documentation focuses on presenting the intended behavior combined with the user services which the application must provide.

### **FR1: Manage Inventory Items**

**FR1.1: Add Item**

* The system shall allow users to add new items to the inventory.
* The following item details shall be captured:  
  + Name *(required)*
  + Category *(required)*
  + Quantity *(required)*
  + Price *(required)*
  + Author *(optional)*
  + Language *(optional)*
  + Description *(optional, see FR6)*
* The system shall validate the input data for required fields (Name, Category, Quantity, and Price).
* Upon successful addition, a confirmation message shall be displayed ( FR7).

**FR1.2: Edit Item**

* The system shall allow users to modify existing inventory items.
* All item attributes shall be editable.
* The system shall validate updated data before saving ( FR6).
* On successful modification, a confirmation message shall be displayed (FR7).

**FR1.3: Delete Item**

* The system shall allow users to delete an item from the inventory.
* The selected item shall be removed from the database.
* Upon successful deletion, a confirmation message shall be displayed ( FR7).

**FR1.4: View Item Details**

* The system shall allow users to view detailed information for a selected item.
* All item attributes (Name, Category, Quantity, Price, Author, Language, Description) shall be displayed in a user-friendly format.

**FR1.5: List All Items**

* The system shall display a complete list of all inventory items.
* The list shall include key attributes: Name, Category, Quantity, Price, etc.
* Items shall be displayed in an organized and readable layout.

### 

### 

### 

### **FR2: Search Inventory Items**

**FR2.1: Search by Title or Category**

* The system shall allow users to search inventory items using their title or category.
* Items that match the search term shall be retrieved from the database and displayed.

### **FR3: Data Persistence**

**FR3.1: Local Data Storage**

* The system shall store all inventory data locally using the Room Persistence Library.

**FR3.2: Offline Access**

* The application shall function offline, allowing users to access and manage inventory without internet connectivity.

**FR3.3: Database Operations**

* The system shall use DAO interfaces for performing all database operations.

### **FR4: Additional Features**

**FR4.1: Sharing Functionality**

* The system shall allow users to share book details through available communication platforms.

**FR4.2: Notification System**

* The system shall notify users upon the addition, update, or deletion of inventory items.

**FR4.3: Filter by Category**

* The system shall allow users to filter items by predefined or custom categories.

**FR4.4: Custom Category Input**

* When “Other” is selected in the category dropdown, the system shall prompt users to enter a custom category.

### **FR5: User Interface**

**FR5.1: Responsive Design**

* The UI shall be responsive and compatible with various screen sizes using Jetpack Compose.

**FR5.2: Clean Architecture**

* The system shall be developed using Clean Architecture to ensure maintainability and scalability.

### **FR6: Data Validation**

* The system shall validate user input to ensure:  
  + Required fields are not left empty (Name, Category, Quantity, Price).
  + Data types are correct (Quantity and Price must be numeric).
  + Quantity is non-negative, and Price must be greater than zero.

### **FR7: Operation Feedback**

* The system shall provide confirmation messages for successful actions such as:  
  + “Item added successfully”
  + “Item updated successfully”
  + “Item deleted successfully”
* In case of failure, appropriate error messages shall be displayed.

## **3.2 Non-Functional Requirements**

The ‘Shelf Stock’ application’s quality attributes are defined by non-functional requirements (NFRs). They describe by how operation, not what operation, the system should behave. Also, they discuss things of performance, usability, reliability and security.

### **NFR1: Performance**

NFR1.1: Responsiveness

* The system shall respond to user interactions within 1 second during normal operation.

NFR1.2: Startup Time

* The application shall load within 2 seconds on startup.

NFR1.3: Optimized Data Retrieval

* The system shall ensure quick data access using optimized queries, regardless of inventory size.

### **NFR2: Usability**

NFR2.1: Intuitive Interface

* The UI shall be easy to navigate and understand for users with minimal training.

NFR2.2: Consistent Design

* All screens and components shall follow a consistent design language.

NFR2.3: Clear Feedback

* The system shall provide clear feedback messages for both successful and erroneous actions (see FR8).

NFR2.4: Accessibility

* The system shall support accessibility features such as screen readers and alternative text for images.

### **NFR3: Reliability**

NFR3.1: Data Integrity

* The system shall ensure that inventory data remains consistent and uncorrupted across all views and operations.

NFR3.2: Error Handling

* The system shall gracefully handle unexpected errors and display informative error messages without crashing.

NFR3.3: Operation Confirmation

* The system shall notify users of the result of each operation (success/failure).

### **NFR4: Maintainability**

NFR4.1: Clean Architecture

* The application shall follow Clean Architecture principles for modular and testable design.

NFR4.2: Code Documentation

* The codebase shall include clear comments and documentation for future maintenance.

NFR4.3: Use of Libraries

* The application shall rely on proven, well-established libraries to reduce bugs and improve performance.

### **NFR5: Security**

Locally stored data shall be protected from unauthorized access using device-level security.

### **NFR6: Data Validation**

NFR6.1: Field Validation

* The system shall ensure required fields are filled before allowing submission.

NFR6.2: Type & Range Validation

* Numeric fields like Quantity and Price must be validated for correct type and logical value (e.g., Quantity ≥ 0, Price > 0).

### **NFR7: User Notifications**

NFR7.1: Feedback on Actions

* Users shall receive notifications for both successful and failed actions.

NFR7.1.1: Success Messages

* Example: “Item added successfully”, “Item updated successfully”.

NFR7.1.2: Error Messages

* Errors shall include context-specific messages for better understanding and resolution.

# 

# **4.Application Title Objective, and Storyboard**

# **4.1 Application Title**

# *Shelf Stock: Inventory Management for Bookstores*

## **4.2 Application Objective**

The ‘Shelf Stock’ application aims to provide bookstore staff with a powerful and easy to use inventory management application. Through this application, the staff will also be able to:

* You effortlessly add new book items to the inventory, to the relevant details.
* Categorized books can be searched quickly by their titles.
* Easily update book details.
* Remove books from the inventory.
* See information for each book.
* Record of the books in stock is accurate and up to date.
* Can filter books by category.

By streamlining these essential inventory management tasks, "Shelf Stock" aims to:

* Lower the time and labor needed to work inventory.
* Minimize errors in inventory records.
* To improve the efficiency of bookstore operations on the whole.
* To enhance book staff's ability to track and manage their book collection.

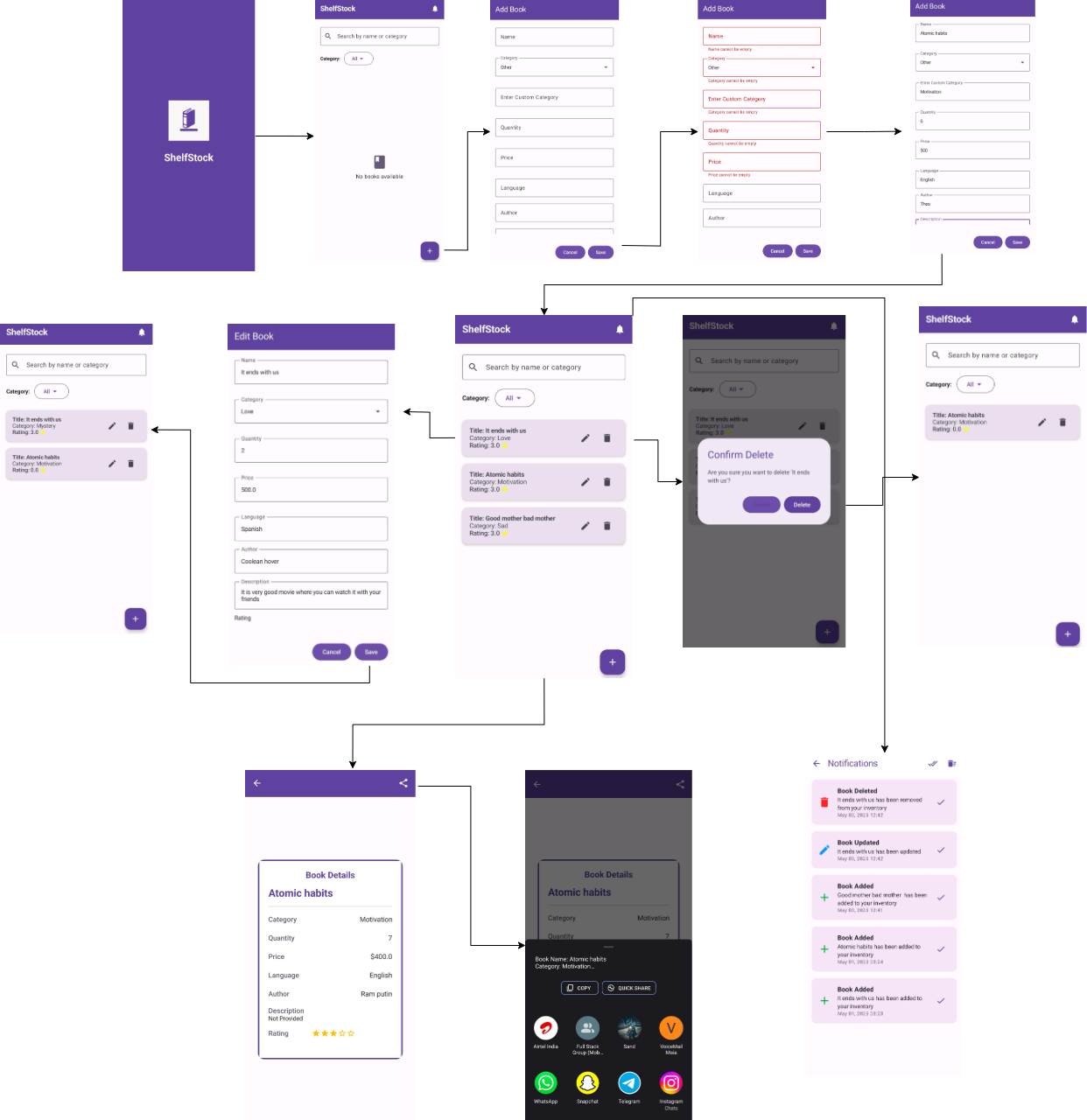
# 

# 

# 

# 

## **4.3 Storyboard**



This storyboard demonstrates how ShelfStock mobile application functions by designing an interactive flow for managing book inventory effortlessly. This document illustrates the main screens accompanied by user interactions that result in system reactions through a detailed and structured layout.

### 

### **1. Splash Screen**

* The launch interface of ShelfStock presents the application logo and named screen through a branded splash screen.

