

**APTECH COMPUTER EDUCATION**  
**(Gulshan- e – Maymar Branch)**

**PROJECT REPORT**

**Submitted in Partial Fulfillment of the Requirements for the Diploma in Software Engineering**

**Semester: Fourth**

**Project Title:** BOOK STORE APP

**Submitted By:**  
**Students Names:**

* Muhammad Hashir
* Muhammad Bin Aamir
* Osama Ali Khan
* Bilal Bin Aamir
* Shahmeer Mughal

eProject Report: Book Store for Our Readers

# Acknowledgement

I would like to express my sincere gratitude to my esteemed professor

Instructor's Name] for providing invaluable guidance, constructive feedback, and consistent support throughout the development of this eProject.

I am profoundly grateful to Institute Name] for providing the necessary resources and infrastructure that significantly enhanced my understanding of mobile application development methodologies and practices.

Furthermore, I extend my appreciation to my colleagues and family members for their unwavering moral support and encouragement throughout the various phases of this project's implementation.

# eProject Synopsis

|  |  |
| --- | --- |
| **Project Title** | Book Store for Our Readers |
| **Technology Stack** | Flutter (Dart), Firebase (Firestore & Authentication) |
| **Platform** | Android (Mobile Application) |
| **Database** | Cloud Firestore |
| **Authentication** | Firebase Email/Password Authentication |

This project presents a comprehensive online bookstore mobile application

developed using Flutter and Firebase technologies. The application facilitates user registration and authentication, enabling customers to browse a diverse collection of books, implement search and filtering functionality based on price ranges or

literary genres, manage selected items in shopping carts or wishlists, and complete purchases through a streamlined checkout process.

Administrative users possess privileges to manage inventory through book listing administration, including addition, modification, and removal capabilities. The application delivers a contemporary user experience with efficient data

management and secure authentication protocols. The primary objective of this

project is to demonstrate the development of a scalable, cloud-integrated mobile application solution.

# eProject Analysis

## Problem Statement:

The contemporary market exhibits a significant deficiency in mobile-optimized platforms where literature enthusiasts can conveniently explore, select, and purchase books without the necessity of visiting physical retail establishments, thus creating a substantial demand for an innovative digital solution.

## Objectives:

To develop a user-centric mobile application with the following functionalities:  Implementation of secure user registration and authentication systems

 Categorization of literary works by genre for enhanced browsability  Integration of wishlist and shopping cart functionality

 Development of an intuitive and efficient checkout interface

 Provision of administrative capabilities for inventory management

## Proposed System Architecture:

The system architecture is bifurcated into two primary functional modules:

 User Module: Facilitates end-user interaction with the application for literature exploration and procurement.

 Administrative Module: Provides authorized personnel with inventory management capabilities.

## System Benefits:

 Comprehensive access to an extensive literary collection

 Enhanced remote shopping experience with intuitive navigation  Sophisticated administrative interface for inventory control

 Implementation of Firebase technology ensuring data security and real-time synchronization

# eProject Design

* **Data Flow Diagrams**

**DFD (Context Diagram):**

┌─────────┐ ┌───────────────┐ ┌─────────┐

│ User │────▶│ Book Store │◀────│ Admin │

│ │ │ Application │ │ │

└─────────┘

│

│

▼

└───────────────┘

│

│

▼

└─────────┘

┌─────────────────┐ ┌─────────────────┐

│ User Operations │ │ Admin Operations│

│ (Browse/Purchase)│ │ (Inventory Mgmt)│

└─────────────────┘ └─────────────────┘

**Level 1 DFD:**

graph TD

Auth["Firebase Authentication"] --- Login User --- Login["Login/Register"] --- Admin Login --- Home["User Home"]

User --- Browse["Browse Books"] Home --- BookDetails["Book Details"] Browse --- BookDetails

Admin --- Manage["Manage Inventory"] Manage --- BookDetails

Browse --- CartWish["Cart/Wishlist"] BookDetails --- CartWish

CartWish --- Checkout

Checkout --- Order["Order Placed"] --- DB"Firestore Database"]

