**Spis Treści**

[**1.** **Wstęp** 2](#_Toc188965694)

[1.1 Cele projektu 2](#_Toc188965695)

[1.2 Zakres projektu 2](#_Toc188965696)

[**2.** **Przegląd technologiczny** 3](#_Toc188965697)

[2.1 Flutter 3](#_Toc188965698)

[2.2 Firebase 3](#_Toc188965699)

[2.3 Architektura aplikacji 5](#_Toc188965700)

[2.3.1 Warstwa Interfejsu Użytkownika (UI Layer) 6](#_Toc188965701)

[2.3.2 Warstwa Zarządzania Stanem (State Management Layer) 6](#_Toc188965702)

[2.3.3 Repository Layer 8](#_Toc188965703)

[2.3.4 Warstwa Usług (Services Layer) 8](#_Toc188965704)

[2.3.5 Warstwa Usług Zewnętrznych (External Services Layer) 10](#_Toc188965705)

[2.4 Visual Studio Code 10](#_Toc188965706)

[**3.** **Określenie wymagań aplikacji** 12](#_Toc188965707)

[3.1 Wymagania funkcjonalne 12](#_Toc188965708)

[3.2 Wymagania niefunkcjonalne 13](#_Toc188965709)

[3.3 Przypadki użycia 14](#_Toc188965710)

[**4.** **Implementacja aplikacji** 19](#_Toc188965711)

[4.1 Struktura projektu i organizacja plików 19](#_Toc188965712)

[4.2 Implementacja warstwy interfejsu użytkownika 20](#_Toc188965713)

[4.3 Implementacja Adaptacyjnego Interfejsu Użytkownika 23](#_Toc188965714)

[**5.** **Funkcje backendowe** 28](#_Toc188965715)

[5.1  Zarządzanie bazą danych Firebase 28](#_Toc188965716)

# **Wstęp**

Biblioteki zawsze były ważne dla społeczeństwa pod względem transferu danych i interakcji z informacjami. Obecnie wiele bibliotek odeszło od starych metod prowadzenia dokumentacji i raportowania, a większość z nich decyduje się na narzędzia do zarządzania danymi oparte na oprogramowaniu. Ta zmiana podkreśla rosnące zapotrzebowanie na rozwiązania, które nie tylko optymalizują obsługę dużych ilości danych, ale także zapewniają przyjazne dla użytkownika doświadczenie.

Ta praca dyplomowa przedstawia aplikację do zarządzania biblioteką. Aplikacja będzie służyć jako kompleksowe narzędzie dla pracowników biblioteki, upraszczając zadania administracyjne, oraz dla członków biblioteki, zapewniając łatwy i intuicyjny dostęp do książek bibliotecznych. Co więcej, ten projekt jest multiplatformowy, co pozwoli użytkownikom korzystać z aplikacji na większości popularnych platform.

## 1.1 Cele projektu

Celem tego projektu jest dostarczenie rozwiązania, które uprości proces wykonywania wszystkich zadań związanych z biblioteką i umożliwi użytkownikom biblioteki dostęp do zasobów bibliotecznych w prosty i intuicyjny sposób; upewnienie się, że pracownicy biblioteki mają najbardziej zoptymalizowane narzędzia do zarządzania katalogiem książek, umożliwienie dokładnego śledzenia i zarządzania statusem wszystkich rezerwacji książek; oraz dbanie o dane użytkowników. Projekt zapewnia również użytkownikom biblioteki łatwy sposób przeglądania, wyszukiwania, oceniania i rezerwowania książek. Ostatecznym celem projektu jest stworzenie zintegrowanego systemu, który usprawni pracę administratorów biblioteki w zakresie zarządzania zbiorami, kontroli dostępu i organizacji przestrzeni, jednocześnie ułatwiając członekam biblioteki korzystanie z zasobów biblioteki.

## 1.2 Zakres projektu

Projekt obejmie pełny cykl zarządzania książkami, od inwentaryzacji po wypożyczanie, w ramach jednej aplikacji. Zakres aplikacji jest celowo ograniczony do fizycznej kolekcji zasobów biblioteki; książki elektroniczne i integracja z zewnętrznymi systemami bibliotecznymi są poza obecnym celem projektu. Rozwiązanie zostanie wdrożone na platformach Windows, Android, iOS i Web, gwarantując maksymalny dostęp i przenośność dla wszystkich użytkowników.

# **Przegląd technologiczny**

Zwykle opracowanie wygodnej i efektywnej aplikacji wymaga starannego doboru i wdrożenia odpowiednich technologii. Wybór tych technologii ma kluczowe znaczenie, ponieważ muszą one zapewniać wysoką wydajność, łatwą skalowalność i prostotę utrzymania aplikacji. W tej sekcji zostaną opisane różne technologie, które zostały wykorzystane podczas opracowywania aplikacji do zarządzania biblioteką.

## 2.1 Flutter

Główna część aplikacji została napisana przy użyciu frameworka Flutter - framework do tworzenia interfejsów użytkownika opracowany przez Google. Wybór Fluttera podyktowany został jego zdolnością do kompilacji natywnej na wiele platform z jednego kodu bazowego. Ta cecha jest kluczowa dla realizacji założenia o wieloplatformowości aplikacji, umożliwiając jej działanie na systemach Desktop (Windows, Linux, MacOS), Android, iOS oraz w przeglądarkach internetowych. Flutter opiera się na języku programowania Dart, również stworzonym przez Google, który charakteryzuje się wysoką wydajnością i jest zoptymalizowany pod kątem tworzenia interfejsów użytkownika.

Framework Flutter wyróżnia się wyjątkową elastycznością w obszarze UI. Architektura oparta na widżetach pozwala konstruować zarówno proste elementy interfejsu (przyciski, pola formularzy), jak i zaawansowane komponenty (złożone layouty, karty). Fundamentalną zasadą jest traktowanie każdego elementu interfejsu jako widżetu, co zapewnia spójność i modularność kodu. To daje ogromne pole do popisu również pod kątem estetycznego zgrania poszczególnych elementów interfejsu użytkownika. Dodatkowo narzędzie oferuje funkcję „hot reload” bezpośrednio odświeżenia aplikacji z podglądem zmian w kodzie na uruchomionym emulatorze lub urządzeniu. Dzięki temu, cały proces developmentu oraz testowania od strony interfejsu użytkownika jest znacznie bardziej płynny.

## 2.2 Firebase

Podejmując decyzję o wyborach fundamentu infrastruktury backendowej, należało zdecydować się na Firebase, czyli kompleksową platformę deweloperska prowadzoną przez firmę Google. Firebase to zestaw narzędzi zintegrowanych usług, które znacząco ułatwiają tworzenie, wdrażanie i skalowanie aplikacji, zwłaszcza działających w modelu SaaS – oprogramowanie jako usługa. W przypadku aplikacji do zarządzania biblioteką, w której z jednej strony ważne jest efektywne gospodarowanie danymi i informacjami o użytkownikach, z drugiej zaś zapewnienie bezpieczeństwa i niezawodności działania usług to Firebase to idealne rozwiązanie dla projektu, uzwalniające deweloperów od mocno nawet zadań infrastrukturalnych. Mogą oni skupić się na implementacji biznesu oraz rozwijaniu idei aplikacji. Na blog potrzebne są nam dwie usługi – Firestore i Authentication.

Firestore to system przechowywania danych w aplikacji. Firestore to bazowany na chmurze serwerowym, bardzo skalowalnej NoSQL bazie danych, która przechowuje dane w postaci zbiorów dokumentów w postaci JSON. Schemat taki jak baz NoSQL jest dość elastyczny, co sprawia, że idealnie nadaje się do zastosowania w aplikacji o schemacie ewolucyjnym, gdzie za każdym kolejnym sprintem może okazać się, że elementów aplikacji jest coraz więcej, co implikuje zmiany w strukturze przechowywania danych. Firestore przechowuje dane w formacie JSON, co ułatwia ich przetwarzanie w języku programowania Dart i ramce Flutter.

Ponadto Firestore ma możliwość synchronizacji danych w czasie rzeczywistym. Dzięki zastosowaniu „listener”, aplikacja może reagować na zdarzenia bazy danych, które na bieżąco zmieniają stan w aplikacji. Jest to kluczowe w kontekście zarządzania biblioteką. W przypadku rezerwacji książki przez innego użytkownika, aplikacja jest aktualizowana automatycznie i wyłącza możliwość zarezerwowania przez innego użytkownika tej samej książki. Naprzykład - Użytkownik A rezerwuje książkę „Portret Doriana Graya”. Dzięki synchronizacji w czasie rzeczywistym użytkownik B, który w tym samym czasie przegląda tę książkę na swoim urządzeniu, natychmiast widzi, że jej dostępność się zmieniła. Przed rezerwacją przez użytkownika A liczba dostępnych egzemplarzy wynosiła 1, ale po rezerwacji liczba ta zmienia się na 0. W rezultacie system uniemożliwia użytkownikowi B dokonanie rezerwacji tej samej książki, ponieważ jest już niedostępna. To samo dotyczy aktualizacji danych każdego użytkownika, na przykład zmiany adresu – każda wprowadzona zmiana jest natychmiast widoczna na wszystkich urządzeniach. Dzięki tej funkcjonalności dane w aplikacji są spójne i aktualne na każdej platformie.

