

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA PROGRAMOWANIE URZĄDZEŃ MOBILNYCH

Dokumentacja aplikacji Biblioteka

Autor:
Jakub Marek Tokarczyk
Mateusz Smaga

Prowadzący:
mgr inż. Dawid Kotlarski

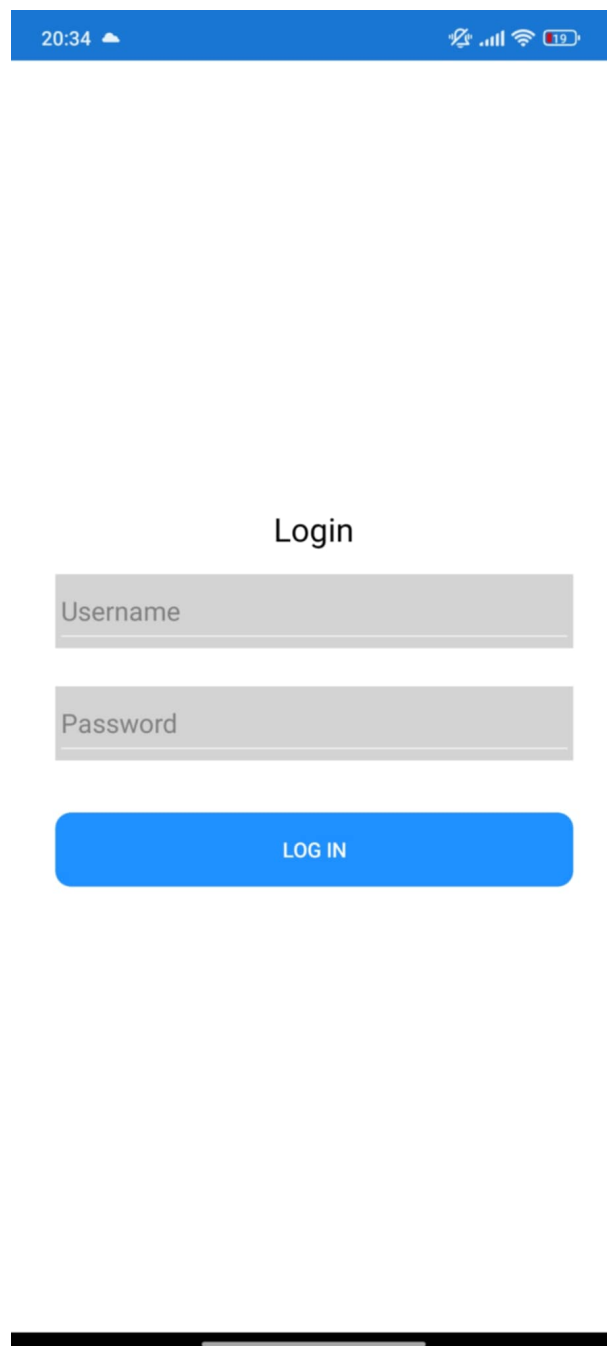
Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań	3
2. Określenie wymagań szczegółowych	6
3. Projektowanie	8
4. Implementacja	12
5. Testowanie	15
6. Podręcznik użytkownika	16
Literatura	17
Spis rysunków	17
Spis tabel	18
Spis listingów	19