### **2. Home Screen (Book List View)**

* Displays a searchable and filterable list of added books.
* Each item card shows: Title, Category, and Rating.
* Users can:  
  + Search by name or category.
  + Use the FAB (floating action button) to add a new book.
  + Tap edit or delete icons on any listed book.

**3. Add Book Screen**

* Accessible via FAB on the home screen.
* Contains input fields: Name, Category (or custom), Description, Quantity, Price, Author, Language.
* Includes form validation:  
  + Required fields must be filled.
  + Quantity and price must be numeric.
  + Validations trigger red borders and error hints if inputs are incorrect or missing.
* On successful submission, a success message is shown, and the book appears in the list.

### **4. Edit Book Screen**

* Triggered by the edit icon on any book card.
* Pre-fills the form with existing book details.
* Allows updates to all fields.
* Includes Save and Cancel buttons for user control.

### **5. Delete Confirmation Dialog**

* Triggered by the delete icon on a book card.
* The interface presents a confirmation dialog for users to either submit or stop the deletion process.
* After confirmation the book removal proceeds and displays a success notification.

### **6. Book Detail Screen**

* Opens when a book item is tapped.
* Shows complete details: Category, Quantity, Price, Language, Author, Description, Rating.
* Includes a Share icon at the top-right to enable detail sharing.

### **7. Share Screen**

* Presents the system's share interface.
* Allows sharing book details via apps like WhatsApp, Email, etc.

### **8. Notification Screen**

* Lists all operation messages such as:  
  + Book Added Successfully
  + Book Updated Successfully
  + Book Deleted Successfully
* Notifications include timestamps and color-coded status indicators for clarity.

# 

# **5. Features and Functions**

Details of the features and functions of the ShelfStock application are detailed in this section, which offers complete understanding of the application’s capabilities.

## **5.1 Core Inventory Management Features**

And these are fundamental pieces of how a bookstore would manage its inventory day to day.

### **5.1.1 Viewing the Inventory (Main Screen)**

First, ShelfStock will welcome the bookstore owners by showing a Welcome Screen which contains the app's logo and app name. After the Welcome Screen, the second screen that the user will see will be the Main Screen, which provides a dynamic list that summarizes the user’s selections in the form of a live overview. This list contains each line that follows each Title or Category, with the Quantity of books we have in stock so we don’t sell more than we have, the Selling Price, and since this makes this task easier, with Category to group them. While the list is there, it is done for easy navigation, has sorting (by name, category, size, etc), pagination (for large inventories), and possibly visual cues (colourised, icons to make things stand out like low stocks or checking what is on backorder).

**User Scenario & Benefit:** Suppose a customer calls in and randomly chooses which title is out available on a very busy Saturday morning. From ShelfStock’s main screen, the owner can quickly search and easily confirm to himself that the book is in stock. No customer service or sales will take battery as they will speed up saving time for searching through shelves or manual logs. Moreover, it grants the owner an opportunity to take a quick glance at the quantities and realize their stock level is low before it is too late and reorders can be placed.

### **5.1.2 Adding New Books (Add New Book Screen)**

Intuition of the user is kept in mind in order to accomplish the purpose of Add New Book Screen by implementing clearly labeled input fields to input all required information about the new book. All of the fields described below must be entered and are mandatory fields, hence, this can assure that the record is never wrong as Name, Category, Quantity and Price will always be present. Thus the database grows optional fields (Language, Author, Description) providing more substance to book records for use internally, or, alternatively, for future features such as product description. It immediately revalues the real time input validation back to the user to tell exactly where to fix any errors before it saves, which manages data integrity from the beginning.

**User Scenario & Benefit:** The Add New Book Screen allows him or her to rapidly patch the titles of new titles for a new shipment of books into inventory. It’s intuitive form which has shrunk the learning curve and it is validated to prevent the crucial details such as being missed or entered in the wrong way. By speeding up the process the owner now has the time to put the books out on the shelves and available for purchase in a shorter period of time.

### **5.1.3 Editing Existing Book Information (Edit Book Screen)**

It is fluid inventory and the Edit Book Screen knows this and has an easy change of existing book records. The form is available from the Item Detail Screen and prepopulated with current data, and when changed is virtually assured to have lowered the reentry risk of the same, unchanged data. For this feature the capability is to adjust the quantity from sales and returns, error correction on the categorization, market change response price update, and supplement with missing info such as author and a short summary.

**User Scenario & Benefit:** To promote a book, a price of it may have to be adjusted. The owners can open the details of the book easily, find the book easily, and change the price immediately in the Edit Book Screen. Similarly, if an inventory has had a miscategorization its remedy also can be done very effectively immediately in order to accurately report and arrange the inventory. First benefit of this real time editing is that there is continuous reflected inventory of inventory of the company at the inventory database.

**5.1.4 Removing Books from Inventory (Item Detail Screen)**

Once from the Item Detail Screen you can use the Delete Book function to do this to remove books away from the active inventory. The one thing that’s saved is the confirmation prompt that’s such an essential part of this safety net that it really functions as a safety net to catch accidental data loss. There is no shortage of books that can not come back into circulation at all (sold out and not going to be restocked), books that are simply mangled enough or are not anymore for sale (returned books that will not be restocked and pump into inventory again) or now outdated and not taking up shelf space (titles are disposed of and ended).

**User Scenario & Benefit:** If a book is permanently removed from the store’s books to show crop, the owner has to use the Delete function to remove it from the currently shown inventory list for active inventory. By this, stock will never be off and will not confuse when counting stock or generating sales reports. That comes in the shape of the confirmation prompt, because peace of mind in this case is being deleted intentionally.

### **5.1.5 Viewing Detailed Book Information (Item Detail Screen)**

On the Book Info Screen, we provide space to view ‘a full view’ of all the attributes recorded on the selected book. We’ve separated the main information to this screen in an organized way and only the optional information that we’ve entered. The second use case is when there is no existing database, therefore we can use it when we’re trying to answer customer questions about the author of a book or its description, or the big inventory from a report to see what’s in the store.

**User Scenario & Benefit**: This is a customer asking the book’s author. It means the bookstore owner can jump to the book in the Main Screen and right to Item Detail Screen for fast access to this information that would usually require quite a few steps. Additionally, since everything is right in one place, the process of creating a sales report or analyzing the inventory trends is done smoothly.

## **5.2 Search Functionality**

The search bar on the Main Screen gives the power to users to search books with regards to keyword relevance easily and quickly.

**User Scenario & Benefit:** Saying what they never did before with symbols for a song that she remembers singing with, a customer asks about a book of songs. The bookstore owner can type these keywords on the search bar and the inventory list will be filtered instantly to identify the book with these keywords.

## **5.3 Underlying Technical Functions**

These essential functions operate behind the scenes to ensure the application's reliability and usability.

### **5.3.1 Reliable Local Data Management**

Inventory management is done directly on the user’s device inside a local database, as the application uses. This is to ensure data integrity and performance. The biggest plus here is that local storage means bookstore owners can still work with inventory without the internet, like in places that don’t have much internet, or even those who prefer not to have it.

**User Benefit:** It makes sure that the inventory data is accessible, secured and perform efficiently at all times. In particular, the offline capability is very useful for on the floor inventory checks and other management tasks that cannot be done normally from stable internet connection.

### **5.3.2 Data Integrity Assurance**

We apply Comprehensive Data Validation in all possible ways to the application, i.e. at Add New Book screen, Edit Book screen. Also, these validation rules will make you send only no empty fields (on mandatory fields and not empty entries) as well as the correct formats, quantity and price (data types) among others. Error messages are clear and informative to clarify to users what is wrong, thus saving the data and avoiding invalid input.

**User Benefit:** Furthermore, it provides a major level of risk elimination of data entry errors that will eliminate data entry errors leading to the dirtiness of the inventory database. So error messages can provide you with immediate feedback to help your user to speed up himself on his mistake.

### **5.3.3 Share Book Details**

The application allows easy sharing of information about the books among users. For example, this feature can be used to tell customers new arrivals, recommend books to colleagues, to promote anything and so on, when we are trying to balance the lightness and speed. Users can choose to share key details of a book (title, author, description) through almost any channel they use to email, messaging apps and social media.

### **5.3.4 Clear and Timely User Feedback Mechanisms**

Throughout the application different types of User Feedbacks are developed so that users are informed as to the ending of their actions. For example, when a book is added to the inventory list, and when books, authors or editors in the list are updated visually (the existing thing is edited, or a new book is added or an existing book is deleted). The input of the data provides good and specific error messages that help them to fold in the way of corrections. Additionally, such indications of success through some visual cues or brief confirmation message can indicate successful operations.

# **6. Key Classes and Components**

## **6.1.1 MainActivity.kt**

**Path:** com/example/shelfstock/[MainActivity.kt](http://mainactivity.kt)

This MainActivity initializes the local database, repositories, and view models for books and notifications. It uses Jetpack Compose to set the UI and passes the necessary view models to the MyApp composable for managing app navigation and state**.**

### **6.1.2 Book.kt (Data Class)**

**Path:** com/example/shelfstock/data/book.kt

@Entity(tableName = "books")

data class Book(

@PrimaryKey(autoGenerate = true) val id: Int = 0,

val name: String,

val category: String,

val quantity: Int,

val price: Double,

val language: String,

val author: String,

val description: String,

val rating: Float = 0.0f

)

The books table Room entity is given here. It has a schema that is used for book information, namely, title, author, description, genre, and stock.

#### **6.1.3 BookDao.kt & BookDatabase.kt**

**Path:**

* BookDao.kt: com/example/shelfstock/data/BookDao.kt
* BookDatabase.kt: com/example/shelfstock/data/BookDatabase.kt

@Dao

interface BookDao {

@Query("SELECT \* FROM books")

fun getAllBooks(): Flow<List<Book>>

@Query("SELECT \* FROM books WHERE id = :id")

suspend fun getBookById(id: Int): Book?

@Insert(onConflict = OnConflictStrategy.REPLACE)

suspend fun insertBook(book: Book)

@Update

suspend fun updateBook(book: Book)

@Delete

suspend fun deleteBook(book: Book)

@Query("SELECT \* FROM books WHERE name LIKE '%' || :query || '%'")

fun searchBooksByName(query: String): Flow<List<Book>>

}

@Database(entities = [Book::class], version = 1)

abstract class BookDatabase : RoomDatabase() {

abstract fun bookDao(): BookDao

}

* BookDao provides SQL operations like fetching all books, inserting, deleting, and searching by title.
* BookDatabase initializes the Room database with Book as the entity.