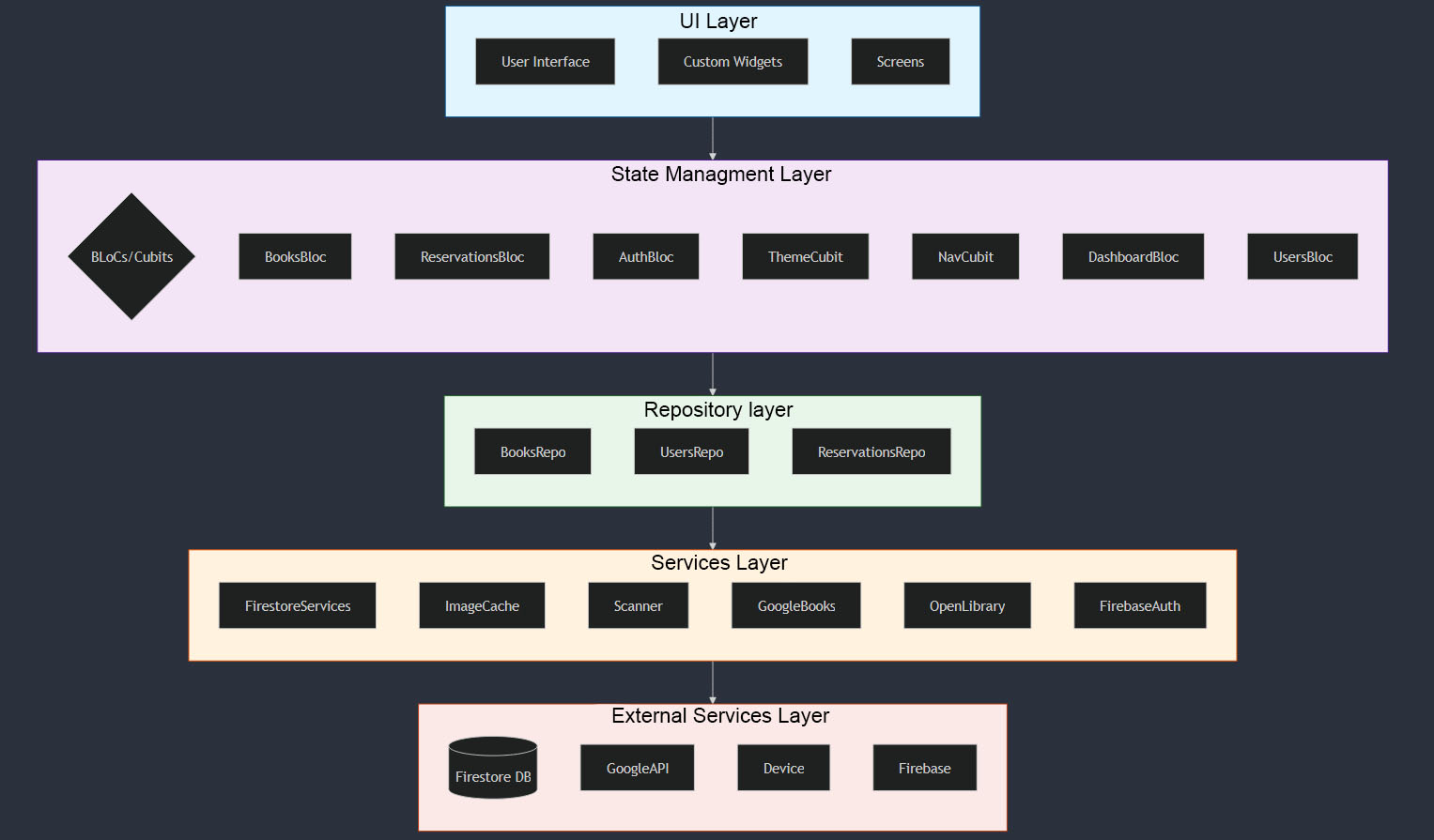
Oprócz tego Firestore oferuje też bardzo zaawansowane funkcje zapytań, umożliwiające filtrowanie i sortowanie danych na wiele różnych sposobów. Można to by było wykorzystane w stworzonej przez nas aplikacji do wyszukiwanie w katalogu zasobów: można zrobić to po nazwisku czy nazwie autora, po tytule czy opisie z jakiego gatunku jest dana książka, po możliwym zakresie daty wydania oraz filtrować rekordy na podstawie dostępności. Co ciekawe, Firestore samodzielnie indeksuje dane, a dodatek ich mojożliwośću łatwego wyszukiwania ich w wyniku zapytanste bez znacząco obniżającej wydajność, nawet przy zapytaniach właśnie w większej liczbie muṡiencia odebranej.

W projekcie wykorzystywany jest również Firebase Authentication. Korzystanie z Firebase Authentication umożliwia korzystanie z gotowego rozwiązania uwierzytelniania Google. W zakresie bezpieczeństwa, Firebase Authentication dba o dane użytkownika do uwierzytelniania, utrzymując je w postaci zaszyfrowanej, i stosuje najlepsze praktyki w zakresie ochrony haseł. System ten jest również zintegrowany z innymi usługami Firebase, takimi jak Firestore.

## 2.3 Architektura aplikacji

Architektura aplikacji to podstawa struktury organizacyjnej systemu informatycznego. Jest to kompleksowy plan systemu, opisujący układ, komponenty oraz sposób ich zachowania i interakcji. Architektura aplikacji jest rdzeniem systemu, co czyni ją niezwykle ważną w dzisiejszym świecie rozwoju oprogramowania. Definiuje ona elementy struktury aplikacji lub interfejsy, które je łączą. Oddziela również organizację komponentów systemu, aby zapewnić ich harmonijne funkcjonowanie. Dobrze zaprojektowana implementacja architektury zapewnia, że system jest łatwy w utrzymaniu, skalowaniu i zarządzaniu kodem. Jest to ważne dla długoterminowego cyklu życia aplikacji, ponieważ muszą one dostosowywać się do działania nowych procesów biznesowych.

Aplikacja do zarządzania biblioteką jest zbudowana zgodnie z najnowszymi wzorcami architektonicznymi i najlepszymi praktykami tworzenia oprogramowania: architektura zależna od wzorca BLoCa Business Logic Component oraz wzorca Repository. Pozwala to na jasny podział odpowiedzialności i efektywne zarządzanie kosztami kodu. Każda warstwa bazy ma wyraźnie określoną odpowiedzialność, co ułatwia zrozumienie, modyfikację i testowanie każdego komponentu systemu. Struktura warstwowa aplikacji została przedstawiona na poniższym diagramie (Rysunek 2.1)

Rysunek 2.1 – diagram „Struktura warstwowa aplikacji”

### 2.3.1 Warstwa Interfejsu Użytkownika (UI Layer)

Pierwsza, a zarazem najbardziej zewnętrzna warstwa - warstwa interfejsu użytkownika. Warto ją traktować jako punkt, w którym użytkownik wchodzi w interakcję z systemem. Ta warstwa jest odpowiedzialna za prezentację danych, oraz zapewnienie mechanizmów interakcji, takich jak wprowadzanie danych, wybieranie opcji czy nawigowanie między sekcjami aplikacji. Biorąc pod uwagę projekt „Aplikacja multiplatformowa do zarządzania biblioteka”, ekrany w tej warstwie organizują naszą zawartość do konkretnych sekcji, takich jak lista książek, ich atrybuty dostępności czy formularz rezerwacyjny. Należy również zastosować widżety, które określane są jako interaktywne komponenty pozwalające użytkownikowi na wykonywanie akcji.

### 2.3.2 Warstwa Zarządzania Stanem (State Management Layer)

Warstwa logiki biznesowej w projekcie wykorzystuje wzorzec BLoC (Business Logic Component) do zarządzania stanem i przetwarzania operacji. W przedstawionym przykładzie klasa BooksBloc pełni rolę głównego kontrolera odpowiedzialnego za obsługę zdarzeń związanych z książkami, takich jak ładowanie danych, wyszukiwanie czy usuwanie rekordów.

Architektura opiera się na strumieniach (Streams), gdzie każda interakcja użytkownika generuje zdarzenie (Event), które jest przetwarzane przez odpowiednią metodę w BLoC. Przykładowo, gdy użytkownik inicjuje ładowanie książek, wywoływane jest zdarzenie LoadBooksEvent. Metoda \_onLoadBooks w BooksBloc wykorzystuje wtedy repozytorium (BooksRepository) do pobrania strumienia danych z Firestore, przekształcając dokumenty Firestore w obiekty domenowe Book poprzez metodę fromMap. Stan aplikacji jest aktualizowany przy użyciu mechanizmu emit, gdzie na podstawie danych z repozytorium generowane są nowe stany. W przypadku błędów emitowany jest stan BooksError z komunikatem błędu. Ciekawym rozwiązaniem jest użycie emit.forEach do automatycznego zarządzania subskrypcjami strumieni - gdy nowe dane pojawią się w Firestore (np. inny użytkownik doda książkę), strumień w repozytorium aktualizuje się automatycznie, a BLoC natychmiast emituje nowy stan BooksLoaded, co powoduje odświeżenie interfejsu użytkownika.

