**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

Факультет комп’ютерних наук та кібернетики

Кафедра теорії та технології програмування

**Звіт**

до лабораторної роботи на тему

Розробка програми для мобільних пристроїв з використанням БД, аресної книги та візуалізацією геопросторових даних

Виконала студентка 4-го курсу

Недавня Юлія Володимирівна

**Київ – 2023**

[Опис завдання 2](#_Toc781006528)

[Використані технології 3](#_Toc323913412)

[Опис роботи програми 4](#_Toc252119033)

[Додаток А 9](#_Toc703473214)

[Додаток В 12](#_Toc1202509425)

[Додаток С 13](#_Toc128487325)

# **Опис завдання**

Метою роботи є формування базових знань та умінь розробляти для мобільного пристрою програми, яка використовує збереження даних у

БД, використовує адресну книгу, обробляє і візуалізує геопросторові дані з використанням відповідних АРІ. За бажанням студента погодженням з

боку викладача, зміст роботи може бути змінений.

Загальні вимоги:

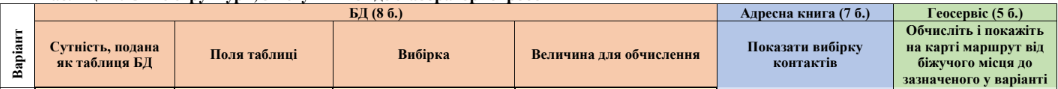
- Має бути документ (PDF) з описом вимог до програмного продукту (ТЗ)

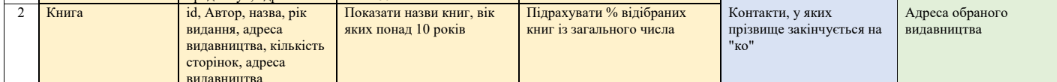
- Програма повинна реалізовувати запис/читання даних з файлу або локальної БД

- Програма повинна мати пункт з інформацією про розробника та його/її фото

- Повинні бути застосовані базові принципи UX-дизайну та ергономіки для мобільних пристроїв

Було взято 2 варіант.





# **Використані технології**

* Integrated Development Environment (IDE): Microsoft Visual Studio 2019 обрано головною IDE для розробки додатку з основою на фреймворку .NET.
* Multi-platform App UI (MAUI): використано MAUI для створення мобільного додатку, що працює на кількох платформах iOS та Android.
* Database: SqlLite Local Database
* Programing language: використано мову програмування C#, яка є однією з основних мов у фреймворку .NET.

# **Опис роботи програми**

1. Робота з базою даних

*Програмна логіка*

Доступ до БД з мобільного застосунку був розроблений з використанням мови програмування С# та технології SqlLite Local Database.

В коді даного додатку було створено об'єкт класу DatabaseContext, який містить ряд методів для роботи з базою даних. Для роботи з даними використовується асинхронний метод, який забезпечує ефективність та швидкість роботи з базою даних (Додаток А).

Метод AddItemAsync<TTable> відповідає за збереження нового запису в базу даних. Вибирається таблиця, яку потрібно створити, і після цього виконується додавання нового запису до таблиці.

Метод UpdateItemAsync<TTable> використовується для оновлення запису в таблиці. Знову ж таки, потрібно вибрати відповідну таблицю та виконати оновлення.

Метод DeleteItemAsync<TTable> відповідає за видалення запису з таблиці. Знову ж таки, потрібно вибрати відповідну таблицю та виконати видалення.

Метод GetAllAsync<TTable> використовується для отримання всіх записів з таблиці. Також є метод GetFileteredAsync<TTable>, який дозволяє отримати записи з таблиці, що відповідають певному фільтру.

Метод GetItemByKeyAsync<TTable> дозволяє знайти запис в таблиці за його ідентифікатором. Цей метод дуже корисний, оскільки сутності в більшості випадків мають унікальний ідентифікатор.

Отже, робота з базою даних в мобільному додатку виконується за допомогою певного набору методів, які дозволяють створювати, зберігати та видаляти записи, а також отримувати дані з таблиці залежно від потреб користувача.

*UI*

Далі наведено скріншоти застосунку, на яких видно, як саме користувач може оперувати із сутностями (книжками): додавати їх до БД, змінювати та видаляти.

Знизу екрану є форма з усіма необхідними полями для заповнення книжки значеннями. Зверху користувач може бачити список доданих книжок(рис. 1, 2).