#### **6.1.4 BookViewModel.kt**

**Path:** com/example/shelfstock/viewmodel/BookViewModel.kt

**class BookViewModel(private val repository: BookRepository) : ViewModel() {**

**val allBooks: LiveData<List<Book>> = repository.allBooks**

**fun insert(book: Book) = viewModelScope.launch {**

**repository.insert(book)**

**}**

**fun delete(book: Book) = viewModelScope.launch {**

**repository.delete(book)**

**}**

**fun searchBooks(query: String): LiveData<List<Book>> {**

**return repository.searchBooks(query)**

**}**

**}**

This ViewModel connects the UI with the repository. It handles insert, delete, and search operations and exposes LiveData to the UI for reactivity.

#### **6.1.5 AppNavigation.kt**

Path: com/example/shelfstock/NavigationRoutes.kt

**@Composable**

**fun AppNavigation() {**

**val navController = rememberNavController()**

**NavHost(navController = navController, startDestination = "main\_screen") {**

**composable("main\_screen") { MainScreen(navController) }**

**composable("add\_edit\_book") { AddEditBookScreen(navController) }**

**composable("book\_detail/{bookId}") { backStackEntry ->**

**val bookId = backStackEntry.arguments?.getString("bookId")?.toInt() ?: 0**

**ItemDetailScreen(navController, bookId)**

**}**

**}**

**}**

This code Handles app navigation using Jetpack Navigation Compose. Defines the routes for main screen, add/edit, and detail view screens.

#### **6.1.6 MainScreen.kt**

**Path:** com/example/shelfstock/screens/mainscreen.kt

**@Composable**

**fun MainScreen(navController: NavController, viewModel: BookViewModel = viewModel()) {**

**val books by viewModel.allBooks.observeAsState(emptyList())**

**LazyColumn {**

**items(books) { book ->**

**BookListItem(book = book, onClick = {**

**navController.navigate("book\_detail/${book.id}")**

**})**

**}**

**}**

**FloatingActionButton(onClick = {**

**navController.navigate("add\_edit\_book")**

**}) {**

**Icon(Icons.Default.Add, contentDescription = "Add Book")**

**}**

**}**

This code displays the list of books using LazyColumn. Provides navigation to AddEditBookScreen and individual ItemDetailScreen on book tap.

#### **6.1.7 AddEditBookScreen.kt**

**Path: com/example/shelfstock/screens/AddEditBookScreen.kt**

**@Composable**

**fun AddEditBookScreen(navController: NavController, viewModel: BookViewModel = viewModel()) {**

**var title by remember { mutableStateOf("") }**

**var author by remember { mutableStateOf("") }**

**Button(onClick = {**

**val newBook = Book(title = title, author = author, ...)**

**viewModel.insert(newBook)**

**navController.popBackStack()**

**}) {**

**Text("Save Book")**

**}**

**}**

This code allows the user to add a new book. Uses mutable state for input fields and calls insert() via the ViewModel to save data.

#### **6.1.8 ItemDetailScreen.kt**

**Path: com/example/shelfstock/screens/ItemDetailScreen.kt**

**@Composable**

**fun ItemDetailScreen(navController: NavController, bookId: Int) {**

**// Logic to fetch and show book details**

**Text("Book Details for ID: $bookId")**

**}**

This code Shows the detailed view of a book. The bookId is passed via navigation and used to fetch and display full book data.

### **6.2 Feature Implementation Details**

#### **6.2.1 Navigation Implementation**

**File: NavigationRoutes.kt**

**NavHost(navController = navController, startDestination = "main\_screen") {**

**composable("main\_screen") { MainScreen(navController) }**

**composable("add\_edit\_book") { AddEditBookScreen(navController) }**

**composable("book\_detail/{bookId}") { backStackEntry ->**

**val bookId = backStackEntry.arguments?.getString("bookId")?.toInt() ?: 0**

**ItemDetailScreen(navController, bookId)**

**}**

**}**

This code Implements navigation using NavHost and composable() functions from Jetpack Navigation Compose. Routes include:

* main\_screen (book list)
* add\_edit\_book (form)
* book\_detail/{bookId} (detail screen)

#### **6.2.2 Data Persistence (Room)**

**Files: BookDao.kt, BookDatabase.kt, book.kt**

**@Entity(tableName = "books")**

**data class Book(...)**

**@Dao**

**interface BookDao {**

**@Query("SELECT \* FROM books") fun getAllBooks(): LiveData<List<Book>>**

**@Insert suspend fun insert(book: Book)**

**@Delete suspend fun delete(book: Book)**

**@Query("SELECT \* FROM books WHERE title LIKE '%' || :query || '%'")**

**fun searchBooks(query: String): LiveData<List<Book>>**

**}**

**@Database(entities = [Book::class], version = 1)**

**abstract class BookDatabase : RoomDatabase() {**

**abstract fun bookDao(): BookDao**

**}**

This code Data is persisted using Room. BookDao handles database queries, and BookDatabase initializes the Room database.

#### **6.2.3 UI Structure (Jetpack Compose)**

**Files: mainscreen.kt, AddEditBookScreen.kt, ItemDetailScreen.kt**

**LazyColumn {**

**items(books) { book ->**

**BookListItem(book = book, onClick = { ... })**

**}**

**}**

**FloatingActionButton(onClick = { navController.navigate("add\_edit\_book") }) {**

**Icon(Icons.Default.Add, contentDescription = "Add Book")**

**}**

Jetpack Compose powers the entire UI:

* LazyColumn for book list
* FloatingActionButton to navigate to add screen
* Composable functions define screen layouts declaratively

#### **6.2.4 Search Logic**

**File: BookViewModel.kt**

**fun searchBooks(query: String): LiveData<List<Book>> {**

**return repository.searchBooks(query)**

**}**

This code Implements search by title using a ViewModel method which calls BookDao.searchBooks(query) through the repository. Search results are LiveData, making the UI reactive.

#### **6.2.5 Share Functionality**

**File: ItemDetailScreen.kt**

**val *context* = LocalContext.current**

**val *shareIntent* = remember {**

**Intent(Intent.ACTION\_SEND).apply {**

**type = "text/plain"**

**putExtra(Intent.EXTRA\_TEXT, "Check out this book: ${book.title} by ${book.author}")**

**}**

**}**

**Button(onClick = {**

**context.startActivity(Intent.createChooser(shareIntent, null))**

**}) {**

**Text("Share")**

**}**

This code Implements Android’s native sharing feature. Creates a SEND intent to share book details via any compatible app (like Messages, Gmail, etc.).

#### **6.2.6 Notification Implementation**

**File: NotificationViewModel.kt**

**class NotificationViewModel(...) : ViewModel() {**

**fun sendNotification(context: Context, message: String) {**

**val builder = NotificationCompat.Builder(context, CHANNEL\_ID)**

**.setContentTitle("ShelfStock Reminder")**

**.setContentText(message)**

**.setSmallIcon(R.drawable.ic\_notification)**

**NotificationManagerCompat.from(context).notify(1, builder.build())**

**}**

**}**

This code Uses Android NotificationCompat to send local notifications. Ideal for reminders, e.g., when a book is low in stock.

#### **6.2.7 Input Validation**

**File: AddEditBookScreen.kt**

**if (title.isBlank() || author.isBlank()) {**

**Toast.makeText(context, "Please fill all fields", Toast.LENGTH\_SHORT).show()**

**} else {**

**viewModel.insert(Book(title = title, author = author, ...))**

**navController.popBackStack()**

**}**

This code Validates that title and author are not blank. Shows a toast if validation fails, else proceeds to insert book data.

# 

# 

# 

# **7. Screenshots of App**

**App Logo**

****

**Splash Screen**

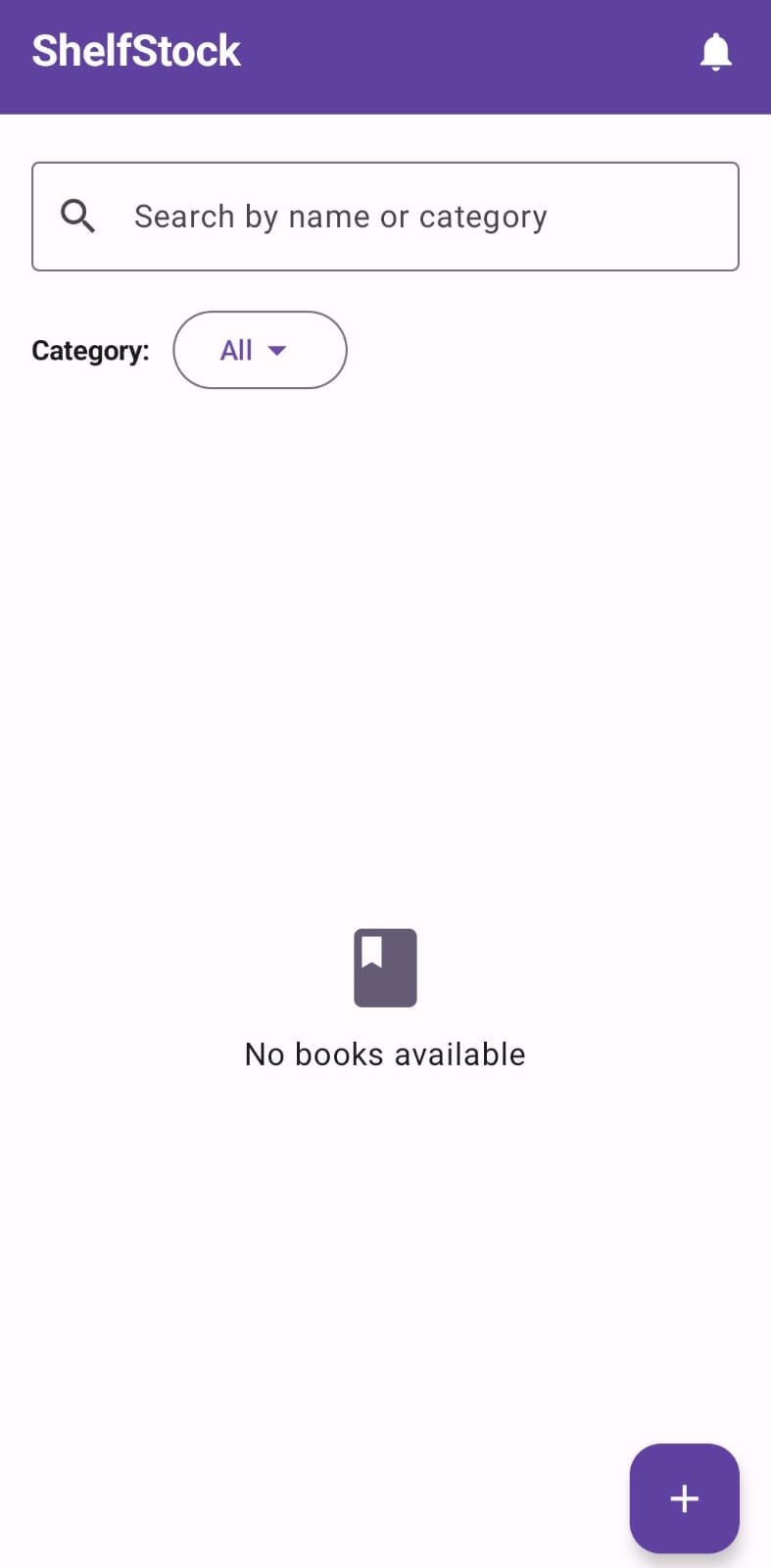
BookInventory shows it's app logo together with its name for a brief period at the application launch. The app logo along with its name appear as the program initializes in the background. BookInventory starts with the loading screen displaying app logo and name before it transitions to the main inventory list screen within a short time.