Przykładowa ścieżka dla operacji usuwania książki:

* UI wywołuje metodę usuwania poprzez context.read<BooksBloc>().add(DeleteBook(bookId))
* BLoC przetwarza zdarzenie w metodzie \_onDeleteBook
* Repozytorium wykonuje operację usunięcia dokumentu w Firestore poprzez BooksFirestoreService
* Po sukcesie BLoC dodaje nowe zdarzenie LoadBooksEvent aby odświeżyć listę
* Zaktualizowane dane są pobierane z Firestore i emitowane do warstwy UI

Ważnym elementem jest separacja odpowiedzialności - BooksFirestoreService bezpośrednio komunikuje się z Firebase, BooksRepository transformuje dne między warstwą infrastruktury a domeną, a BLoC zarządza logiką przepływu danych i stanem aplikacji. Przykładowo mechanizm sortowania jest obsługiwany poprzez parametr sortType w zdarzeniu LoadBooksEvent, który przekazywany jest do zapytania Firestore w metodzie getBooksStream poprzez enum SortType.

Dodatkowe zabezpieczenia obejmują sprawdzanie aktualnego stanu przed emisją nowych wartości (if (state is! BooksLoaded)), co zapobiega niepotrzebnym przeładowaniom interfejsu. W przypadku operacji na ulubionych książkach, BooksFirestoreService wykorzystuje cache'owanie strumieni statusu polubień (\_favoriteCache), optymalizując liczbę zapytań do Firestore.

### 2.3.3 Repository Layer

Warstwa Repozytorium działa jako most między logiką biznesową a infrastrukturą danych, pełniąc kluczową rolę w zarządzaniu operacjami na danych. Na przykład klasa BooksRepository wykorzystuje serwis BooksFirestoreService do komunikacji z Firebase Firestore, ale zachowuje możliwość rozszerzenia o inne źródła danych dzięki implementacji interfejsu BaseRepository. Główną odpowiedzialnością tej warstwy jest transformacja danych między formatem surowym (np. dokumenty Firestore) a modelami domenowymi aplikacji, co widać w metodzie getAllBooks(), gdzie strumień dokumentów jest mapowany na obiekty Book przy użyciu metody Book.fromMap(data, id).

Komunikacja między BLoC a serwisami odbywa się poprzez hierarchię zależności: BLoC wywołuje metody repozytorium, które z kolei delegują operacje do odpowiednich serwisów Firestore. Przykładowo, gdy użytkownik ocenia książkę:

* BLoC przetwarza zdarzenie RateBookEvent, wywołując metodę rateBook() z repozytorium
* Repozytorium przekazuje żądanie do BooksFirestoreService
* Serwis Firestore wykonuje aktualizację dokumentu poprzez updateDocument() z ścieżką ratings.$userId
* Zmiana w Firestore automatycznie aktualizuje strumień danych w repozytorium
* BLoC przechwytuje nowe dane i emituje zaktualizowany stan do warstwy UI

Architektura umożliwia łatwą integrację z zewnętrznymi API – przykładowo, dodanie metody łączącej dane z Firestore i Google Books wymagałoby jedynie rozszerzenia repozytorium o nową metodę bez modyfikacji istniejących zależności.

### 2.3.4 Warstwa Usług (Services Layer)

Jako system pośredniczący między danymi pobieranymi ze źródeł zewnętrznych a logiką działającą w aplikacji, warstwa usług jest kluczową częścią każdej nowoczesnej architektury aplikacji. Abstrahuje ona od szczegółów implementacji podstawowych interfejsów API i obsługuje całą bezpośrednią komunikację z zewnętrznymi interfejsami API oraz usługami w ogóle. Głównym celem jest oddzielenie złożoności zewnętrznych integracji w celu osiągnięcia mniej skomplikowanego i bezpieczniejszego przepływu danych.

Rdzeniem warstwy usług jest Firestore Services, komponent obsługujący bezpośrednią komunikację z bazą danych Firebase Firestore. CRUD to skrót od Create, Read, Update i Delete - usługa ta obsługuje te operacje na danych i pozwala tworzyć, odczytywać, aktualizować i usuwać documenty w bazie danych Firestore. Zapewnia również buforowanie zapytań i techniki optymalizacji, umożliwiając wyższą wydajność i mniejsze obciążenie bazy danych.

Aplikacja korzysta również z innej funkcji Firebase - Firebase Authentication. Ten komponent procesu jest obsługiwany przez usługę uwierzytelniania. Posiada ona funkcje logowania i rejestracji, a także zarządzania tokenami użytkowników i sesjami.

Niezbędna jest również usługa Image Cache Service, która obsługuje buforowanie obrazów, takich jak okładki książek. Usługa ta buforuje często używane obrazy w celu zwiększenia wydajności aplikacji poprzez zmniejszenie ilości czasu i danych potrzebnych do załadowania często używanych obrazów. Usługa ta zarządza również pamięcią podręczną urządzenia, ułatwiając wykorzystanie dostępnej pamięci.

Aby ułatwić dodawanie nowych książek do biblioteki, stworzono usługę skanera. Usługa skanera umożliwia skanowanie kodów kreskowych ISBN za pomocą kamery urządzenia. Usługa ta zarządza dostępem do kamery, przetwarza zeskanowane obrazy i weryfikuje kody kreskowe ISBN, zapewniając ich poprawność i zgodność z formatem.

Usługa wyszukiwania jest kluczowym elementem systemu bibliotecznego, umożliwiającym efektywne przeszukiwanie zasobów biblioteki. Usługa obsługuje zaawansowane zapytania do bazy danych Firestore, umożliwiając użytkownikom szybkie znajdowanie książek, rezerwacji i innych zasobów. Implementuje różne strategie wyszukiwania, takie jak wyszukiwanie pełnotekstowe, wyszukiwanie według pól (np. tytuł, autor, ISBN) i filtrowanie wyników. Usługa optymalizuje zapytania do bazy danych za pomocą indeksowania i buforowania, co znacznie poprawia wydajność systemu przy dużej liczbie rekordów. Ponadto usługa wyszukiwania obsługuje wielojęzyczność i różne formaty danych, zapewniając wszechstronne możliwości wyszukiwania w całym systemie bibliotecznym.

Usługa ta jest szczególnie ważna w kontekście dużych zbiorów bibliotecznych, gdzie szybkie i dokładne wyszukiwanie ma kluczowe znaczenie dla użyteczności systemu. Implementuje również mechanizmy obsługi błędów i walidacji zapytań, zapewniając niezawodność i stabilność procesu wyszukiwania.

### 2.3.5 Warstwa Usług Zewnętrznych (External Services Layer)

Warstwa usług zewnętrznych jest podstawą architektury aplikacji, najniższą warstwą w architekturze aplikacji. Składa się z szeregu zasobów sprzętowych, interfejsów API i systemów zewnętrznych, z którymi aplikacja komunikuje się za pośrednictwem warstwy usług. Warstwa ta dostarcza dane i usługi, które są niezbędne do prawidłowego działania aplikacji.

Warstwa ta składa się z:

* Firestore DB: Baza danych Firestore, która służy jako główne repozytorium danych aplikacji. Zapewnia przechowywanie danych opartych na dokumentach, synchronizację danych w czasie rzeczywistym i skalowalną infrastrukturę do obsługi rosnących ilości informacji.
* Google API: Interfejsy programistyczne Google, w szczególności Google Books API, które dostarczają metadane książek, takie jak tytuły, autorzy, opisy i okładki. Integracja z tym interfejsem API umożliwia wzbogacenie katalogu biblioteki o szczegółowe informacje pochodzące od zewnętrznego dostawcy.
* Urządzenie: Zasoby i funkcje urządzenia użytkownika. Obejmuje to dostęp do kamery (używanej przez usługę skanera do skanowania kodów ISBN), zarządzanie lokalną pamięcią masową oraz obsługę uprawnień systemowych.
* Firebase: Platforma Firebase, która zapewnia szereg usług zaplecza, w tym system uwierzytelniania za pomocą Firebase Authentication, pamięć masową w chmurze, bazę danych w czasie rzeczywistym, a także narzędzia do analizy i monitorowania wydajności.

## Visual Studio Code

Visual Studio Code (VS Code) to szeroko stosowane lekkie narzędzie zintegrowanego środowiska programistycznego stworzone przez firmę Microsoft. Działa na kilku systemach operacyjnych, takich jak Windows, macOS i systemach uniksowych, i jest szeroko stosowany w prawie każdym języku programowania i technologii. VS Code, wydane w 2015 roku, szybko zyskało na popularności dzięki swojej prostocie, szybkości i łatwo rozszerzalnym wtyczkom, które pozwoliły każdemu programiście zaprojektować IDE do własnych celów. Jest to narzędzie typu open-source, którego kod źródłowy jest dostępny publicznie, umożliwiając społeczności.