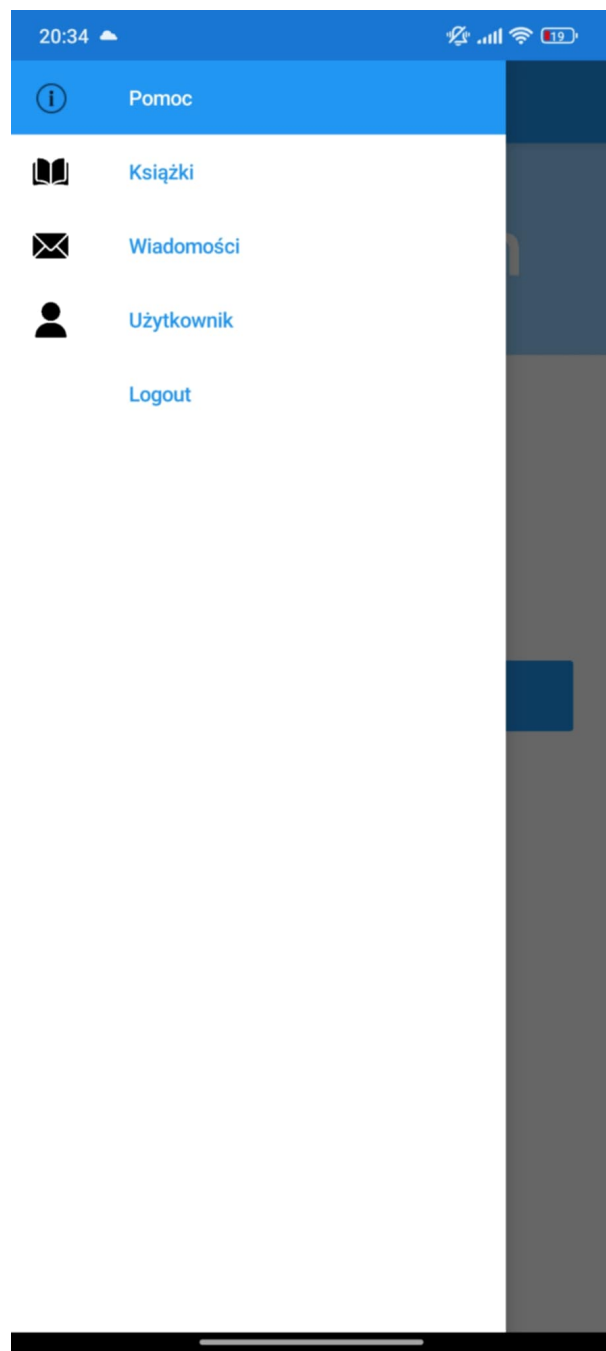
1. Ogólne określenie wymagań

Celem aplikacji Biblioteka jest dostarczenie użytkownikom tej aplikacji do efektywnego zarządzania własną kolekcją książek oraz umożliwienie interakcji z administratorem aplikacji. Aplikacja ma za zadanie uprościć proces kolekcjonowania książek poprzez funkcję skanowania, co pozwoli na szybsze wyszukiwanie książek w bazie danych. Dzięki temu, użytkownicy mogą szybko i wygodnie dodawać nowe pozycje do swojej biblioteki. Ponadto, aplikacja będzie posiadać opcje związane z edytowaniem oraz organizacją książek, co pozwoli na lepsze zarządzanie osobistymi ulubionymi zestawami książek. Aplikacja ma także za zadanie wspierać komunikację między hostem a użytkownikami. Host, pełniący rolę administratora aplikacji, będzie mógł przekazywać użytkownikom istotne informacje, takie jak wprowadzone aktualizacje, powiadomienia o zmianach, nowościach w aplikacji, zmianach w regulaminie lub inne komunikaty dotyczące samej obsługi aplikacji. Aplikacja Biblioteka skierowana jest do wszystkich osób, które posiadają własne kolekcje książek, niezależnie od ich wielkości. Może to być użyteczne narzędzie zarówno dla miłośników literatury jak i dla osób zajmujących się sprzedażą, wymianą, czy też wynajmem książek. Dzięki prostym funkcjom aplikacja pozwoli użytkownikom na organizację książek według własnych kryteriów (np. ulubione) a także na łatwe przeglądanie i zarządzanie zebranymi informacjami. Aplikacja będzie posiadała system logowania oraz rejestracji nowych użytkowników. Użytkownik, który nie posiada jeszcze konta, będzie mógł je utworzyć poprzez podanie podstawowych danych czyli nazwa użytkownika oraz hasło. Gdy użytkownik utworzy konto, będzie mógł zalogować się za pomocą nazwy użytkownika i hasła. Zakładka "Użytkownik" będzie zawierała szczegółowe informacje o aktualnie zalogowanym użytkowniku. W tej sekcji użytkownik będzie mógł przeglądać i edytować osobiste informacje takie jak: imię, nazwisko, nazwę użytkownika oraz adres e-mail. Zakładka "Książki" będzie pełniła funkcję zarządzania biblioteką książek użytkownika. Użytkownicy będą mieli możliwość dodania książek, ich edycje lub usunięcie. Zakładka "Wiadomości" pozwoli użytkownikom na przeglądanie otrzymanych wiadomości od administratora.