### 

### **Main Screen (No Books)**

ShelfStock shows a descriptive initial display to users when it finds unstocked inventory through its launch sequence. A notice signaling the absence of available books will replace the standard inventory list. From this screen users can easily perform the process of entering their initial inventory items through an prominently displayed button.



### 

### 

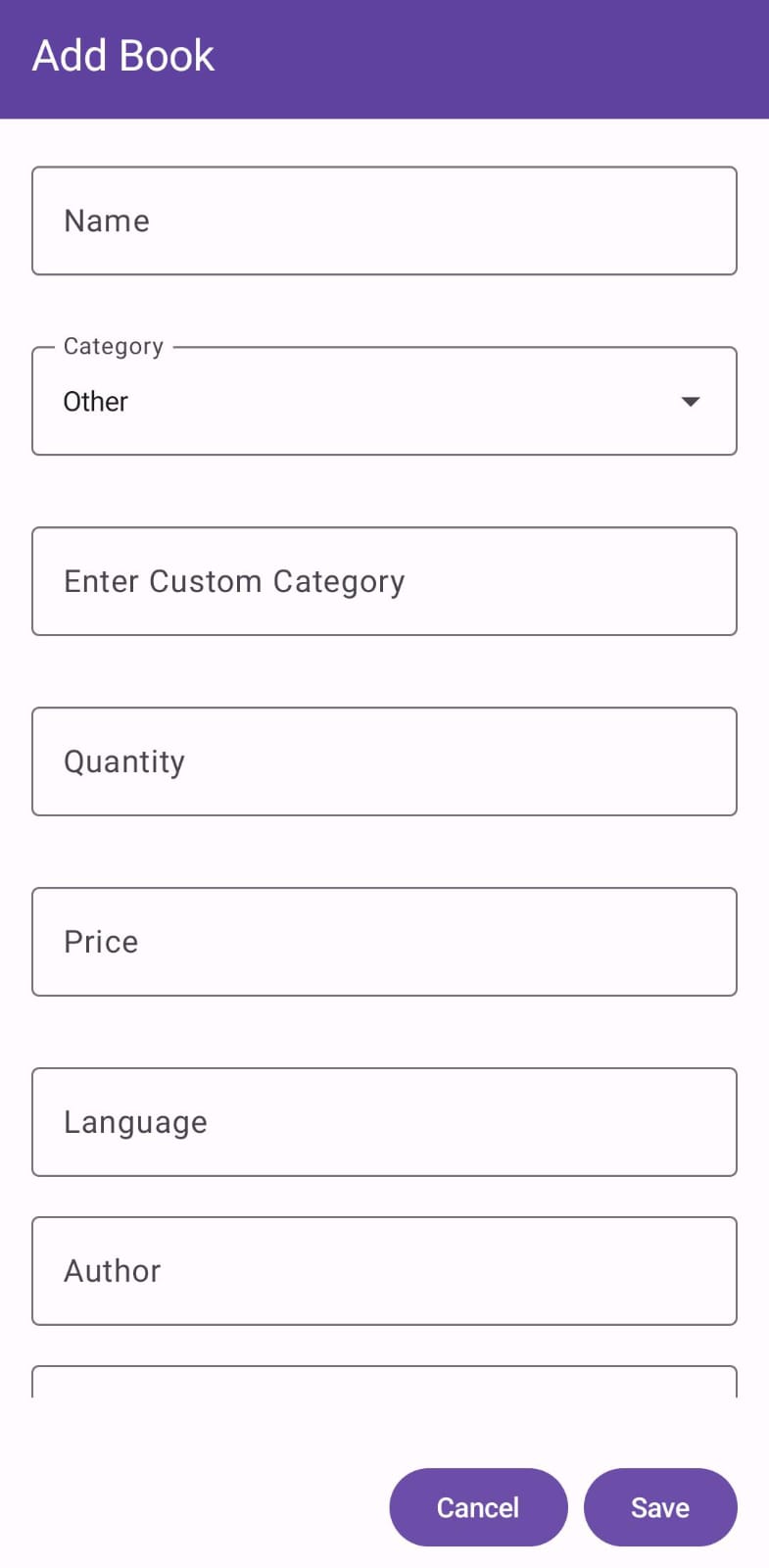
### 

### 

### 

### **Add Book Screen**

Users can enter book data using "Add Book" screen fields which request name, category, quantity, price, author, rating and language information. The application checks the data inputs for accuracy before accepting them. The application saves this record by utilizing it to generate a new book entry in the Room database located on the device. After performing data entry the system provides explicit approval messages when data has been registered correctly but it also clearly shows any registration errors to the user.