W projekcie "Aplikacja hybrydowa do zarządzania biblioteką" wykorzystano VS Code do tworzenia aplikacji w technologii Flutter. Visual Studio Code posiada wiele funkcji, które spełniają wszystkie niezbędne potrzeby podczas pracy z Flutter. Funkcje rozszerzeń dla Flutter i Dart obejmują podświetlanie składni, uzupełnianie kodu, obsługę debugowania i integrację z Flutter DevTools. Ułatwia to debugowanie w czasie rzeczywistym i umożliwia testowanie aplikacji na emulatorach, a także na urządzeniach fizycznych, z dostępem do punktów przerwania, inspekcji zmiennych i dedykowanej konsoli. Odgrywa również bardzo ważną rolę funkcja Hot Reload, obsługiwana przez Flutter i zintegrowana z VS Code, umożliwia szybkie aktualizacje aplikacji w czasie wykonywania, znacznie przyspieszając rozwój i testowanie. VS Code szybko stało się jednym z najbardziej pożądanych narzędzi do budowania aplikacji Flutter wszystkich rozmiarów. Visual Studio Code w projekcie „Aplikacja hybrydowa do zarządzania biblioteką” znacznie ułatwiło proces tworzenia i testowania oraz przyczyniło się do wydajnego przepływu pracy i wysokiej jakości produktu końcowego.

# **Określenie wymagań aplikacji**

Rozwój każdej aplikacji rozpoczyna się od dokładnej analizy potrzeb użytkowników oraz zdefiniowania szczegółowych wymagań. Wymagania te określają cechy i funkcje aplikacji niezbędne do osiągnięcia celów projektowych i spełnienia potrzeb użytkowników. Podstawą sprawnego planowania, projektowania i wdrażania systemu są dobrze sformułowane wymagania, które charakteryzują się jasnością, kompletnością i wykonalnością. Wymagania można podzielić zasadniczo na dwie kategorie - funkcjonalne, które dotyczą specyficznych funkcji i działań systemu, oraz niefunkcjonalne, które odnoszą się do jakości i wydajności systemu.

## 3.1 Wymagania funkcjonalne

Projekt „Aplikacja hybrydowa do zarządzania biblioteką” zakłada stworzenie kompleksowego narzędzia wspierającego procesy biblioteczne. Główną ideą projektu jest stworzenie aplikacji, która w intuicyjny i wydajny sposób obsługuje wszystkie kluczowe aspekty działalności biblioteki. W tym celu konieczne jest opisanie wszystkich niezbędnych funkcji aplikacji.

Podstawowa funkcjonalność systemu powinna obracać się wokół trzech głównych obszarów: zasobów bibliotecznych, procesów rezerwacyjnych oraz zarządzania użytkownikami. System powinien zapewniać kompletne wyszukiwanie książek z zaawansowanymi filtrami, umożliwiając użytkownikom dokładne znalezienie interesujących ich tytułów.

Należy wdrożyć mechanizmy szczegółowego podglądu informacji o książce, w tym możliwość edycji danych przez uprawnionych użytkowników.

Musi być zaprojektowany proces rezerwacji książek jako wieloetapowy, obejmujący kompleksowe śledzenie statusów: od momentu złożenia reserwacji, poprzez jego realizację, aż po ostateczny zwrot. System będzie automatycznie śledził daty zwrotu i w razie potrzeby zmieniał status reserwacji książek, np. jeśli reserwacja jest przeterminowana. Przewidziano siedem kluczowych statusów rezerwacji: oczekujące, wypożyczone, zwrócone, przeterminowane, odrzucone, wygasłe oraz anulowane.

Zarządzanie użytkownikami będzie obejmować system autoryzacji wykorzystujący Firebase oraz system ról, który pozwoli użytkownikom logować się i definiować różne poziomy uprawnień. Użytkownicy na poziomie administratora będą również mogli dodawać, usuwać lub edytować informacje o użytkownikach.

Integralnym elementem systemu będzie zaawansowany mechanizm wyszukiwania, pozwalający na dynamiczne przeszukiwanie zasobów bibliotecznych. Użytkownicy mogą przeprowadzać złożone zapytania obejmujące książki, profile użytkowników, historię rezerwacji oraz ulubione pozycje.

Ważne będzie również zintegrowanie systemu dodawania nowych książek i uczynienie go szybkim i łatwym w użyciu. Wdrażając skaner ISBN i integrację z Google Books API, administratorzy będą mogli dodawać książki do elektronicznej bazy danych biblioteki.

Projektowany system charakteryzuje się wysoką elastycznością, kompleksowością oraz intuicyjnością interfejsu, zapewniając efektywne narzędzie zarówno dla bibliotekarzy, jak i czytelników.

## 3.2 Wymagania niefunkcjonalne

Druga kategoria wymagań - wymagania niefunkcjonalne, definiuje standardy operacyjne i ograniczenia aplikacji do zarządzania biblioteką. Koncentrują się one na cechach systemu wykraczających poza jego konkretną funkcjonalność, takich jak wydajność lub użyteczność aplikacji.

Aplikacja posiada wbudowany mechanizm buforowania obrazów, który zapewnia wydajność i płynne działanie. Działa to w następujący sposób - gdy użytkownik otwiera aplikację po raz pierwszy lub gdy nowa książka pojawia się w bazie danych, okładka książki jest ładowana bezpośrednio z łącza internetowego, a następnie buforowana, dzięki czemu okładka książki ładuje się znacznie szybciej w kolejnych sesjach tego samego użytkownika. System ten dynamicznie zarządza pamięcią lokalnego magazynu danych. Użytkownik może wybrać optymalny rozmiar pamięci podręcznej w ustawieniach aplikacji. Buforowanie pamięci jest ograniczone do maksymalnie 100 MB, co zapewnia efektywne wykorzystanie bez przeciążania zasobów, utrzymując responsywność aplikacji.

Aplikacja ma również możliwość zmiany motywu na ciemny lub jasny, co sprawia, że interfejs aplikacji jest bardziej elastyczny dla różnych użytkowników. Elementy interfejsu są konsekwentnie pozycjonowane i skalowane na różnych urządzeniach, zapewniając spójne wrażenia użytkownika na różnych rozmiarach ekranu.

## 3.3 Przypadki użycia

Przypadki użycia są narzędziem, które pozwala na szczegółowe określenie, jak użytkownicy będą korzystać z aplikacji. Każdy scenariusz opisuje wymagane funkcjonalności, które muszą zostać zaimplementowane, oraz dostarcza cennych informacji, które można wykorzystać podczas projektowania systemu. Dzięki przypadkom użycia możliwe jest zrozumienie, jak system będzie reagował na różne działania użytkowników i jakie cele muszą zostać osiągnięte, aby spełnić ich potrzeby.

Scenariusze przypadków użycia można łatwo przedstawić w formie tabeli, która zawiera osiem kluczowych elementów. Tytuł przypadku użycia stanowi krótki i zwięzły opis zadania, które ma zostać wykonane. Definicja aktora głównego wskazuje rolę użytkownika, z perspektywy której analizowany jest dany przypadek. Opis z kolei przedstawia krótki opis oczekiwanego zachowania systemu z punktu widzenia użytkownika. Warunki wstępne określają wymagania, które muszą zostać spełnione przed rozpoczęciem wykonywania zadania, a warunki końcowe pokazują zamierzony rezultat, który ma zostać osiągnięty po zakończeniu zadania. Scenariusz sukcesu to lista kroków, które użytkownik musi wykonać, aby osiągnąć swój cel, natomiast rozszerzenia uwzględniają alternatywne sytuacje, takie jak błędy czy inne nieoczekiwane zdarzenia, które mogą wystąpić w trakcie realizacji zadania. Na końcu tabeli określa się częstotliwość, czyli oszacowanie, jak często dany przypadek użycia będzie realizowany przez użytkowników systemu.

Poniżej przedstawione są scenariuszy przypadków użycia dla projektu „Aplikacja hybrydowa do zarządzania biblioteką”.

|  |  |
| --- | --- |
| Tytuł | Logowanie |
| Opis | Użytkownik chcę zalogować się do systemu przy użyciu adresu e-mail i hasła. |
| w | Użytkownik (Administrator oraz członek biblioteki) |
| Warunki wstępne | Użytkownik nie jest zalogowany. |
| Warunki końcowe | Użytkownik jest zalogowany do systemu. |
| Scenariusz sukcesu | 1. Użytkownik wprowadza adres e-mail.  2. Użytkownik wprowadza hasło.  3. Użytkownik klika przycisk „Zaloguj”.  4. System weryfikuje dane logowania.  5A. Weryfikacja zakończona sukcesem – system wyświetla ekran główny.  5B. Weryfikacja nieudana – system wyświetla komunikat o błędzie. |
| Rozszerzenia | 4a. Dane logowania mogą być nieprawidłowe.  4a1. Użytkownik mógł wprowadzić błędne dane.  4a2. Użytkownik mógł nie zarejestrować się wcześniej.  4a2. Użytkownik mógł nie zarejestrować się wcześniej.  4a3. Błąd połączenia internetowego |
| Częstotliwość | Wysoka |

Tabela 3.1 - Przypadek użycia: Logowanie

|  |  |
| --- | --- |
| Tytuł | Rezerwacja książki |
| Opis | Jako użytkownik chcę zarezerwować dostępne książki. |
| Główny aktor | Użytkownik (Administrator oraz członek biblioteki) |
| Warunki wstępne | Użytkownik jest zalogowany, książka jest dostępna. |
| Warunki końcowe | Książka zostaje zarezerwowana dla użytkownika. |
| Scenariusz sukcesu | 1. Użytkownik wybiera książkę.  2. System sprawdza dostępność książki - Jeśli książka jest dostępna, przycisk rezerwacji będzie aktywny.  3. Użytkownik klika przycisk rezerwacji.  4. System wyświetla okno dialogowe z potwierdzeniem rezerwacji.  5A. Role: Członek biblioteki - Użytkownik potwierdza rezerwację. (wybiera datę rezerwacji i datę zwrotu)  5B. Role: administrator - Użytkownik potwierdza rezerwację. (wybiera datę rezerwacji i datę zwrotu oraz członeka biblioteki)  6. System tworzy rezerwację i aktualizuje status książki. |
| Rozszerzenia | 3. System wyświetla komunikat o braku dostępności. |
| Częstotliwość | Wysoka |