* **User Interaction Flowchart**

┌─────────────┐

│ Start │

└──────┬──────┘

│

▼

┌─────────────┐

│Splash Screen│

└──────┬──────┘

│

▼

┌─────────────┐

│Login/Register│

└───┬───────┬─┘

│ │

▼ ▼

┌─────┐ ┌───────┐

│Login│ │Register│

└──┬──┘ └───┬───┘

│ │

└────┐ ┌──┘

│ │

▼ ▼

┌─────────┐

│Home Page│◀───────┐

└────┬────┘ │

│ │

▼ │

┌─────────┐ ┌───────────┐

│View Book│ │ Book List │

└────┬────┘ └─────┬─────┘

│

▼

│

│

┌────────────────┐

│

│Add to Cart/

│Wishlist

│

│

│

│

└───┬────────────┘ │

│ │

▼ │

┌────────┐ ┌───────────┐

│Checkout│ │Continue │

│ │ │Browsing │

└───┬────┘ └─────┬─────┘

│ │

│ │

▼ │

┌─────────────┐ │

│Order Placed │ │

└──────┬──────┘ │

│ │

▼ │

┌─────────────┐ │

│ End │◀───┘

└─────────────┘

## Database Design (Firestore Collections)

**Collection: users**

* billingAddress: "testing"
* email: "hashir@test.com'
* name: "Muhammad Hashir"
* profileImage: "https://api.dicebear.com/8.x/initials/png?seed=M\*|
* role: "user"
* shippingAddress: "Al noor society, Ahsanabab, karachi, 75990

## Collection: books

## authorName: "Frank Herbert"

## description: " Set in a distant future, the novel follows Paul Atreides,

## whose family assumes control of the desert planet

## Arrakis. As the only producer of a highly valuable resource, jurisdiction over Arrakis is contested among competing noble families. After Paul and his family are betrayed, the story explores themes of politics, religion, and man's relationship to nature, as Paul leads a rebellion to restore his family's reign."