### 

### 

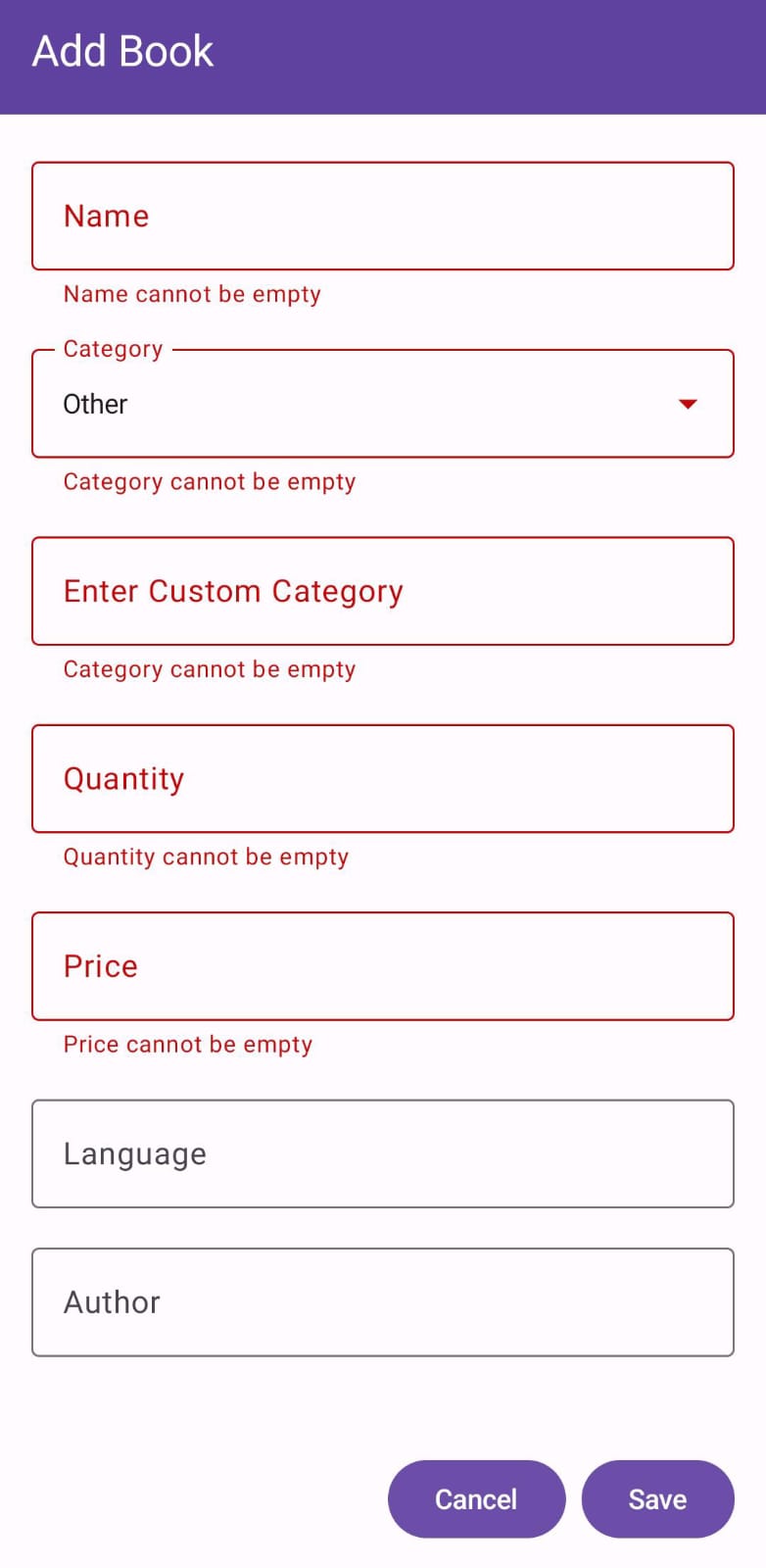
### 

### 

### 

### **Data Validation**

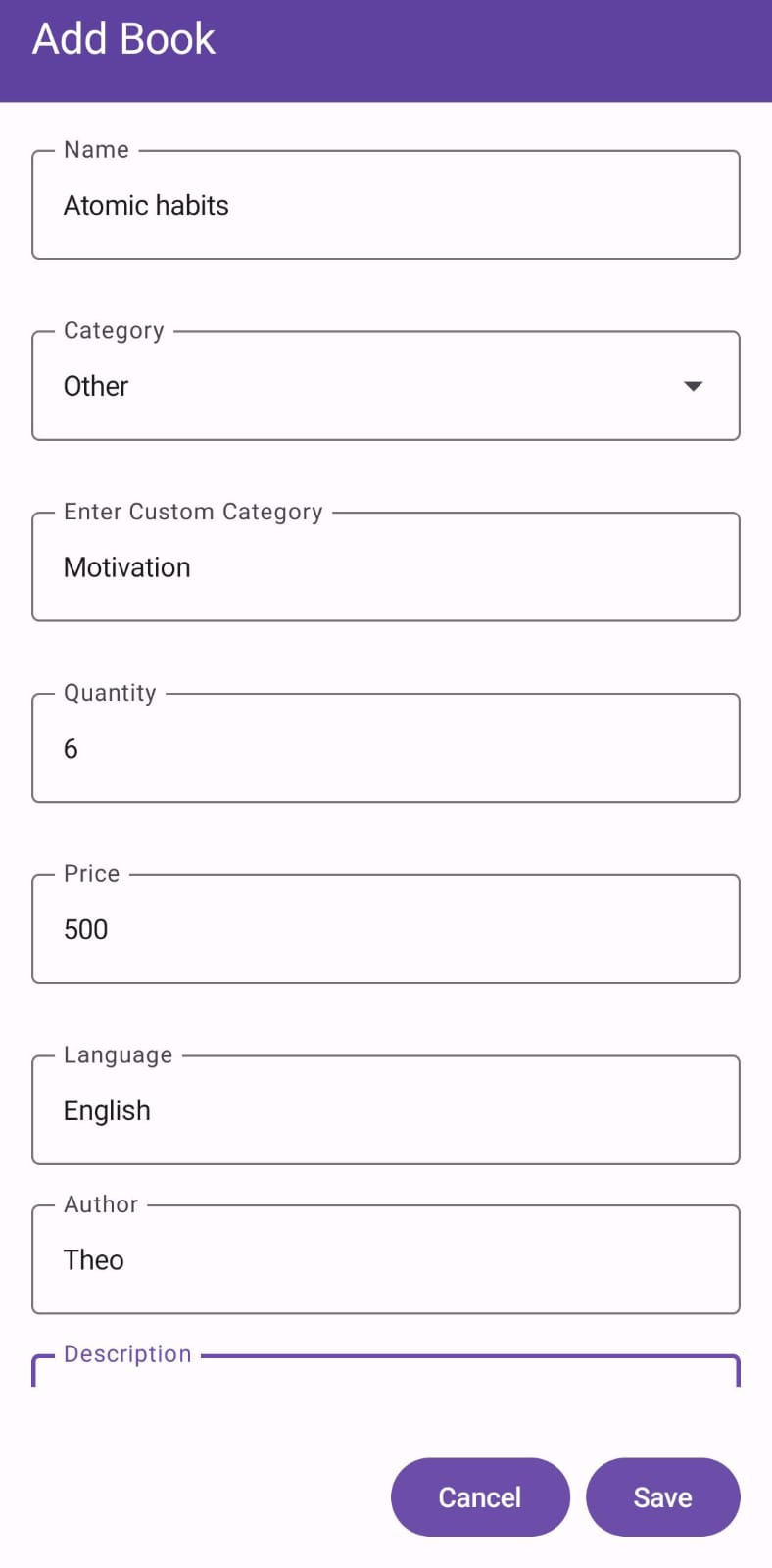
ShelfStock will create thorough data validation systems for the "Add New Item" interface. ShelfStock will verify the presence of user input data in name, category, quantity and price entry fields. Data validation will apply to the quantity and price fields to verify their contents are numerical values. The program enables users to move forward with inventory database addition only after valid entries fill all necessary fields.



### 

### **Adding Book**

ShelfStock shows the newly added book after a successful addition to both Main Inventory List and display confirms the action was finished correctly.



### 

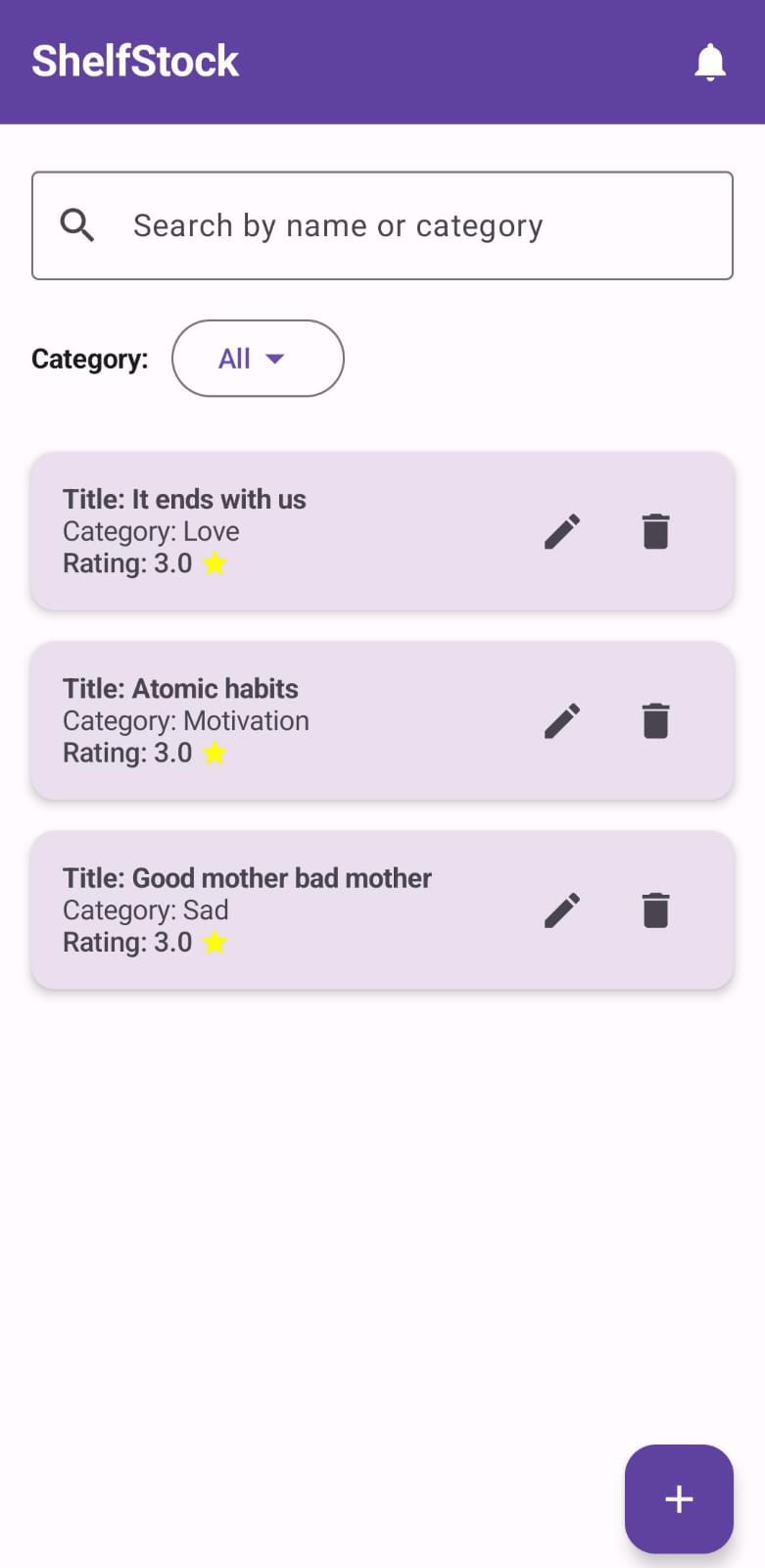
### 

### 

### 

### **Main Screen (With Books)**

The system shows all books that exist within the database current collection.



### 

### 

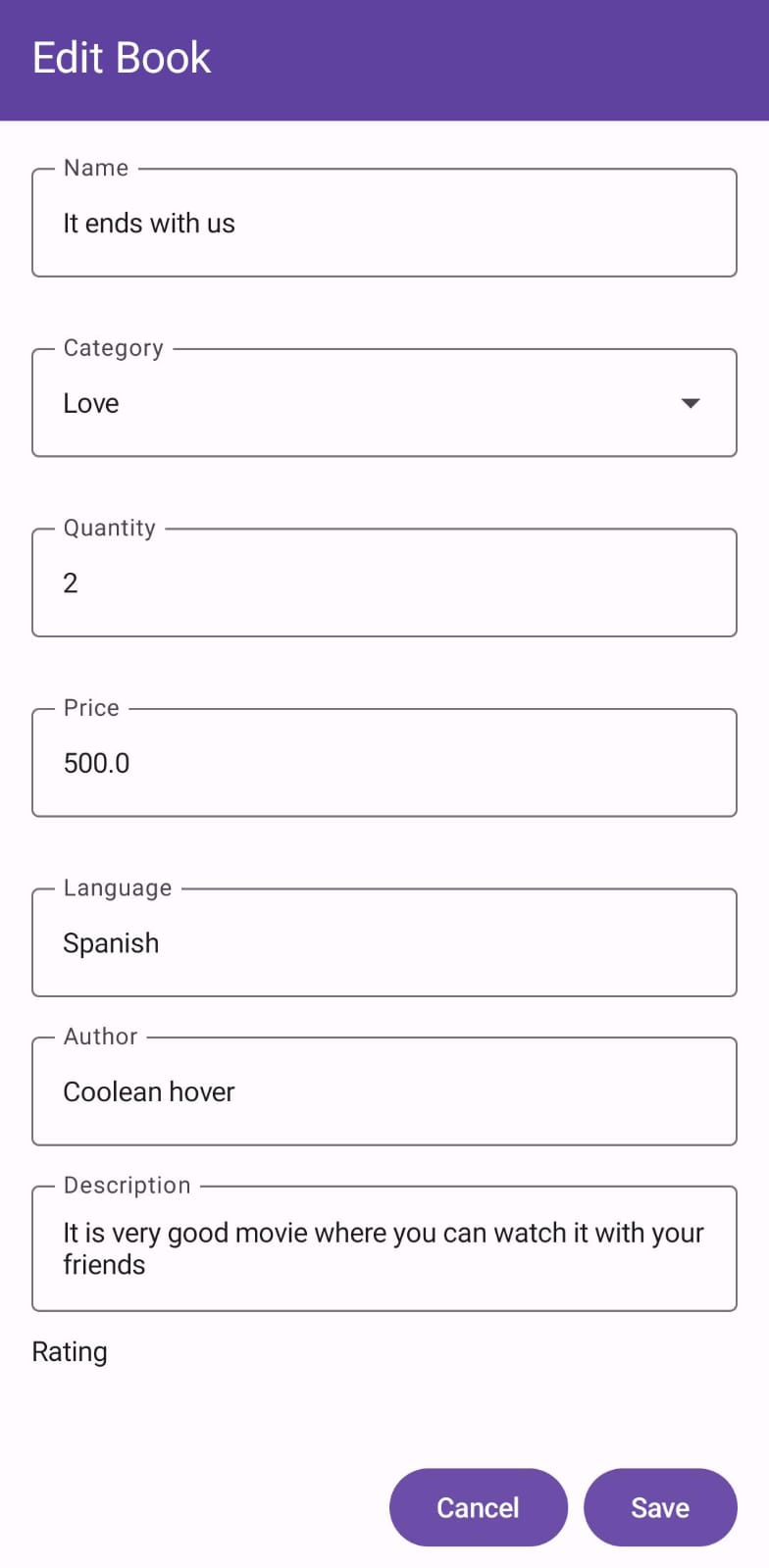
### 

### 

### 

### **Edit Book**

Through ShelfStock users can modify all information about books that remain in their inventory system. The update feature enables users to select a book from their inventory selection followed by the "Edit Item" screen. The "Edit Item" screen displays all existing details of the chosen book which automatically appear in the input fields. ShelfStock provides users with a screen where they can edit the existing book information which they can modify before saving the changes. After making a change the system at ShelfStock will update Room database entries while simultaneously updating the main inventory display.