Tabela 3.2 - Przypadek użycia: Rezerwacja książki

|  |  |
| --- | --- |
| Tytuł | Dodanie książki |
| Opis | |  | | --- | |  |  |  | | --- | | Jako administrator chcę dodać nowe książki do systemu. | |
| Główny aktor | |  | | --- | |  |  |  | | --- | | Administrator | |
| Warunki wstępne | Administrator jest zalogowany. |
| Warunki końcowe | Nowa książka zostaje dodana do systemu. |
| Scenariusz sukcesu | |  | | --- | | 1. Administrator wybiera opcję "Dodaj książkę". |  |  |  | | --- | --- | |  | 2. System wyświetla formularz informacji o książce. |  |  |  | | --- | --- | |  | 3. Administrator wprowadza szczegóły książki lub skanuje ISBN. |  |  |  | | --- | --- | |  | 4. System weryfikuje poprawność wprowadzonych danych. |  |  |  | | --- | --- | |  | 5. Administrator dodaje okładkę książki. |  |  |  | | --- | --- | |  | 6. Administrator zatwierdza formularz. |  |  |  | | --- | --- | |  | 7. System zapisuje informacje o książce. | |
| Rozszerzenia | |  | | --- | | 3a. Dostępny skan ISBN: |  |  |  | | --- | --- | |  | 3a1. System automatycznie wypełnia informacje o książce. |  |  |  | | --- | --- | |  | 4a. Wprowadzono nieprawidłowe dane: |  |  |  | | --- | --- | |  | 4a1. System wyświetla komunikaty o błędach walidacji. |  |  |  | | --- | --- | |  | 5a. Przesyłanie obrazu nie powiodło się: |  |  |  | | --- | --- | |  | 5a1. System wyświetla komunikat o błędzie przesyłania. | |
| Częstotliwość | Średnia |

Tabela 3.3 - Przypadek użycia: Dodanie książki

|  |  |
| --- | --- |
| Tytuł | Zarządzanie rezerwacjami |
| Opis | |  | | --- | |  |  |  | | --- | | Jako administrator chcę zarządzać rezerwacjami książek. | |
| Główny aktor | Administrator |
| Warunki wstępne | Administrator jest zalogowany. |
| Warunki końcowe | Status rezerwacji zostaje zaktualizowany. |
| Scenariusz sukcesu | |  | | --- | | 1. Administrator przechodzi do panelu rezerwacji. |  |  |  | | --- | --- | |  | 2. System wyświetla wszystkie aktywne rezerwacje. |  |  |  | | --- | --- | |  | 3. Administrator wybiera rezerwację. |  |  |  | | --- | --- | |  | 4. System wyświetla szczegóły wybranej rezerwacji. |  |  |  | | --- | --- | |  | 5. Administrator aktualizuje status rezerwacji. |  |  |  | | --- | --- | |  | 6. System zapisuje zmiany i powiadamia użytkownika. | |
| Rozszerzenia | |  | | --- | | 2a. Brak aktywnych rezerwacji: |  |  |  | | --- | --- | |  | 2a1. System wyświetla komunikat o braku aktywnych rezerwacji. |  |  |  | | --- | --- | |  | 5a. Aktualizacja statusu nie powiodła się: |  |  |  |  | | --- | --- | --- | |  | 5a1. System wyświetla komunikat o błędzie. |  | |
| Częstotliwość | Wysoka |

Tabela 3.4 - Przypadek użycia: Zarządzanie rezerwacjami

|  |  |
| --- | --- |
| Tytuł | Zarządzanie kontami użytkowników |
| Opis | Jako administrator chcę tworzyć i zarządzać kontami użytkowników. |
| Główny aktor | Administrator |
| Warunki wstępne | Administrator jest zalogowany. |
| Warunki końcowe | Konto użytkownika zostaje utworzone lub zmodyfikowane. |
| Scenariusz sukcesu | |  | | --- | | 1. Administrator przechodzi do sekcji zarządzania użytkownikami. |  |  |  | | --- | --- | |  | 2. System wyświetla interfejs zarządzania użytkownikami.  3. Administrator dodaje nowego użytkownika. |  |  |  | | --- | --- | |  | 4. Administrator wprowadza dane użytkownika (imię, e-mail, rola). |  |  |  | | --- | --- | |  | 5. System weryfikuje poprawność danych użytkownika. |  |  |  | | --- | --- | |  | 6. Administrator przypisuje uprawnienia użytkownikowi. |  |  |  | | --- | --- | |  | 7. System tworzy lub aktualizuje konto użytkownika. | |
| Rozszerzenia | |  | | --- | | 4a. Podany e-mail już istnieje: |  |  |  | | --- | --- | |  | 4a1. System wyświetla błąd o zduplikowanym e-mailu. |  |  |  | | --- | --- | |  | 5a. Nieprawidłowe dane (e-mail, peles) |  |  |  | | --- | --- | |  | 5a1. System wyświetla komunikat o błędzie. | |
| Częstotliwość | Średnia |

Tabela 3.5 - Przypadek użycia: Zarządzanie kontami użytkowników

|  |  |
| --- | --- |
| Tytuł | Generowanie raportów |
| Opis | Jako administrator chcę generować raporty systemowe. |
| Główny aktor | Administrator |
| Warunki wstępne | Administrator jest zalogowany. |
| Warunki końcowe | Raport zostaje wygenerowany. |
| Scenariusz sukcesu | |  |  | | --- | --- | |  | 1. Administrator wybiera okres daty raportu. |  |  |  | | --- | --- | |  | 2. System weryfikuje poprawność parametrów. |  |  |  | | --- | --- | |  | 3. System generuje raport. |  |  |  | | --- | --- | |  | 4. System wyświetla raport oraz eksportu go w formacie PDF. | |
| Rozszerzenia | |  | | --- | | 3a. Nieprawidłowy zakres dat: |  |  |  | | --- | --- | |  | 3a1. System wyświetla błąd parametrów. |  |  |  | | --- | --- | |  | 4a. Brak dostępnych danych: |  |  |  | | --- | --- | |  | 4a1. System generuje pusty raport. |  |  |  | | --- | --- | |  | 5a. Eksport nie powiódł się: |  |  |  | | --- | --- | |  | 5a1. System oferuje opcję ponowienia eksportu. | |
| Częstotliwość | Tygodniowa |

Tabela 3.5 - Przypadek użycia: Generowanie raportów

# **Implementacja aplikacji**

Tworzenie aplikacji to moment, gdy wizja projektu ożywa i zmienia się w funkcjonujący program gotowy do działania. W przypadku aplikacji bibliotecznej kluczowym zadaniem było opracowanie systemu, który precyzyjnie odpowiadałby zarówno na potrzeby użytkowników, jak i na wymagania techniczne.

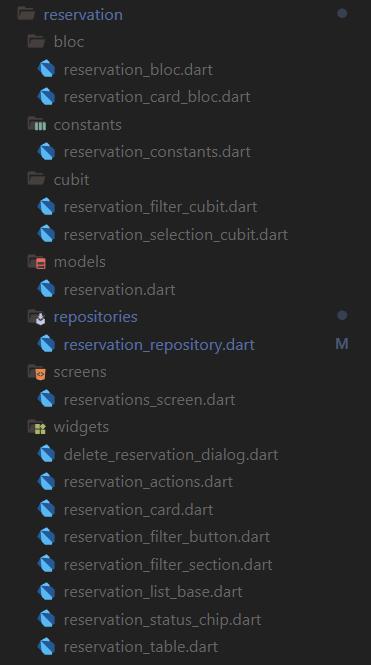
Cały proces miał na celu stworzenie aplikacji przyjaznej dla użytkownika, a jednocześnie wydajnej i niezawodnej w codziennym działaniu. W tym celu wykorzystano nowoczesny wzorzec projektowy BLoC (Business Logic Component), który umożliwia wyraźny podział między logiką biznesową a interfejsem użytkownika. To podejście działa jak precyzyjny zegar – każda część systemu wykonuje swoją rolę w harmonii z pozostałymi, zapewniając elastyczność i łatwość modyfikacji w przyszłości.

## 4.1 Struktura projektu i organizacja plików

Podczas opracowywania aplikacji hybrydowej do zarządzania biblioteką kluczowym elementem było zaprojektowanie projektu tak, aby funkcjonalność mogła być łatwo rozwijana, zarządzana i utrzymywana w przyszłości. W tym celu wybrano podejście „Feature-First”, polegające na grupowaniu kodu w osobne funkcje biznesowe. Takie rozwiązanie pozwala na zorganizowanie logicznie powiązanych komponentów w jednym miejscu, co ułatwia pracę deweloperów i zwiększa przejrzystość kodu. W przeciwieństwie do tradycyjnej separacji według typów technicznych, podejście to pozwala lepiej zarządzać zależnościami między komponentami, a także wspiera modułowość i ponowne wykorzystanie kodu.