The image shows a mobile application interface for login. At the top is a blue status bar with the time 20:34, a cloud icon, and icons for cellular signal, Wi-Fi, and a battery level of 19%. Below the status bar is a white rectangular area containing the title "Login" in bold black text. Under the title are two light gray input fields: the first is labeled "Username" and the second is labeled "Password". Below these fields is a blue rounded rectangular button with the text "LOG IN" in white. At the very bottom of the screen is a black horizontal bar representing the mobile home indicator.

Rys. 1.1. Logowanie



Rys. 1.2. Layout

2. Określenie wymagań szczegółowych

Aplikacja ma na celu ułatwienie użytkownikom gromadzenia, organizowania i zarządzania książkami w jednym miejscu, zarówno ulubionymi, starymi, jak i nowymi. Dzięki zastosowaniu nowoczesnych funkcji, takich jak dodawanie zdjęć książek, logowanie przez biometrię oraz automatyczne dostosowywanie interfejsu do jakości oświetlenia w danym pomieszczeniu czy otoczeniu, aplikacja zapewnia intuicyjne i wygodne narzędzie dla miłośników literatury. Użytkownicy będą mogli łatwo dodawać książki do swojej biblioteki, edytować dane książek oraz przeglądać je w zorganizowany sposób. Głównym założeniem jest, aby każdy użytkownik mógł przechowywać wszystkie swoje książki w jednym miejscu, niezależnie od tego, czy są to książki fizyczne, które już posiada, czy też tytuły, które planuje przeczytać. W aplikacji można dodać książki ręcznie, wpisując tytuł, autora, wydawnictwo i inne dane, ale również można wykorzystać aparat do dodania zdjęcia okładki danej książki. Po zalogowaniu do aplikacji użytkownik może przechodzić między różnymi funkcjami a logowanie może odbywać się za pomocą tradycyjnego hasła lub biometrii w przypadku administratora, czyli odcisku palca. Aplikacja automatycznie wykrywa warunki oświetleniowe otoczenia użytkownika. Jeśli w pomieszczeniu lub na zewnątrz jest jasno, aplikacja przechodzi w tryb dzienny, z jasnym interfejsem, który ułatwia przeglądanie książek i nawigację. Kiedy robi się ciemno, aplikacja automatycznie przełącza się w tryb nocny, zmieniając interfejs na ciemne kolory, co jest przyjemniejsze dla oczu i bardziej oszczędza baterię urządzenia. Użytkownik może także w każdej chwili ręcznie zmienić tryb, jeśli preferuje inny wygląd aplikacji. Dzięki integracji z aparatem, użytkownik może robić zdjęcia książek które dodaje do swojej biblioteki, lub robić zdjęcia swojej osoby i dodawać je jako zdjęcie profilowe. Każda z funkcji ma być prosta i intuicyjna a interfejs aplikacji będzie skoncentrowany na łatwości obsługi. Aplikacja będzie wykorzystywać bazę danych Firebase do przechowywania wszystkich danych użytkowników oraz ich książek. Firebase oferuje dynamiczne i bezpieczne przechowywanie danych, zapewniając przy tym synchronizację między różnymi urządzeniami. Oznacza to, że użytkownik będzie mógł korzystać z aplikacji na różnych urządzeniach, a jego dane zawsze będą aktualne, niezależnie od tego, z którego urządzenia korzysta. Środowiskiem programistycznym, w którym będzie tworzona aplikacja, jest Visual Studio 2019 z użyciem frameworka Xamarin. Zdecydowaliśmy się na to środowisko, ponieważ Visual Studio 2019 oferuje bogaty zestaw narzędzi deweloperskich oraz dobrą integrację z Xamarin, co pozwala na tworzenie aplikacji mobilnych działających na różnych systemach operacyjnych, takich jak Android