## genre: "Sci-Fi"

## imageUr1: "https://i.ibb.co/LWc6YMR/book-cover-1750587511972.jpg™

## price: "6400"

## title: "Dune"

## Collection: cart

## book-id: "ZLJP5q7VrfHj1QP4lUpR"

## final-price: "5000"

## quantity: 1

## user-id: "JGXI4AWOD0fxVIP2BaBX5|prvK43"

## Collection: wishlist

## book-id: "2|WwNjZcJDjtLv5jnytT"

## user-id: \*YCsNCejnMtSzadZqLFBzj5mu6001'

## Collection: orders

# book-¡d: "HsHGEJхDs5tbhM5jеаОj"

# invoice-num: "1751221246544"

# price: "2500"

# quantity: 1

# status: "orderReceived"

# timestamp: June 29, 2025 at 11:20:46 PM UTC+5

# user-id: "YCsNCejnMtSzadZqLFBzj5muбoo1"

## Collection: Reviews

## book-id: \*2lWwNjZcJDjtLv5jnytT")

## comment: "nice book"

## is-liked: true

## rating: 5

## timestamp: June 29, 2025 at 10:55:34 AM UTC+5

## user-id: "ClruNPAlanemB3hmdr94vSGciPD3\*

# The subsequent sections will include:

## Application Screenshots

## User Interface:

## 

## 

## 

## Admin Panel:

## 

## 

## Source Code Documentation

## Admin Screen:

## AddBook.dart:

## /// Uploads the selected cover image to the imgbb.com image hosting service.

## /// Converts the selected image to a base64 string, sends it via a POST request

## /// to the imgbb API, and retrieves the URL of the uploaded image.

## /// Throws an exception if the upload fails.

## /// Parameters:

## /// - [imageFile]: The selected image file from the user's device.

## /// Returns:

## /// - A [Future<String>] containing the URL of the uploaded image.

## EditBook.dart:

## r /// Initializes form controllers with existing book data for editing.

## @override

## void initState() {

## // ...

## }

## /// Uploads a new cover image to imgbb and returns the image URL.

## /// Throws an exception if the upload fails.

## Future<String> uploadCoverImage(XFile imageFile) async {

## // ...

## }

## /// Validates the form, uploads new image if selected,

## /// updates the book using BookService, and shows feedback.

## Future<void> \_submitForm() async {

## // ...

## }

## /// Builds the UI for the Edit Book screen, including form fields,

## /// image selection, preview, and submit button.

## @override

## Widget build(BuildContext context) {

## // ...

## }

## View Book.dart:

## /// Initializes the state and fetches the list of books.

## @override

## void initState() {

## // ...}

## /// Fetches all books from the backend and updates the UI.

## Future<void> fetchBooks() async {

## // ...}

## /// Deletes a book after showing a confirmation dialog,

## /// then refreshes the book list.

## Future<void> \_deleteBook(String id) async {

## // ...}

## /// Displays a confirmation dialog before deleting a book.

## /// Returns true if the user confirms, false otherwise.

## Future<bool> \_showDeleteConfirmationDialog() async {

## // ...}

## /// Navigates to the EditBook screen and refreshes the list on return.

## void \_editBook(BookModel book) {

## // ...}

## /// Builds the UI for displaying all books in a list view,

## /// with options to edit or delete each book.

## @override

## Widget build(BuildContext context) {

## // ...}

## Auth\_Screen:

## Login.dart:

## /// Initializes animation values after a short delay to trigger fade-in and scale-in effects.

## @override

## void initState() {

## // ...}

## /// Handles user login by validating input, calling [UserService.loginUser],

## /// showing appropriate SnackBars, and navigating to the splash screen on success.

## void \_login() async {

## // ...}

## /// Builds the login screen UI, including email/password fields, animations,

## /// social sign-in buttons, and navigation to the registration screen.

## @override

## Widget build(BuildContext context) {

## // ...}

## Register.dart

## . /// Initializes animation values to trigger fade-in and scale-in effects on load.

## @override

## void initState() {

## // ...}

## /// Handles user registration using email/password form data.

## /// Also generates an avatar image URL from the name field.

## /// On success, clears input fields and navigates to login screen.

## void \_register() async {

## // ...}

## /// Generates a unique avatar URL using the first letter of the user's name.

## String getAvatarUrl(String name) {

## // ...}

## /// Builds the UI for the registration screen including form fields,

## /// password visibility toggle, social sign-in buttons, and navigation to login.

## @override

## Widget build(BuildContext context) {

## // ...}

## Services:

## BookServices.dart:

## . /// Fetches all books from the Firestore 'Books' collection.

## /// Returns a list of [BookModel] objects.

## static Future<List<BookModel>> getAllBooks() async {

## // ...}

## /// Retrieves a single book by its document ID from Firestore.

## /// Returns a [BookModel] if found, otherwise null.

## static Future<BookModel?> getBookById(String id) async {

## // ...}

## /// Adds a new book to the Firestore 'Books' collection using [BookModel].

## static Future<void> addBook(BookModel book) async {

## // ...}

## /// Deletes a book document from Firestore using its ID.

## static Future<void> deleteBook(String id) async {

## // ...