Projekt podzielony jest na dwa główne katalogi. Pierwszy, „features”, zawiera wszystkie komponenty związane z funkcjami biznesowymi, takimi jak zarządzanie książkami, rezerwacje czy autoryzacja użytkowników. Każda z tych funkcji ma własną strukturę, co dodatkowo zwiększa czytelność i organizację projektu. Drugi katalog, „core”, przechowuje wspólne komponenty, takie jak modele, repozytoria, usługi i wspólne widżety. Jest to miejsce, w którym gromadzone są komponenty używane w różnych modułach aplikacji, co pozwala uniknąć redundancji kodu.

Dla każdej funkcjonalności w katalogu "features" przewidziano jednolitą strukturę wewnętrzną. Na przykład moduł związany z reserwacjami zawiera osobne foldery na zarządzanie stanem, modele danych, repozytoria, komponenty interfejsu użytkownika oraz ekrany (Rysunek 4.1) . Dzięki temu każda część związana z konkretną funkcjonalnością jest łatwa do znalezienia i zmodyfikowania w razie potrzeby.



Rysunek 4.1 - Struktura katalogó „features/reservation”

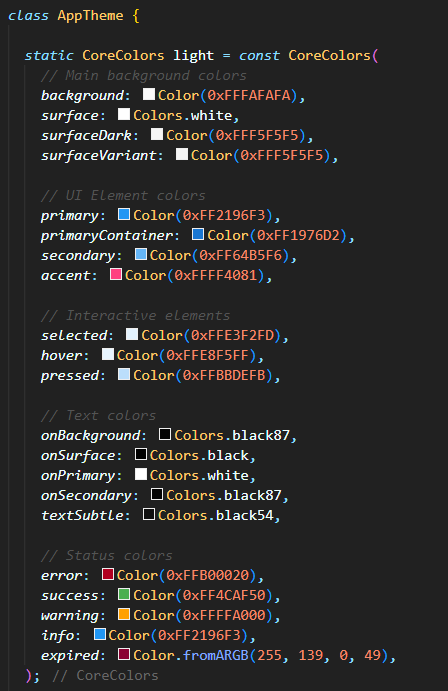
## 4.2 Implementacja warstwy interfejsu użytkownika

Każda nowoczesna aplikacja powinna posiadać interfejs użytkownika, którego głównym zadaniem jest zapewnienie płynnej interakcji z użytkownikami, zminimalizowanie ryzyka błędów i stworzenie pozytywnego doświadczenia. Dobry interfejs użytkownika jest łatwy w użyciu, spójny wizualnie i reaguje na różne urządzenia i potrzeby użytkowników.

W przypadku tego projektu interfejs został stworzony przy użyciu frameworka Flutter, co pozwoliło połączyć nowoczesne wzornictwo z wysoką funkcjonalnością. Projekt bazował na trzech głównych filarach: minimalizmie, responsywności oraz szeroko pojętej dostępności. Każdy aspekt interfejsu został zaprojektowany z myślą o prostocie i estetyce, jednocześnie dbając o to, by aplikacja była łatwa do obsługi niezależnie od platformy czy urządzenia.

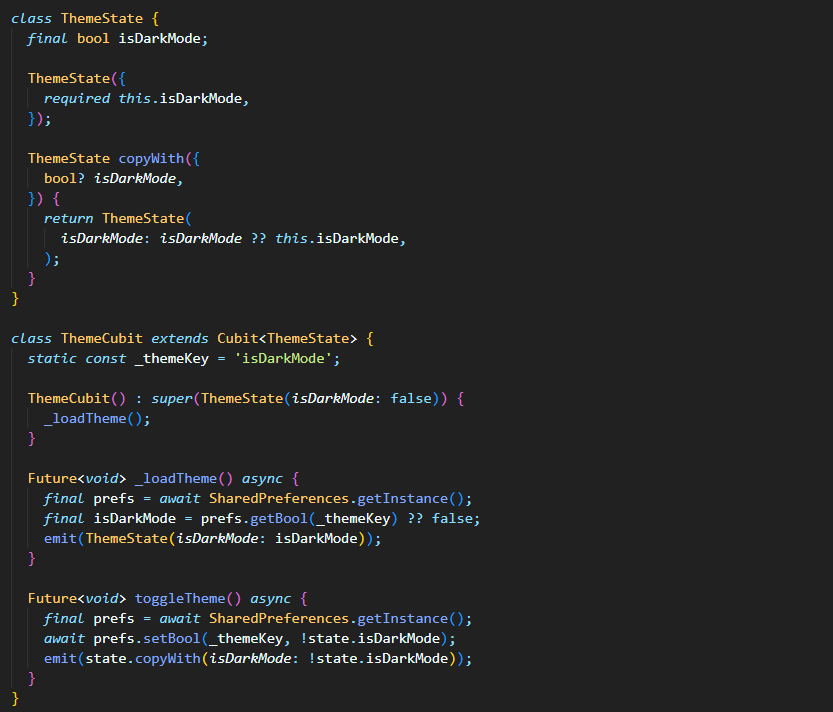
Kluczowym elementem projektu były niestandardowe widżety, które umożliwiły wielokrotne wykorzystanie tych samych komponentów w różnych miejscach aplikacji. Na przykład zamiast tworzyć oddzielne przyciski o podobnym wyglądzie dla każdego ekranu, opracowano uniwersalny widżet „custom\_button.dart”. Dzięki temu wystarczyło zaimportować gotowy komponent, by użyć go na dowolnym ekranie. Takie podejście nie tylko zwiększyło efektywność pracy nad kodem, ale również zapewniło spójność wizualną i ułatwiło rozwój aplikacji w przyszłości. Każdy widżet został zaprojektowany tak, aby był zarówno elastyczny, jak i łatwy do dostosowania w różnych kontekstach.

Aby ułatwić użytkownikowi pracę w różnych warunkach oświetleniowych, aplikacja obsługuje również tryby Light Mode i Dark Mode. Funkcjonalność ta nie tylko poprawiła użyteczność aplikacji, ale także jej estetykę. System został zaprojektowany z dbałością o szczegóły, aby zapewnić spójność wizualną, wydajność i dostępność dla szerokiego grona użytkowników. Fundamentem systemu motywów jest klasa CoreColors, która definiuje wszystkie kolory używane w aplikacji. Dla każdego trybu zdefiniowano osobną paletę kolorów (Rysunek 4.2).



Rysunek 4.2 - Palety kolorów

Następnie ThemeCubit (Rysunek 4.3) działa jako pośrednik między preferencjami użytkownika a systemem motywów. Używając wzorca BLoC, zarządza stanem motywu aplikacji i przechowuje informacje o tym, czy użytkownik preferuje tryb ciemny czy jasny. ThemeCubit komunikuje się z pamięcią urządzenia poprzez SharedPreferences, przechowując i odczytując preferencje użytkownika. Gdy użytkownik zmienia motyw, ThemeCubit tworzy nowy stan, który jest propagowany w całej aplikacji.



Rysunek 4.3 – ThemeCubit

Następnie na ekranie ustawień użytkownik będzie miał możliwość zmiany motywu aplikacji za pomocą prostego przełącznika. Gdy użytkownik przełączy jego pozycję, ekran ustawień (SettingsScreen) wywoła metodę toggleTheme w ThemeCubit, co zainicjuje zmianę motywu w całej aplikacji. Dzięki wykorzystaniu BlocBuilder, aplikacja automatycznie zareaguje na zmianę stanu, którą zarządza ThemeCubit, i zaktualizuje widok przełącznika w czasie rzeczywistym.

Nawigacja między ekranami jest również ważna w interfejsie użytkownika. System nawigacji w aplikacji został zaimplementowany przy użyciu szablonu BLoC i Navigator 2.0 Flutter. Głównym celem było stworzenie przewidywalnego, łatwego w utrzymaniu i rozszerzalnego systemu nawigacji, który obsługuje złożone scenariusze przepływu użytkownika. Głównym elementem systemu nawigacji jest NavigationCubit, który zarządza stanem nawigacji aplikacji. Przechowuje on bieżącą ścieżkę nawigacji i stos poprzednich ekranów. NavigationCubit tworzy nowe stany nawigacji, które są interpretowane przez RouterDelegate w celu aktualizacji widocznego ekranu.