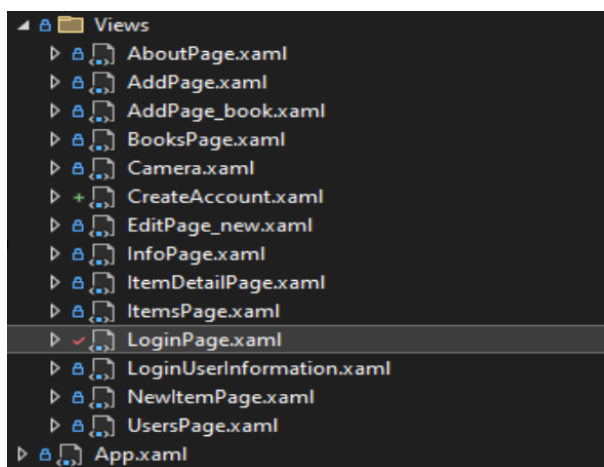
i iOS, z użyciem jednej bazy kodu. Xamarin umożliwia tworzenie aplikacji natywnych, co oznacza, że aplikacja będzie działać szybko i płynnie na obu platformach. Dodatkową zaletą jest wsparcie dla interfejsów mobilnych i łatwa integracja z różnymi funkcjami urządzeń, takimi jak aparat czy biometria. Zaletą Visual Studio 2019 jest jego rozbudowany interfejs, liczne wtyczki i narzędzia ułatwiające debugowanie oraz tworzenie aplikacji, a także wsparcie dla wielu języków programowania. Dzięki temu środowisku możemy szybko rozwijać aplikację a Xamarin zapewnia, że aplikacja będzie działać na wielu platformach bez konieczności pisania oddzielnych wersji na każdą z nich. Wadą może być to, że Visual Studio bywa zasobożerne, co sprawia, że może działać wolniej na słabszych komputerach. Xamarin natomiast, chociaż umożliwia pisanie aplikacji na różne platformy, nie jest tak elastyczny i wydajny jak natywne środowiska programistyczne dedykowane tylko dla jednej platformy.

3. Projektowanie

Przygotowanie narzędzi, czyli git i visual studio. Do zarządzania zmianami kodu projektu używamy Gita jako systemu kontroli wersji, co pozwala na utrzymanie historii zmian, rozdzielenie funkcji na osobne gałęzie czyli branch oraz sprawne współdzielenie pracy w zespole. Git umożliwia śledzenie każdej modyfikacji, co jest przydatne przy pracy zespołowej. Każdy nowy element lub poprawka jest rozwijana w dedykowanej gałęzi, co ułatwia testowanie i eliminuje ryzyko nadpisania zmian w kodzie. Projekt jest zamieszczony na GitHubie, co zapewnia łatwy dostęp do kodu dla całego zespołu. GitHub umożliwia automatyczne uruchamianie testów po każdym commicie dzięki GitHub Actions, co znacząco poprawia stabilność projektu i eliminuje błędy zanim zostaną wdrożone. Visual Studio Code to główne środowisko pracy wykorzystywane do tworzenia aplikacji mobilnych. Wersja 2022 jest idealna do tego projektu, ponieważ oferuje nowoczesne funkcje optymalizujące wydajność programistyczną, co przekłada się na bardziej wydajną i przyjazną pracę nad kodem. Visual Studio 2022 jest wyposażone w zaawansowane narzędzia wspierające debugowanie, testowanie, zarządzanie błędami oraz integrację z systemem kontroli wersji Git. Zawiera on zintegrowane wsparcie dla Xamarin, co umożliwia tworzenie aplikacji mobilnych, które działają zarówno na systemie Android, jak i iOS, wykorzystując wspólny kod. Dzięki temu Xamarin pozwala na jednoczesne pisanie kodu dla obu platform, co zmniejsza czas i nakład pracy potrzebny na stworzenie wersji aplikacji dla różnych urządzeń. Xamarin umożliwia dostęp do natywnych funkcji systemów operacyjnych Android i iOS, dzięki czemu aplikacja może korzystać z zaawansowanych możliwości, takich jak GPS, aparat, czujniki oraz inne funkcje dostępne na urządzeniach mobilnych. W projekcie wykorzystywany jest język C#, który jest w pełni kompatybilny z Xamarin i doskonale nadaje się do tworzenia aplikacji mobilnych. C# jest językiem obiektowym, co ułatwia organizację kodu, a także posiada wbudowaną obsługę wyjątków i bezpieczne typowanie, co znacząco zmniejsza liczbę błędów. Biblioteka FirebaseDatabase.net to biblioteka umożliwiająca integrację aplikacji .NET z Firebase Realtime Database, czyli bazą danych typu NoSQL oferowaną przez Google, która służy do przechowywania i synchronizacji danych w czasie rzeczywistym dla aplikacji mobilnych i internetowych. Biblioteka zapewnia prosty interfejs API, który pozwala na szybkie dodawanie nowych obiektów, ich modyfikację lub usuwanie. Firebase Realtime Database synchronizuje dane w czasie rzeczywistym, co oznacza, że wszelkie zmiany dokonane przez jednego użytkownika lub urządzenie są natychmiast widoczne dla pozostałych użytkowników. Biblioteka ta umożliwia integrację z Firebase Authentication, co pozwala na bezpieczny dostęp do danych. Dzięki kom-