}

## /// Updates an existing book document in Firestore using its ID and new [BookModel] data.

## static Future<void> updateBook(String id, BookModel book) async {

## // ...}

## Cart\_Services.dart

## . /// Adds a book to the user's cart in Firestore with specified quantity and price.

## /// Validates inputs before adding the document.

## static Future<void> addToCart({

## required String userId,

## required String bookId,

## required int quantity,

## required String finalPrice,

## }) async {

## // ...}

## /// Retrieves all cart items for a specific user from Firestore.

## /// Returns a list of cart item maps containing cartId, bookId, quantity, and price.

## static Future<List<Map<String, dynamic>>> getCartForUser(String userId) async {

## // ...}

## /// Removes a specific cart item for a user based on userId and bookId.

## static Future<void> removeFromCart(String userId, String bookId) async {

## // ...}

## /// Checks whether a specific book is already in the user's cart.

## /// Returns true if the book exists in the cart; false otherwise.

## static Future<bool> isBookInCart(String userId, String bookId) async {

## // ...}

## /// Updates the quantity and final price of a cart item for a specific user and book.

## static Future<void> updateCartItem({

## required String userId,

## required String bookId,

## required int quantity,

## required String finalPrice,

## }) async {

## // ...}

## CheckoutServices.dart

## . /// Places a new order in the Firestore 'Orders' collection with user ID, book ID,

## /// quantity, price, invoice number, and status.

## /// Validates input before saving.

## static Future<void> placeOrder({

## required String userId,

## required String bookId,

## required String price,

## required int quantity,

## required String invoiceNum,

## String status = 'orderReceived',

## }) async {

## // ...}

## /// Retrieves all orders placed by a specific user.

## /// Returns a list of order maps including orderId and order details.

## static Future<List<Map<String, dynamic>>> getOrdersForUser(String userId) async {

## // ...}

## /// Updates the status field of an order document by order ID.

## static Future<void> updateOrderStatus(String orderId, String newStatus) async {

## // ...}

## /// Deletes an order from Firestore using its document ID.

## static Future<void> deleteOrder(String orderId) async {

## // ...}

## ReviewServices:

## . /// Adds a new review for a book to the Firestore 'Reviews' collection,

## /// including user ID, rating, comment, and like status.

## static Future<void> addReview({

## required String bookId,

## required String userId,

## required bool isLiked,

## required double rating,

## required String comment,

## }) async {

## // ...}

## /// Fetches all reviews for a specific book, enriched with user data

## /// and formatted timestamp using timeago.

## static Future<List<Map<String, dynamic>>> getReviewsForBook(String bookId) async {

## // ...}

## /// Calculates and returns the average rating for a given book ID.

## /// Returns 0.0 if there are no ratings.

## static Future<double> getAverageRatingForBook(String bookId) async {

## // ...}

## UserServices.dart

## . /// Registers a new user using Firebase Auth and saves their details to Firestore.

## static Future<User?> registerUser(

## String email,

## String pass,

## Map<String, dynamic> userData,

## ) async {

## // ...}

## /// Logs in a user using Firebase Auth, stores user ID and role in SharedPreferences.

## static Future<User?> loginUser(

## String email,

## String pass,

## bool isRemeber,

## ) async {

## // ...}

## /// Logs out the current user and clears SharedPreferences.

## static Future<bool> logoutUser() async {

## // ...}

## /// Signs in user using Google, and saves their data in Firestore if new.

## static Future<User?> signInWithGoogle(Map<String, dynamic> userData) async {

## // ...}

## /// Signs in user using Facebook, and saves their data in Firestore if new.

## static Future<User?> signInWithFacebook(Map<String, dynamic> userData) async {

## // ...}

## WhishlistServices.dart:

## . /// Adds a book to the user's wishlist in Firestore.

## static Future<void> addToWishlist(String userId, String bookId) async {

## // ...}

## /// Retrieves all wishlisted books for a specific user.

## static Future<List<Map<String, String>>> getWishlistForUser(String userId) async {

## // ...}

## /// Removes a book from the user's wishlist using user ID and book ID.

## static Future<void> removeFromWishlist(String userId, String bookId) async {

## // ...}

## /// Checks if a specific book is already in the user's wishlist.

## static Future<bool> isBookWishlisted(String userId, String bookId) async {

## // ...}

## UserPages:

## All\_Books.dart:

## . /// Fetches all books from the database, extracts unique genres, and filters them based on the search query.

## Future<void> \_fetchBooks() async { ... }

## /// Updates the search query and re-filters books.

## void \_onSearchChanged() { ... }

## /// Retrieves user ID from shared preferences.

## Future<void> \_decideRoute() async { ... }

## /// Adds or removes a book from the user's wishlist.