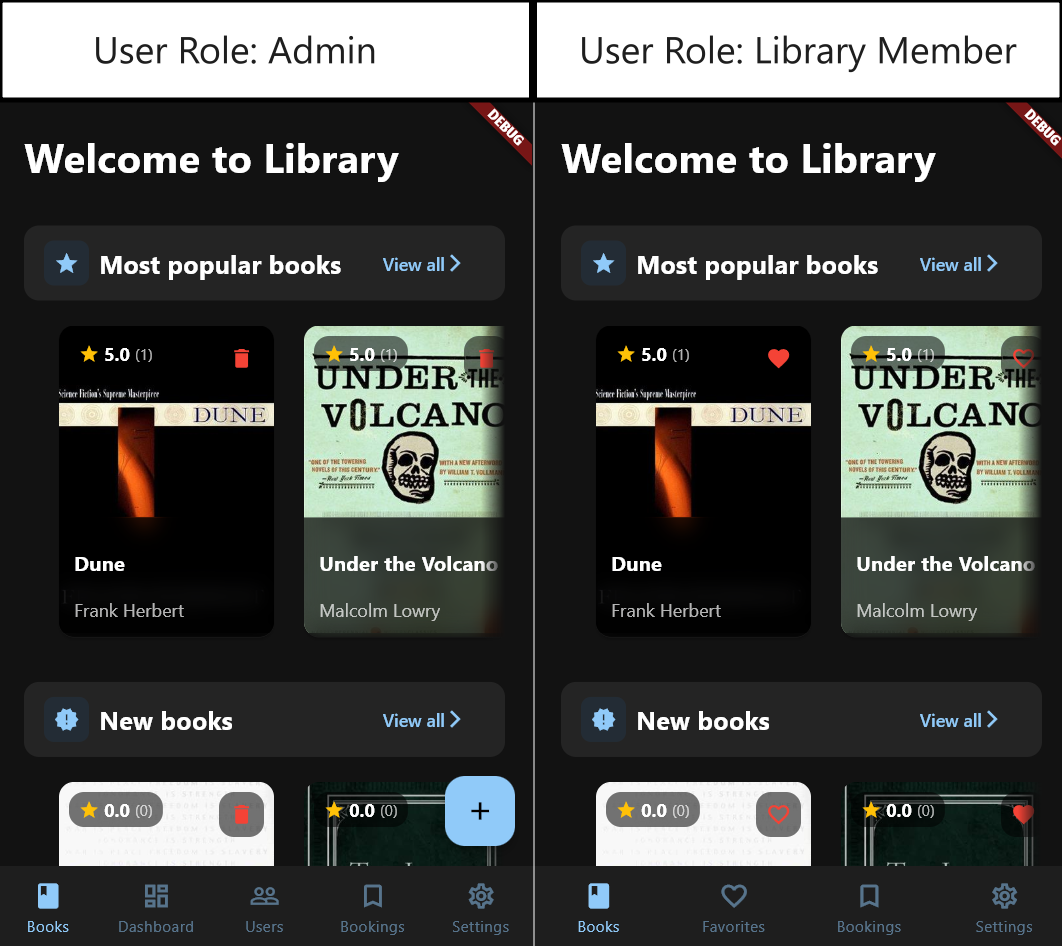
Projekt wykorzystuje dwa rodzaje nawigacji (Rysunek 4.4) - standardową nawigację od ekranu do ekranu i nawigację z parametrem, na przykład, jeśli użytkownik przechodzi do ekranu ze szczegółami książki, aplikacja używa metody navigateToBookDetails, w której navigate pobiera bookId i wypełnia dane widgetów (TextField, Book Cover Image) danymi książki z firebase.



Rysunek 4.4 – NavigationCubit

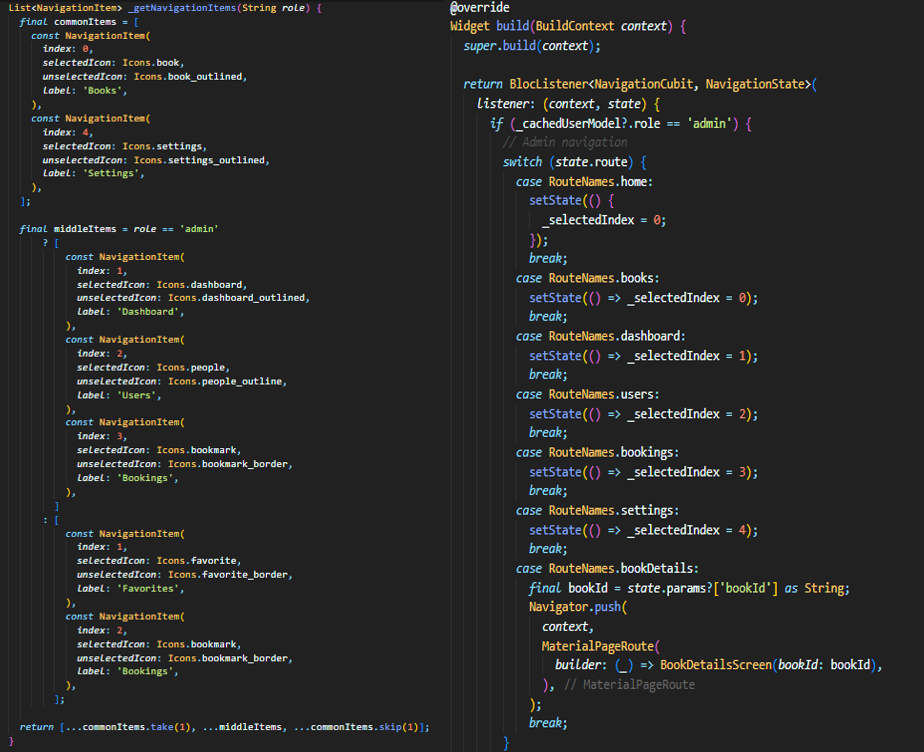
## 4.3 Implementacja Adaptacyjnego Interfejsu Użytkownika

Cały system opiera się na dynamicznej rekonfiguracji interfejsu w zależności od dwóch kluczowych czynników: roli użytkownika (admin/ członek biblioteki) oraz typu urządzenia (mobile/desktop). Na przykład ekran główny (Rysunek 4.5), który działa jak „przewodnik” koordynujący różne elementy interfejsu użytkownika, ma widżet „*Custom Navigation Bar*”, który jest inny dla użytkowników z rolą administratora i członka biblioteki. Ponadto administrator widzi przycisk dodawania nowej książki oraz przycisk usuwania książki, podczas gdy członek biblioteki nie ma dostępu do tej funkcji.



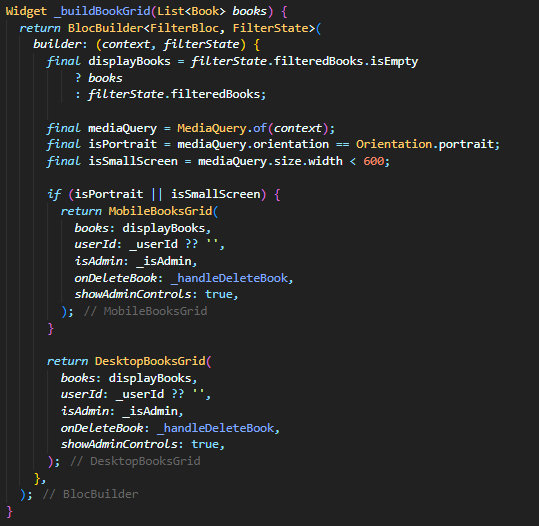
Rysunek 4.5 - Różnice w interfejsie w zależności od roli użytkownika

Metoda *\_getNavigationItems(String role)* (rysunek 4.6) generuje różne zestawy przycisków nawigacyjnych dla administratorów i członków biblioteki. Dla roli „admin” dodawane są specjalne sekcje, takie jak dashboard i zarządzanie użytkownikami. Również na przykład dla ekranu rezerwacji nawigator używa parametru *userID*, który pozwala na pobranie z bazy danych informacji o wszystkich rezerwacjach zalogowanego użytkownika. Z kolei administrator widzi wszystkie rezerwacje wszystkich użytkowników. Po wybraniu określonego elementu na Custom Navigation Bar wywoływana jest metoda nawigacji z NavigationCubit.



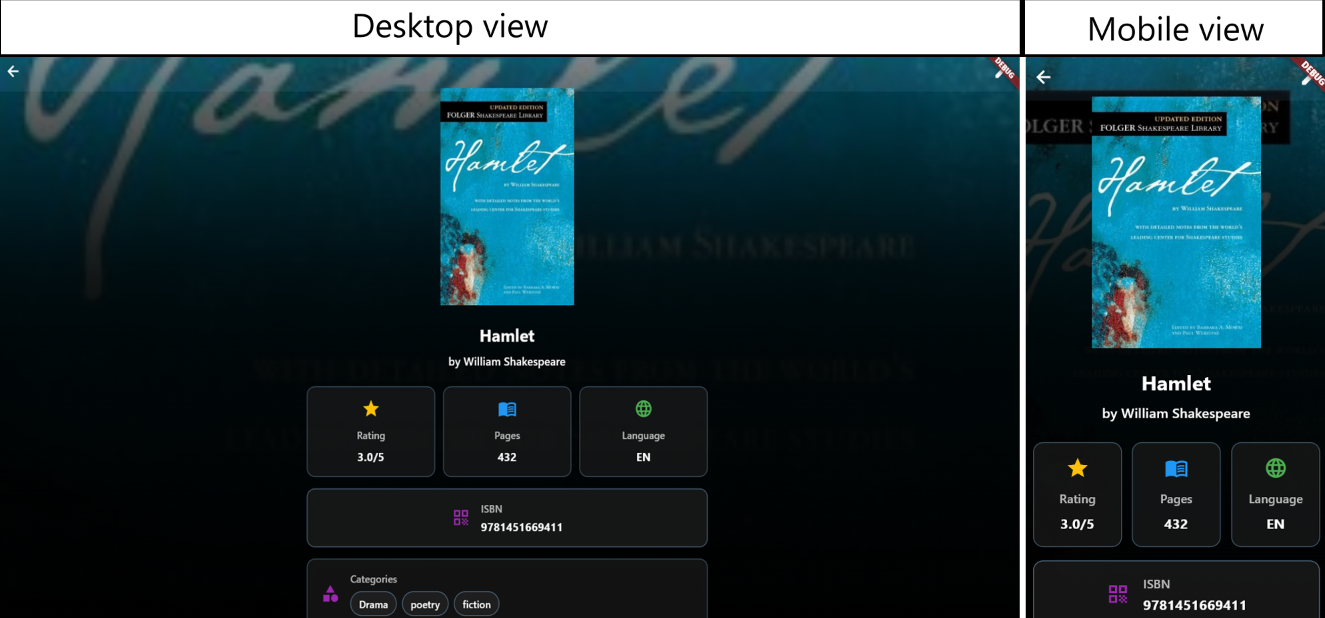
Rysunek 4.6 - Prezentacja metody \_getNavigationItems oraz navigacji