patybilności z Xamarin FirebaseDatabase.net jest idealnym wyborem do aplikacji mobilnych tworzonych w .NET, umożliwiając szybkie połączenie aplikacji z Firebase i korzystanie z bazy danych w czasie rzeczywistym.

Menu wysuwane posiada następujące zakładki: Książki jest to główna sekcja aplikacji, prowadząca do zbioru zapisanych książek. Po kliknięciu w ikonkę przeniesie nas, do zbioru naszych dotychczasowych książek, gdzie będzie można je dodawać i usuwać. Wiadomości to sekcja z wiadomościami lub uaktualnieniami aplikacji od hosta do użytkowników aplikacji, po wejściu w sekcję, będziemy mogli zobaczyć wiadomości od hosta dotyczące na przykład aktualizacji aplikacji. Użytkownik to sekcja profilu użytkownika, gdzie po wejściu w nią, można zobaczyć dane na swój temat takie jak imię, nazwisko, e-mail i również je edytować. Logout to opcja, która umożliwia wylogowanie się z aplikacji, co zapewni prywatność i bezpieczeństwo dostępu, po kliknięciu zostaniemy przekierowani do ekranu logowania do aplikacji. Każda z ikon oraz tekst w menu są w jasnym niebieskim kolorze, co pasuje do ogólnego motywu kolorystycznego aplikacji i zapewnia spójność wizualną. Samo tło menu jest białe, co daje kontrast w stosunku do ciemnego motywu głównego formularza i ułatwia czytelność opcji nawigacyjnych.



Rys. 3.1. Widoki

Widoki w projekcie zazwyczaj reprezentują interfejs użytkownika. W tym przypadku, odpowiedzialne są za:

AddPage.xaml - Widok umożliwiający dodanie nowego elementu np. wpisu, zawiera pola do wprowadzania danych,

AddPagebook.xaml - jest dedykowany dodawaniu książek, umożliwia wprowadzanie szczegółów książki, takich jak tytuł, autor,

BooksPage.xaml - zawiera liste książek, umożliwia interakcje z książkami takie jak usuwanie, edytowanie,

Camera.xaml - jest to widok integrujący funkcję aparatu, który pozwala użytkownikowi na robienie zdjęć,

CreateAccount.xaml - jest to Widok rejestracji nowego użytkownika, zawiera pola do wprowadzenia danych,

EditPage.xaml - Widok umożliwiający edycję danych istniejącego elementu, pozwala na ich modyfikację oraz zapisanie zmian,