### 

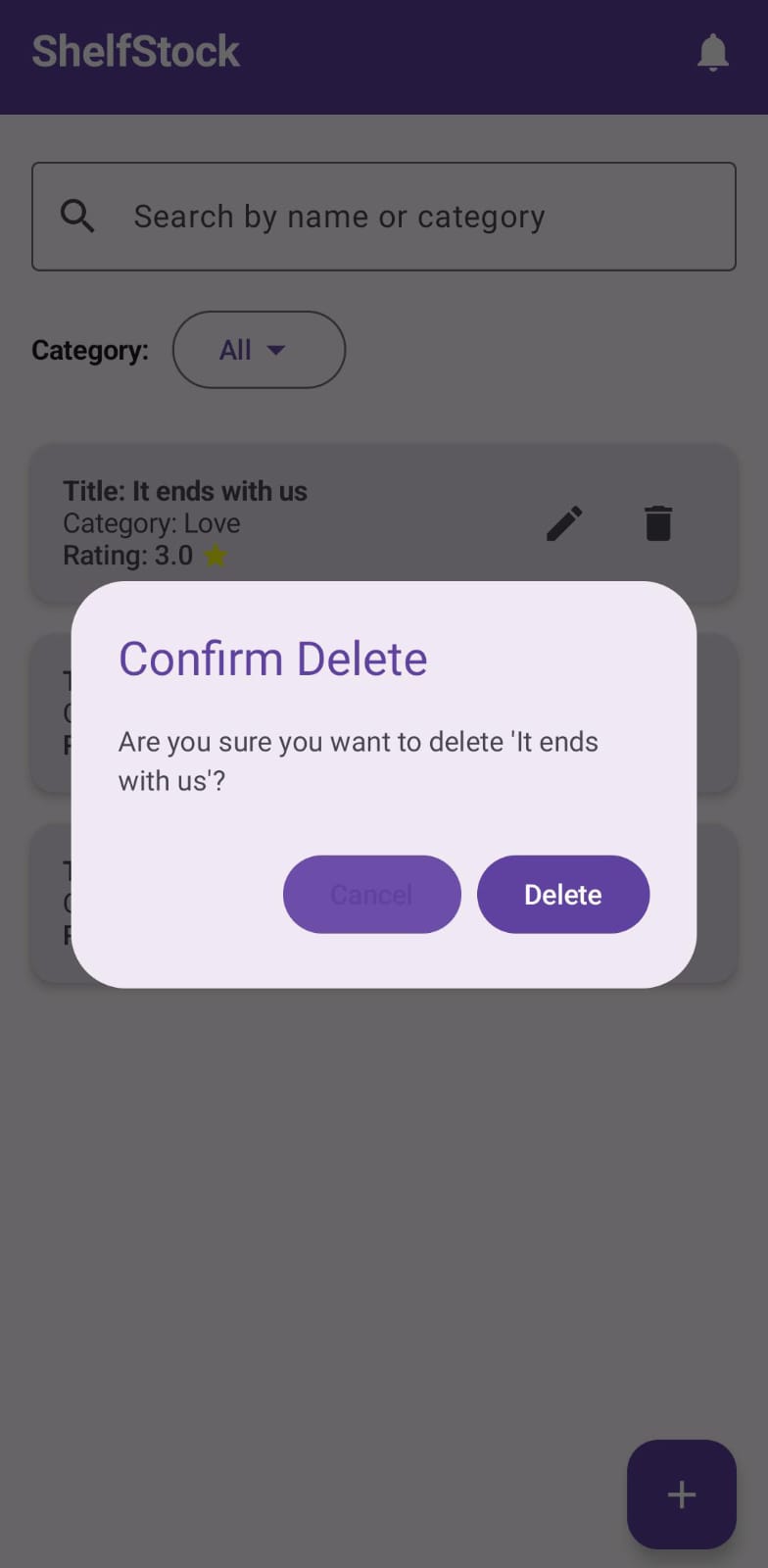
### 

### 

### 

### **Deleting Book**

Users will benefit from ShelfStock by finding a tool to permanently remove books from inventory. Before performing a book deletion operation ShelfStock will first show a confirmation window to avoid accidental actions. After user confirmation the app uses Inventory Item Dao to perform the action of deleting the corresponding entry for the selected book from the local Room database. The main inventory list gets automatically updated with the removed book information. Successful book removal becomes clear through brief "Book deleted" information displayed to the user.



### 

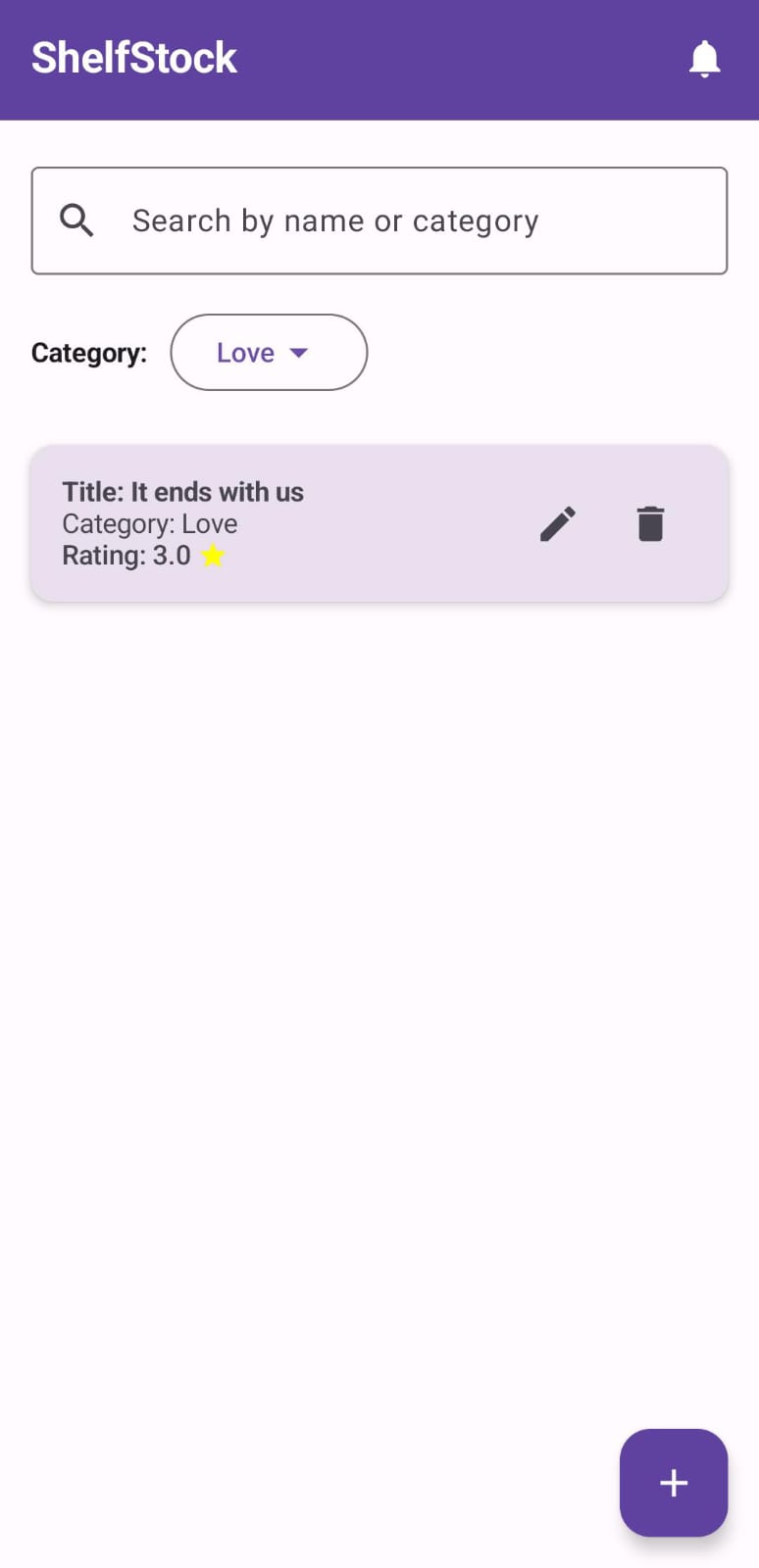
### 

### 

### 

### **Search by Category**

ShelfStock will build a search tool that lets users locate their store's inventory books in an effective manner. A search bar with prominence will be located on the main inventory screen or users can access it easily. Users need to type keywords which correspond to book titles or categories. Users can type into ShelfStock while it automatically shows matching results from the book list based on their input. Search operations at ShelfStock work without regard to letter case for improved result flexibility. Clearing the search bar or deleting entered search terms will restore the display of the entire inventory list to the user. Users can find particular books easily by using this search feature instead of needing to browse an extensive inventory list.