W aplikacji zaimplementowano dynamiczną zmianę rozmiaru i położenia elementów na ekranie, co sprawia, że interfejs jest przyjazny dla użytkownika zarówno na komputerach, jak i urządzeniach mobilnych. Na przykład na ekranie Books Screen dostępne są dwa widżety dla siatki książek – DekstopGrid oraz MobileGrid. DekstopGrid - wersja siatki przeznaczona dla ekranów desktopowych. W tej wersji liczba kolumn w siatce dynamicznie zmienia się w zależności od szerokości ekranu, co pozwala na optymalne wykorzystanie dostępnej przestrzeni. Ponadto widżet Book Card w tej wersji został zmodyfikowany, aby prezentować bardziej przemyślany układ wizualny. Mobile Book Grid - wersja siatki dla urządzeń mobilnych, która w rzeczywistości jest listą książek. Rozwiązanie to ułatwia nawigację po elementach na małych ekranach i jest bardziej odpowiednie dla ograniczonej przestrzeni. Przykład jego zastosowania pokazano na rysunku 4.7



Rysunek 4.7 - Przykład użycia dwóch różnych widżetów dla siatki książek

Ale także w aplikacji są niektóre widżety, które nie wymagają specjalnego widoku dla urządzeń mobilnych, po prostu używają dynamicznego rozmiaru. Na przykład w projekcie używam LayoutBuilder *final isSmallScreen = constraints.maxWidth < 600;* сo umożliwia automatyczną optymalizację układu w oparciu o dostępną przestrzeń i adaptacyjne wartości wypełnienia w celu dopasowania do punktu przerwania 600p . Przykład użycia w BoxContainer - *constraints: BoxConstraints(maxWidth: isSmallScreen ? double.infinity : 300,).* Jeśli isSmallScreen jest true (czyli ekran jest mały), maksymalna szerokość widżetu zostaje ustawiona na double.infinity. Oznacza to, że widżet zajmie całą dostępną szerokość kontenera nadrzędnego. Jeśli isSmallScreen jest false (czyli ekran jest większy, np. tablet lub komputer), maksymalna szerokość widżetu zostaje ograniczona do 300 pikseli (Rysunek 4.8). Ma to na celu zapobieganie nadmiernemu rozciąganiu widżetów na ekranach typu desktop. Jednak w niektórych przypadkach (takich jak karta rezerwacji) zdecydowano się nie dodawać tego ograniczenia, aby uniknąć tworzenia zbyt dużej pustej przestrzeni między widżetami.



Rysunek 4.8 - Widok ekranu „Book Details” dla desktop oraz urządzeń mobilnych

# **Funkcje backendowe**

Stworzenie wydajnej i bezpiecznej aplikacji to nie tylko kwestia atrakcyjnego interfejsu, ale także niezawodnego backendu, na którym opiera się jej działanie. W przypadku tego projektu kluczowe było stworzenie systemu gwarantującego płynny transfer danych, ochronę poufnych informacji i niezawodność w codziennym użytkowaniu. Backend, jako niewidoczny mechanizm, odpowiada za koordynację procesów, dzięki którym aplikacja działa płynnie i przewidywalnie.

W tym kontekście wykorzystano usługi Firebase – platformę oferującą narzędzia do zarządzania bazą danych oraz uwierzytelniania użytkowników. Każdy z tych komponentów działa jak tryb w precyzyjnym mechanizmie: baza danych synchronizuje informacje w czasie rzeczywistym oraz system autoryzacji chroni dostęp do zasobów. Dzięki integracji tych elementów aplikacja zyskuje nie tylko wydajność, ale także skalowalność, co jest niezbędne w dynamicznie rozwijających się projektach.

W kolejnych sekcjach szczegółowo omówiono klucze aspekty backendu: organizację bazy danych, bezpieczeństwo użytkowników oraz strategię przechowywania informacji, które razem tworzą spójny ekosystem aplikacji.

## 5.1  Zarządzanie bazą danych Firebase

W ramach wdrożenia systemu bibliotecznego kluczowym elementem architektury jest wykorzystanie Firebase Firestore, skalowalnej bazy danych NoSQL opartej na chmurze, zaprojektowanej do obsługi dynamicznych aplikacji z wymaganiami w czasie rzeczywistym. Firebase Firestore jest centralnym komponentem systemu, odpowiedzialnym za przechowywanie, zarządzanie i synchronizację danych pomiędzy użytkownikami, zapewniając wysoką dostępność i bezpieczeństwo. Elastyczny model dokumentów pozwala na efektywne modelowanie struktury danych dostosowanej do złożonych wymagań funkcjonalnych systemu bibliotecznego. Flutter komunikuje się z firebase za pomocą biblioteki cloud\_firestore.dart. Biblioteka ta umożliwia dostęp do interfejsu API Cloud Firestore.

Baza została zorganizowana w logiczne kolekcje dokumentów, odzwierciedlające kluczowe encje systemowe:

* **Books**: Przechowuje metadane książek, w tym tytuł, autora, ISBN, opis, dostępną liczbę egzemplarzy oraz oceny użytkowników.
* **Users**: Zawiera informacje o kontach użytkowników, powiązane z Firebase Authentication poprzez unikalne UID, wraz z rolą czytelnik oraz administrator.
* **Books\_reservation**: Śledzi historię rezerwacji, uwzględniając statusy, daty wypożyczeń i zwrotów oraz powiązania między użytkownikami a książkami.
* **Genres** i **Library\_settings**: Obsługują konfigurację systemu, takie jak kategorie książek i ustawienia operacyjne biblioteki.

Każdy dokument został zoptymalizowany pod kątem wydajności poprzez zastosowanie indeksów złożonych, pól wyszukiwawczych (searchTerms) oraz mechanizmów cache’owania. Struktura schematu uwzględnia wymagania biznesowe, takie asynchroniczną walidację danych (np. poprawność ISBN) i transakcyjne aktualizacje stanów rezerwacji.

Integracja z warstwą aplikacyjną w Flutter została zrealizowana poprzez abstrakcyjną klasę *BaseFirestoreService*, która enkapsuluje logikę operacji bazodanowych. BaseFirestoreService to abstrakcyjna klasa bazowa, która zapewnia podstawowe operacje CRUD (Create, Read, Update, Delete) dla Firebase Firestore. Na przykład wykonuje takie funkcje, jak dodawanie nowego dokumentu do firestore, pobieranie dokumentu z frestore, aktualizowanie dokumentu i tak dalej. Oprócz usługi BaseFirestoreService, w projekcie utworzono usługi Firestore dla każdej interakcji z określoną kolekcją w Firestore.

*books\_firestore\_service.dart* - Jest to klasa, która zarządza kolekcją książek. Ta klasa spełnia takie funkcje jak: Dodawanie nowych książek, zarządzanie ilością dostępnych egzemplarzy, system oceniania książek, obsługa ulubionych książek, generowanie statystyk wypożyczeń, sortowanie książek według różnych kryteriów.

*users\_firestore\_service.dart* - obsługuje kolekcję użytkowników. Ta klasa spełnia takie funkcje jak: tworzenie nowych profili użytkowników, aktualizacja danych użytkowników, pobieranie, strumienia danych użytkowników, wyszukiwanie użytkowników, weryfikacja istnienia użytkowników, zarządzanie uprawnieniami.

*reservations\_firestore\_service.dart* - zarządza rezerwacjami książek: tworzenie nowych rezerwacji, aktualizacja statusu rezerwacji, pobieranie historii rezerwacji, walidacja dat rezerwacji, obsługa przedłużeń wypożyczeń, monitorowanie przeterminowanych rezerwacji.

*genres\_firestore\_service.dart* - obsługuje kolekcję gatunków książek: pobieranie listy dostępnych gatunków, dodawanie nowych gatunków, usuwanie gatunków, inicjalizacja domyślnych gatunków, zarządzanie hierarchią gatunków.

*search\_service.dart* - implementuje funkcjonalność wyszukiwania: wyszukiwanie w wielu kolekcjach, obsługa filtrów wyszukiwania, wyszukiwanie przyrostowe, optymalizacja zapytań wyszukiwania, obsługa dodatkowych parametrów filtrowania.

*admin\_check\_service.dart* - zarządza uprawnieniami administratora: weryfikacja istnienia administratorów, sprawdzanie uprawnień użytkowników, logowanie prób dostępu administracyjnego.

## 5.2  Autoryzacja użytkowników