## Future<void> toggleWishlist(BuildContext context, String bookId) async { ... }

## /// Fetches the current user's wishlist and stores book IDs.

## Future<void> \_fetchWishlist() async { ... }

## /// Adds a book to the cart with quantity 1 and shows a confirmation message.

## Future<void> \_addToCart(int index) async { ... }

## /// Initializes data: search listener, user ID, books, and wishlist.

## Future<void> \_initializeData() async { ... }

## /// Builds the bottom sheet for filtering by genre or sorting by price.

## Widget \_buildFilterSheet(BuildContext context) { ... }

## Cart\_page.dart:

## . /// Initializes the widget state by loading user ID and fetching cart items.

## @override

## void initState() { ... }

## /// Loads the logged-in user's ID from shared preferences and triggers cart fetch.

## Future<void> \_loadUserIdAndCart() async { ... }

## /// Fetches all cart items for the current user and updates UI.

## Future<void> \_fetchCartItems() async { ... }

## /// Updates quantity and recalculates final price of a specific cart item.

## Future<void> \_updateQuantity(int index, int newQuantity) async { ... }

## /// Removes a specific book from the cart.

## Future<void> \_removeItem(int index) async { ... }

## /// Calculates the total price of all items in the cart.

## double \_calculateTotal() { ... }

## /// Navigates to the checkout page with the current user's ID.

## void \_goToCheckout() { ... }

## /// Builds the entire cart page UI with cart list, total price, and checkout button.

## @override

## Widget build(BuildContext context) { ... }

## Checkout\_page.dart:

## . /// Initializes state by fetching user's saved addresses and cart items.

## @override

## void initState() {

## super.initState();

## fetchUserAddress();

## fetchCartItems();

## }

## /// Fetches the shipping and billing address of the user from Firestore.

## Future<void> fetchUserAddress() async { ... }

## /// Fetches cart items for the current user and calculates total amount.

## Future<void> fetchCartItems() async { ... }

## /// Opens a dialog to update the shipping (delivery) address and saves it to Firestore.

## Future<void> updateAddress() async { ... }

## /// Opens a dialog to update the billing address and saves it to Firestore.

## Future<void> updateBillingAddress() async { ... }

## /// Handles the "Pay Now" process:

## /// - Validates addresses

## /// - Generates invoice number

## /// - Places orders for each cart item

## /// - Clears the cart after placing the order

## /// - Shows a confirmation message

## void handlePayNow() async { ... }

## /// Builds the checkout UI, displaying:

## /// - Shipping and billing address sections

## /// - Payment method selection

## /// - Final payment button with total amount

## @override

## Widget build(BuildContext context) { ... }

## Singbook\_page.dart:

## /// Fetches book details from the database using the given bookId.

## Future<void> \_fetchBooks() async { ... }

## /// Sends a review to the database using user input (text + like/dislike).

## /// Then refreshes the list of reviews.

## Future<void> \_addReview() async { ... }

## /// Retrieves all reviews for the current bookId from the database.

## Future<void> \_fetchReviews() async { ... }

## /// Initializes the widget state by:

## /// - Fetching book details

## /// - Fetching book reviews

## @override

## void initState() {

## super.initState();

## \_fetchBooks();

## \_fetchReviews();

## }

## /// Builds the book detail UI, including:

## /// - Book cover, title, author, genre, and price

## /// - Buy and cart buttons

## /// - Book description

## /// - Review input (with like/dislike and comment)

## /// - Displaying all submitted reviews

## @override

## Widget build(BuildContext context) { ... }

## Home.dart:

## /// Initializes the widget state:

## /// - Loads the user's UID from shared preferences

## /// - Fetches the user's wishlist from Firestore

## Future<void> \_loadUserAndWishlist() async { ... }

## /// Fetches wishlist book IDs for the current user and retrieves each book’s details.

## /// Stores the complete list of wishlist books in `wishlistBooks`.

## Future<void> \_fetchWishlist() async { ... }

## /// Called when a bottom navigation item is tapped.

## /// - Updates `\_selectedIndex`

## /// - Changes page in `\_pageController`

## void \_onNavTapped(int index) { ... }

## /// Builds the main UI with:

## /// - PageView for navigation

## /// - AppBar with dynamic title

## /// - Drawer showing wishlist books with delete option

## /// - BottomNavigationBar with Home, Categories, Cart, Account

## @override

## Widget build(BuildContext context) { ... }

## /// Initializes the page when it's first created.

## @override

## void initState() {

## super.initState();

## \_loadUserAndWishlist();

## }

## User Manual

Detailed instructions for end-users on application installation, account creation, and feature utilization.

## Developer Guide

Technical documentation for future developers including setup procedures, architecture explanation, and maintenance protocols.