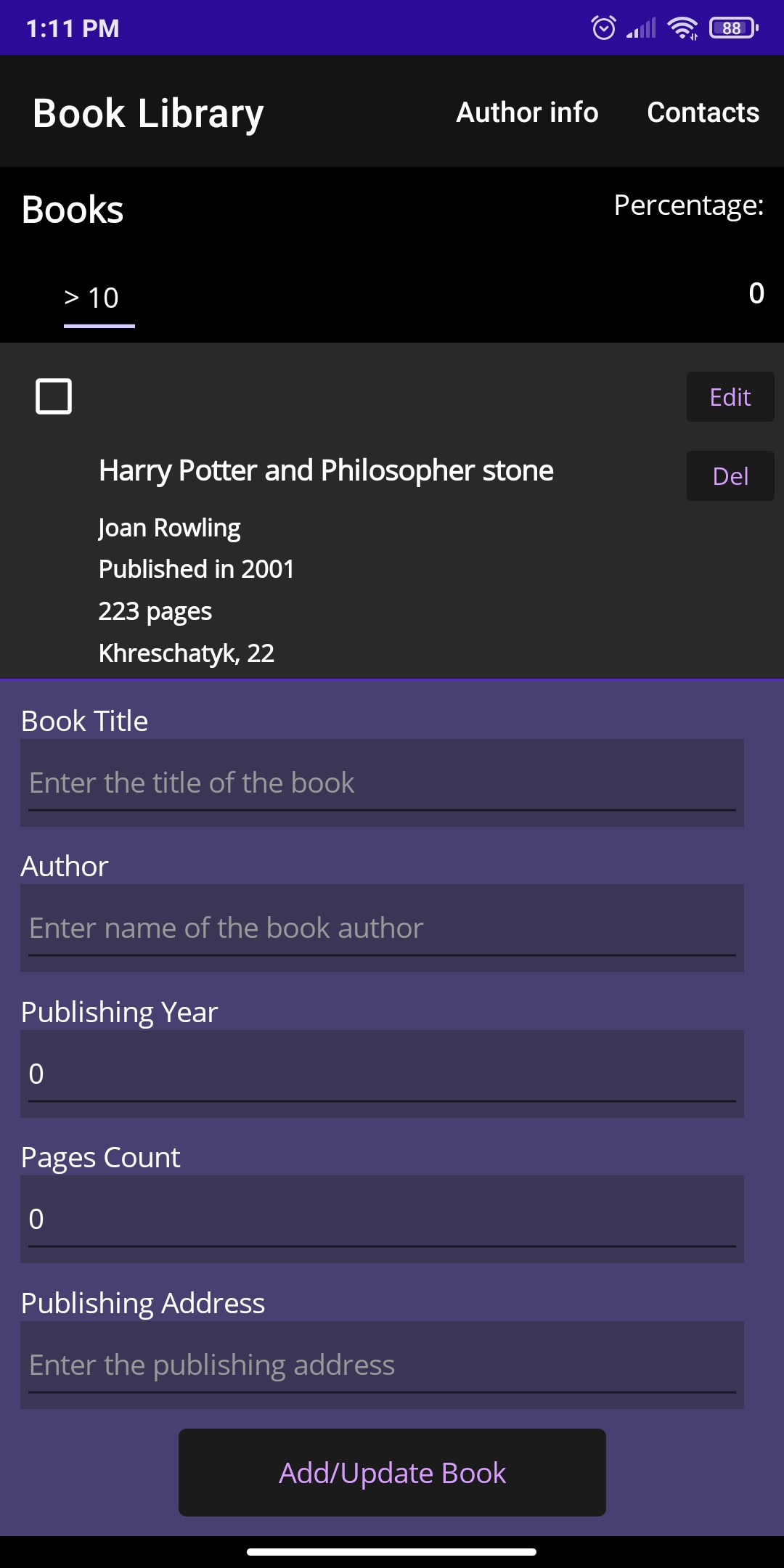


Рис.1

Також можна отримати з БД вібірку книжок, які були опубліковані більш ніж 10 років тому, для цього було створено dropdown list, де користувач може вибрати, хоче він бачити всі книжки, або лише ті, яким більше 10 років (рис. 1, 2).