InfoPage.xaml - strona wyświetlająca szczegółowe informacje o konkretnym elemencie, może być używana np. do pokazania pełnych informacji o książce czy użytkowniku,

ItemDetailPage.xaml - może służyć do wyświetlania szczegółowych danych o konkretnym wpisie, podobnie jak InfoPage.xaml,

ItemsPage.xaml - widok wyświetlający listę elementów, np. użytkowników czy książek, zawiera listę z możliwością interakcji np. kliknięcie w element

LoginPage.xaml - strona logowania do aplikacji, zawiera pola na login i hasło oraz przycisk logowania. Oferuje także opcje utworzenia konta,

LoginUserInformation.xaml - widok wyświetlający informacje o zalogowanym użytkowniku, pokazując dane takie jak imię i nazwisko użytkownika, e-mail,

NewItemPage.xaml - widok umożliwiający dodanie nowego elementu,

UsersPage.xaml - widok listy użytkowników.

Plik .xaml to specjalny typ pliku używany w technologiach takich jak WPF, Xamarin.Forms czy MAUI, który służy do definiowania interfejsu użytkownika w sposób deklaratywny. Umożliwia to tworzenie widoków graficznych poprzez określanie układu elementów interfejsu, takich jak przyciski, pola tekstowe, siatki, obrazy, etykiety czy inne kontrolki wizualne. Dzięki składni XML, pliki .xaml są przejrzyste i łatwe do edycji zarówno dla programistów, jak i projektantów. Można w nich precyzyjnie opisać, jakie elementy interfejsu mają się znajdować na danej stronie, jakie mają mieć style, rozmiary czy rozmieszczenie. Z kolei plik .cs, czyli tak zwany kod-behind jest bezpośrednio powiązany z plikiem .xaml i zawiera logikę aplikacji, która jest związana z daną stroną lub widokiem. Każdy plik .xaml ma odpowiadający mu

plik .xaml.cs, gdzie znajdują się definicje metod i zdarzeń, takich jak reakcje na kliknięcia przycisków czy wprowadzanie danych przez użytkownika. Na przykład, jeśli mamy stronę LoginPage.xaml, która wyświetla formularz logowania, jej kod-behind będzie znajdować się w pliku LoginPage.xaml.cs. W tym miejscu można zaimplementować zachowanie tej strony, np. walidację danych logowania, przejście do innej strony po udanym logowaniu lub wyświetlenie komunikatu o błędzie. Dzięki podziałowi na .xaml i .cs, programiści mogą oddzielić część wizualną od logiki biznesowej, co czyni kod bardziej przejrzystym i łatwiejszym w utrzymaniu. Takie podejście wspiera również pracę zespołową, ponieważ różne osoby mogą jednocześnie zajmować się projektowaniem interfejsu i implementowaniem jego funkcjonalności.

4. Implementacja

```

1
2 <FlyoutItem Title="About" Icon="icon_about.png">
3     <ShellContent Route="AboutPage" ContentTemplate="{DataTemplate
4         local:AboutPage}" />
5 </FlyoutItem>
6 <FlyoutItem Title="Browse" Icon="icon_feed.png">
7     <ShellContent Route="ItemsPage" ContentTemplate="{DataTemplate
8         local:ItemsPage}" />
9 </FlyoutItem>
10 <FlyoutItem Title="Users" Icon="">
11     <ShellContent Route="UsersPage" ContentTemplate="{DataTemplate
12         local:UsersPage}" />
13 </FlyoutItem>
14 <MenuItem Text="Logout" StyleClass="MenuItemLayoutStyle" Clicked="
15     OnMenuItemClicked">