### 

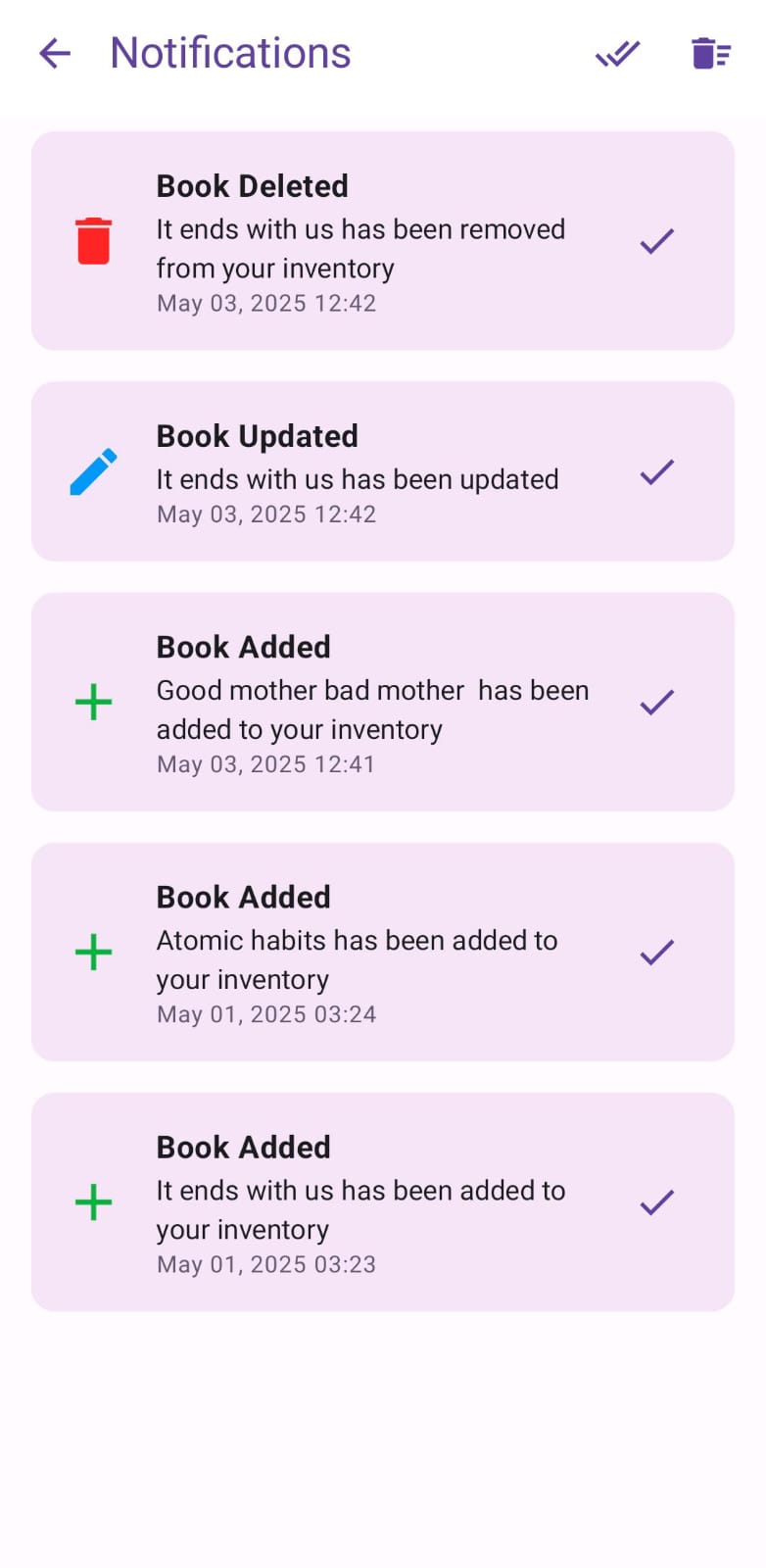
### 

### 

### 

### **Notifications**

The application will present timely alerts that provide users with essential information. The application generates notification alerts to show users what successful steps they have taken including book additions and updates together with deletions.



### 

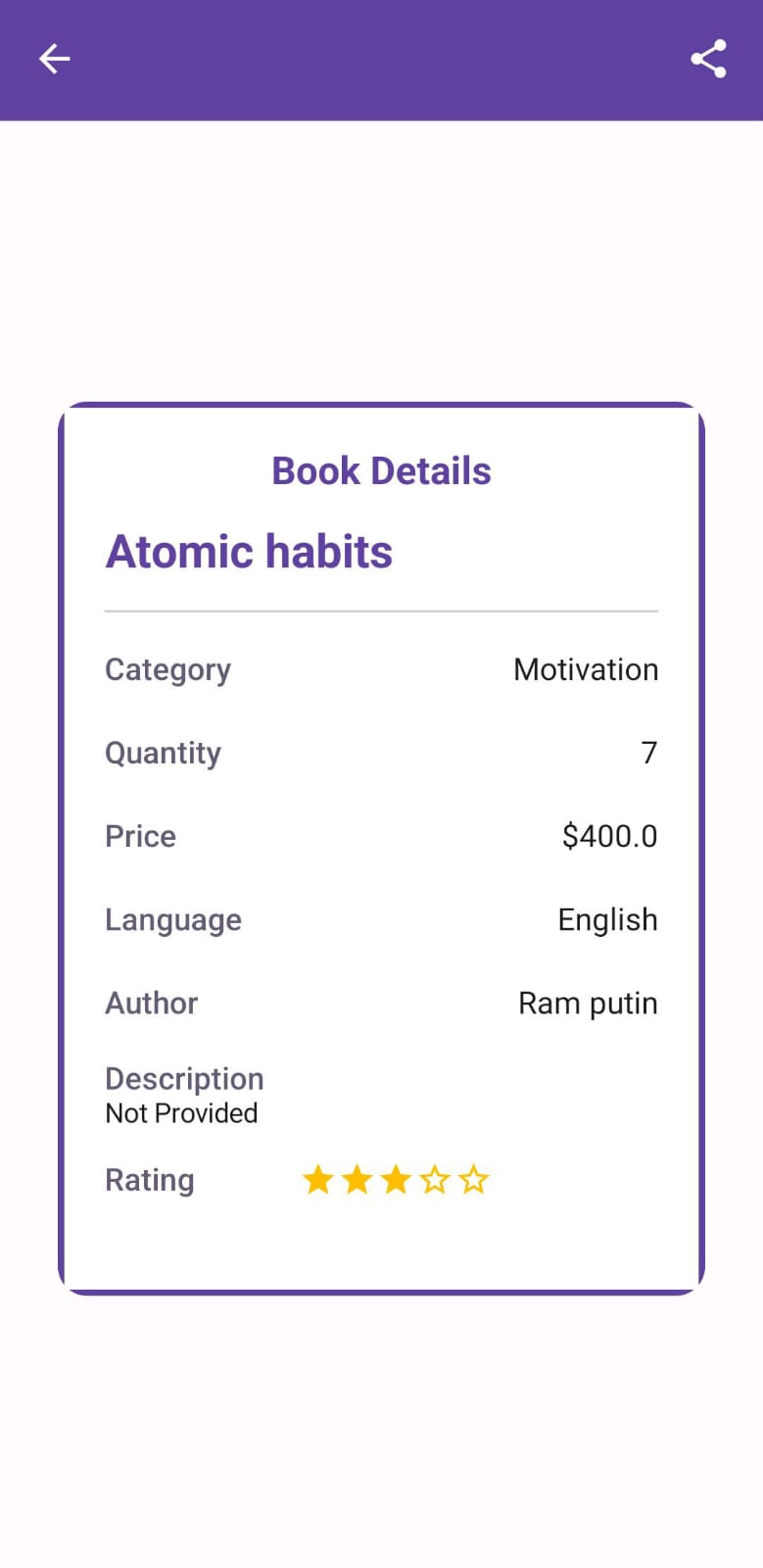
### 

### 

### 

### **Display Book**

This screen presents all information concerning the selected book including its full details about title, author, price and additional metadata. Users can easily understand detailed book information because the screen provides extensive viewing options.