```

Listing 1. Zakładki

Listing definiuje strukturę nawigacyjną aplikacji mobilnej za pomocą Xamarin.Forms i komponentu Shell, który organizuje widoki w formie menu. Tworzy on główne zakładki takie jak About, Browse i Users reprezentujące poszczególne strony aplikacji, do których użytkownik może szybko się przemieścić poprzez menu wysuwane aplikacji. Element MenuItem dla przycisku "Logout" służy do wylogowania.

```

1
2     FirebaseClient firebaseClient = new FirebaseClient("https://
3     bibliotekapro-eeddd-default-rtdb.firebaseio.com/");
4
5 //add
6 public async Task<bool> Save(User user)
7 {
8     var data = await firebaseClient.Child(nameof(User)).PostAsync(
9     JsonConvert.SerializeObject(user));
10    if(!string.IsNullOrEmpty(data.Key))
11    {
12        return true;
13    }
14    return false;
15 }
16
17 //save
18 public async Task<List<User>> GetAll()

```

```
16 {
17     return (await firebaseClient.Child(nameof(User)).OnceAsync<User>()).Select(item => new User
18     {
19         Email = item.Object.Email,
20         Name = item.Object.Name,
21         Image = item.Object.Name,
22         Id = item.Key
23     }).ToList();
24 }
25 //get id
26 public async Task<User>GetById(string id)
27 {
28     return (await firebaseClient.Child(nameof(User) + "/" + id).
29     OnceSingleAsync<User>());
30 }
31 //update
32 public async Task<bool> Update(User user)
33 {
34     await firebaseClient.Child(nameof(User) + "/" + user.Id).
35     PutAsync(JsonConvert.SerializeObject(user));
36     return true;
37 }
38 //delete
39 public async Task<bool>Delete(string id)
40 {
41     await firebaseClient.Child(nameof(User) + "/" + id).DeleteAsync
42     ();
43     return true;
44 }
```

Listing 2. Metody

Listing ten umożliwia podstawowe operacje na danych użytkowników w bazie Firebase Realtime Database przy użyciu FirebaseClient. Zawiera pięć metod: Save do dodawania nowych użytkowników, GetAll do pobierania wszystkich użytkowników, GetById do wyszukiwania użytkownika według identyfikatora, Update do aktualizowania danych konkretnego użytkownika oraz Delete do usuwania użytkownika z bazy na podstawie jego identyfikatora.

```
1 public class User: IUserWithId
2 {
3     public string Id { get; set; }
4     public string Name { get; set; }
5     public string Email { get; set; }
6     public string Image { get; set; }
7
8     public string Login { get; set; }
9     public string Password { get; set; }
10 }
```

Listing 3. Obiekt User

Klasa user dziedziczy po interfejsie IUserWithId aby mieć unikalne id, dzięki czemu każdy obiekt w systemie ma własny, unikalny identyfikator, który odróżnia go od innych, co pełni kluczową rolę w zarządzaniu danymi, zwłaszcza w aplikacjach, które obsługują wiele obiektów.

```
1 public interface IUserWithId
2 {
3     string Id { get; set; }
4 }
```

Listing 4. Interfejs IUserWithId

Jest to unikalne ID, które będzie używane do identyfikacji obiektów. Celem tego jest zapewnienie, że wszystkie klasy implementujące ten interfejs mają pole Id, które identyfikuje obiekt. Interfejs IUserWithId: Gwarantuje, że każdy obiekt typu User będzie miał właściwość Id. Dzięki temu można traktować obiekty różnych typów w podobny sposób, na przykład w systemach identyfikacji.

```
1 public class Book: IUserWithId
2 {
3     public string Id { get; set; }
4     public string Name { get; set; }
5     public string Author { get; set; }
6     public string Image { get; set; }
7 }
```

Listing 5. Klasa book

Klasa Book implementuje interfejs IUserWithId, definiując właściwości Id, Name, Author oraz Image, które reprezentują identyfikator książki, jej nazwę, autora oraz obraz, co umożliwia przechowywanie i zarządzanie danymi o książkach.

5. Testowanie

6. Podręcznik użytkownika

Spis rysunków

1.1. Logowanie	4
1.2. Layout	5
3.1. Widoki	9

Spis tabel

Spis listingów

1.	Zakładki	12
2.	Metody	12
3.	Obiekt User	14
4.	Interfejs IUserWithId	14
5.	Klasa book	14