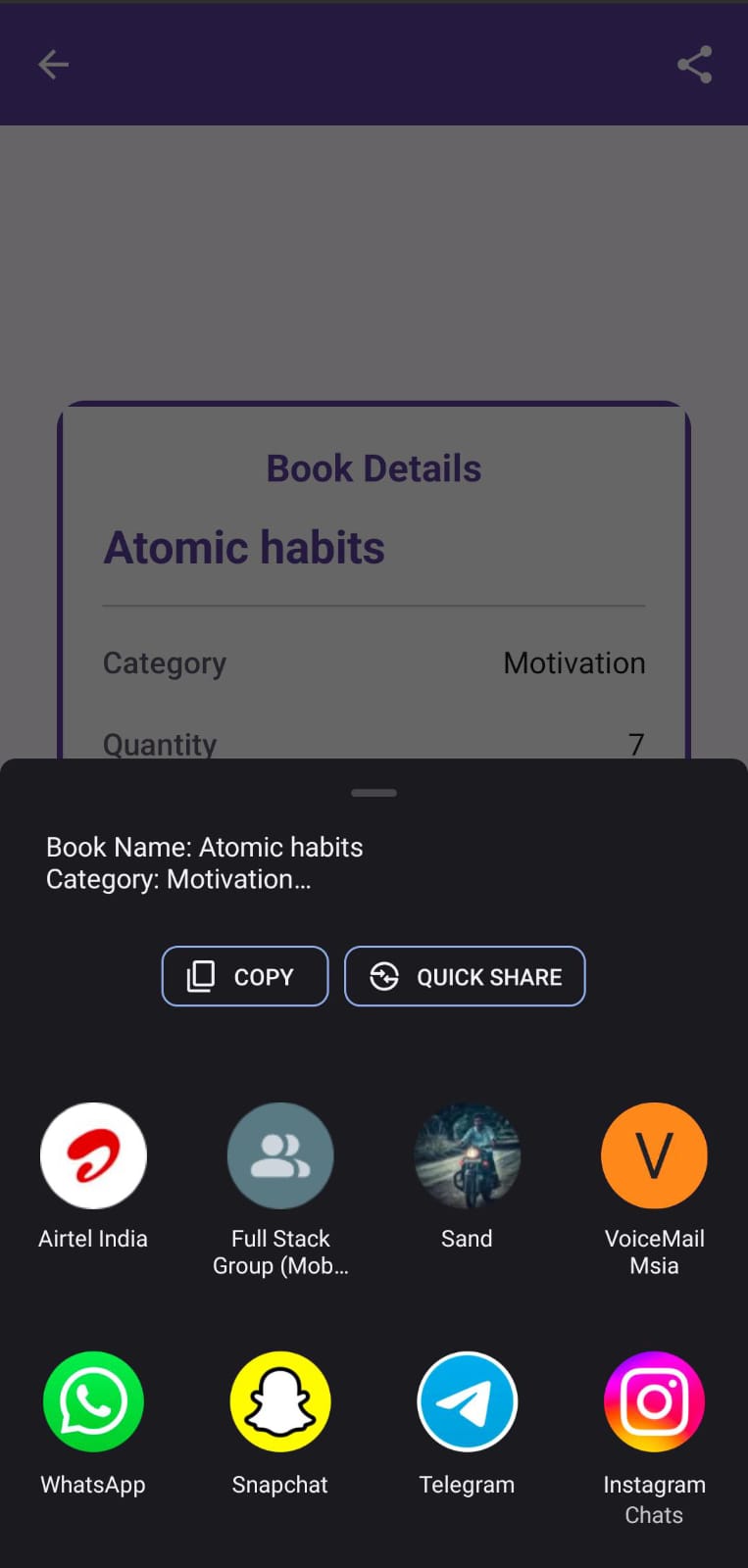
### 

### 

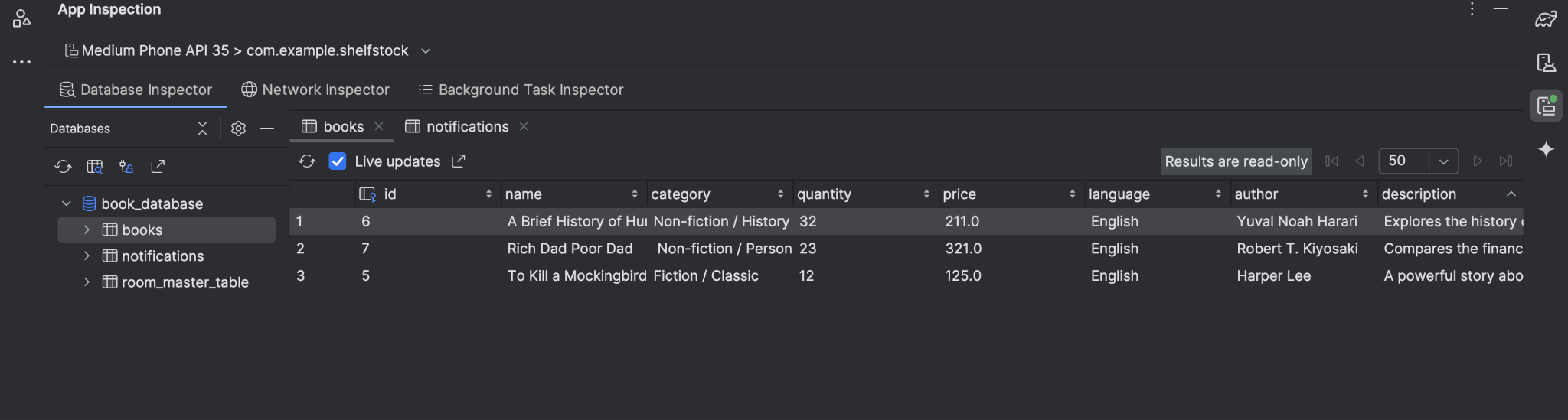
### 

### **Share Book**

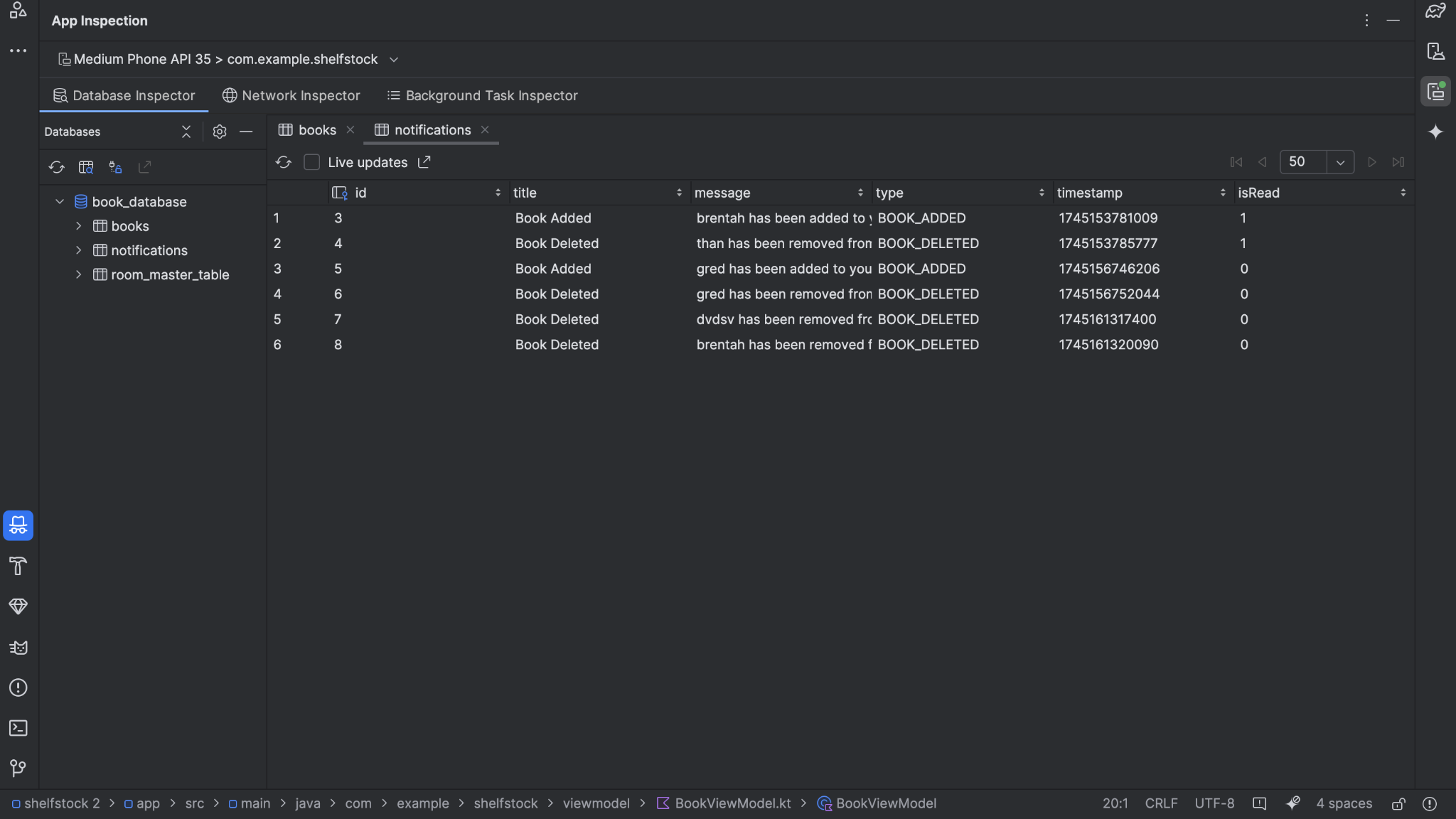
The app lets users generate book details for sharing through alternative applications and platforms.



**Database for book details**

Books inspection for different books  ****

**7.12 Database for notifications**

****

## 

## 

## 

## **8. Contribution, Lessons Learned, and Conclusion**

### **8.1 Contribution**

Implementation of a bunch of technical components and integration of several Android architecture principles occurred during the development of the ShelfStock application. They also contributed one of the most notable contributions in the sense that they used the MVVM (Model View Model) architecture that separated concerns better and allowed the code to be even more maintainable and scalable. The architecture had a clear defined role per layer: VM Directory managing business logic, Repository dealing with data access, and View (UI) solely responsible for

Secondly, the successful implementation of Room Database to store and persist data was another key contribution. Through Room, the app stores locally all the book entries that the user adds which can be later edited or deleted. This feature helps to make sure that users data stays even after closing the app or restarting the device, which has its influence on the reliability and trust users have on the app.

User interface (UI) was developed with Jetpack Compose, Android’s new striking tool to create native UIs. With this approach, the interfaces became clean, declarative, and highly responsive on various screens of the application. The main book listing screen, the add/edit form, the detail view are all composed of modular UI elements following best practice of accessibility and design.

Furthermore, the app also includes a couple practical functionalities which makes the app more user friendly. The offered features include a real time search functionality, filtering book entries with respect to the keywords, a sharing function that users can use in order to share book info through other apps, and a notification system that can remind users about that status updates or some other. I integrated it with input validation in the app as well, so that users input all needed info before getting it saved and avoiding saving incomplete or incorrect stuff. As a result, each of these features was designed with user experience in mind and also met best practices on Android development.

### 

### 

### **8.2 Lessons Learned**

I was happy to be able to learn and put my modern Android development skills to use, and in a practical way, by being able to build the ShelfStock application. What I learnt was to use Jetpack Compose as an effective way to build reactive as well as dynamic UI’s. Compose uses a declarative syntax to construct the UI and with state and is more preferable for deployments than with a traditional XML based UI. When I explored how to handle state with Compose tools like remember, mutableStateOf, state hoisting and overall, how you can make the whole component reactive and reusable.

However, working with Room Database gave me understanding on how to persist data in Android. The straight clean data modeling and lifecycle aware operations lesson was learned wrt to creating entities, implying DAOs and SQL queries and how to configure the Room database instance. One of the key factors while building user friendly dynamic applications was to have real time update UI whenever underlying data changes, and LiveData, along with Room helped give us it.

Also you can see very well why separation of concerns is an important factor in developing a scalable application with MVVM architecture. Learning how to structure code in layers helps make code readable and maintainable and is helpful since the structure is implemented in two layers (ViewModel which is responsible for UI related logic, and Repository which manages data sources along with the involved DAOs to do data transaction).

The use of Jetpack Navigation is another very valuable learning outcome. With structurally and intuitively designed navigation flow between the screens in order to clarify how to transmit arguments to the composable functions and how a navigation controller works with back stack behavior. These skills will help users make multi-screen Android apps as natural and easy as possible.

It was also in line with the design for the user. Aside from that, I did form validation, I have also decided to provide feedback with Toast messages and I decided that consistency on the UI elements must be created. Not only was there hands-on learning with testing and debugging, learning how to solve runtime errors and how to optimize performance of the app to make sure that the app is behaving as expected under different possible conditions.

### 

### 

### 

### 

### 

### **8.3 Conclusion**

Finally, the creation of the ShelfStock application represents a major step in learning and practicing the use of Android development. The application fulfills it’s purpose of simple managing and editing of a personal book inventory. And it integrates Data persistence, nice UI and user friendly, real time search, and shareable content in one well structured application.

This project was very beneficial from a learning point of view, as it served as a good practical application reinforcing theoretical knowledge. It showed how the modern tools like Jetpack Compose, Room, and Navigation Component come together to make robust Android applications. Additionally, it also enhanced one's knowledge in terms of software design patterns, modular architecture, and practices of clean code.

Secondly, it also gives a good base of experience to build other projects. You can also use Firebase, user authentication, barcode scanning for quick book entry, and analytics to see how the user interacts with your app. This can be expanded by more features. Through enhancements, we could upgrade the application to a level that can be used as a production level software that can be used by other users as a whole.

As a whole, ShelfStock is not a completed project, but a symbol of how much I learned in the Android development area. What comes together is a thoughtful design, functional practicality that scales, and a technical achievement at the same time that is a moving towards more complicated and impactful mobile applications in the future.