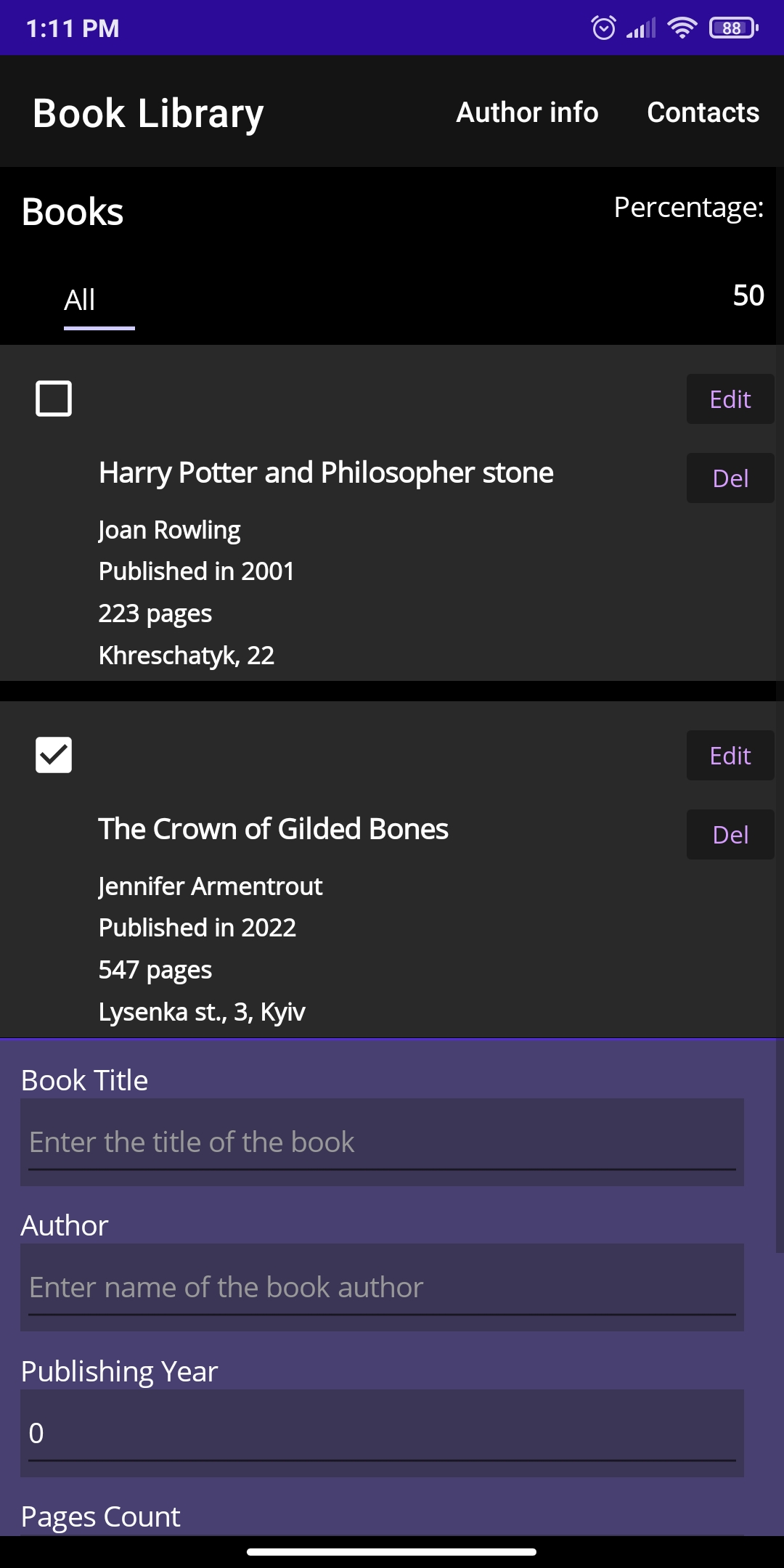


Рис. 2

Книжки можна обирати, для цього для кожної книжки збоку є checkbox. Процентна кількість вибраних книжок відображається у верхньому правому кутку (рис.1, 2).

2. Робота з адресною книгою

*Програмна логіка*

Застосунок отримує список всіх контактів за допомогою бібліотеки Microsoft.Maui.ApplicationModel.Communication. Потім, використовуючи LINQ, відфільтровує лише ті контакти, щ закінчуються на “ко” (Додаток В).

Також, для отримання доступу до контактів, в AndroidManifest.xml було помічено як true поля READ\_CONTACTS, WRITE\_CONTACTS.

*UI*

У застосунку виводяться назви та номери всіх контактів, ім’я яких закінчується на “ко” (рис. 3).

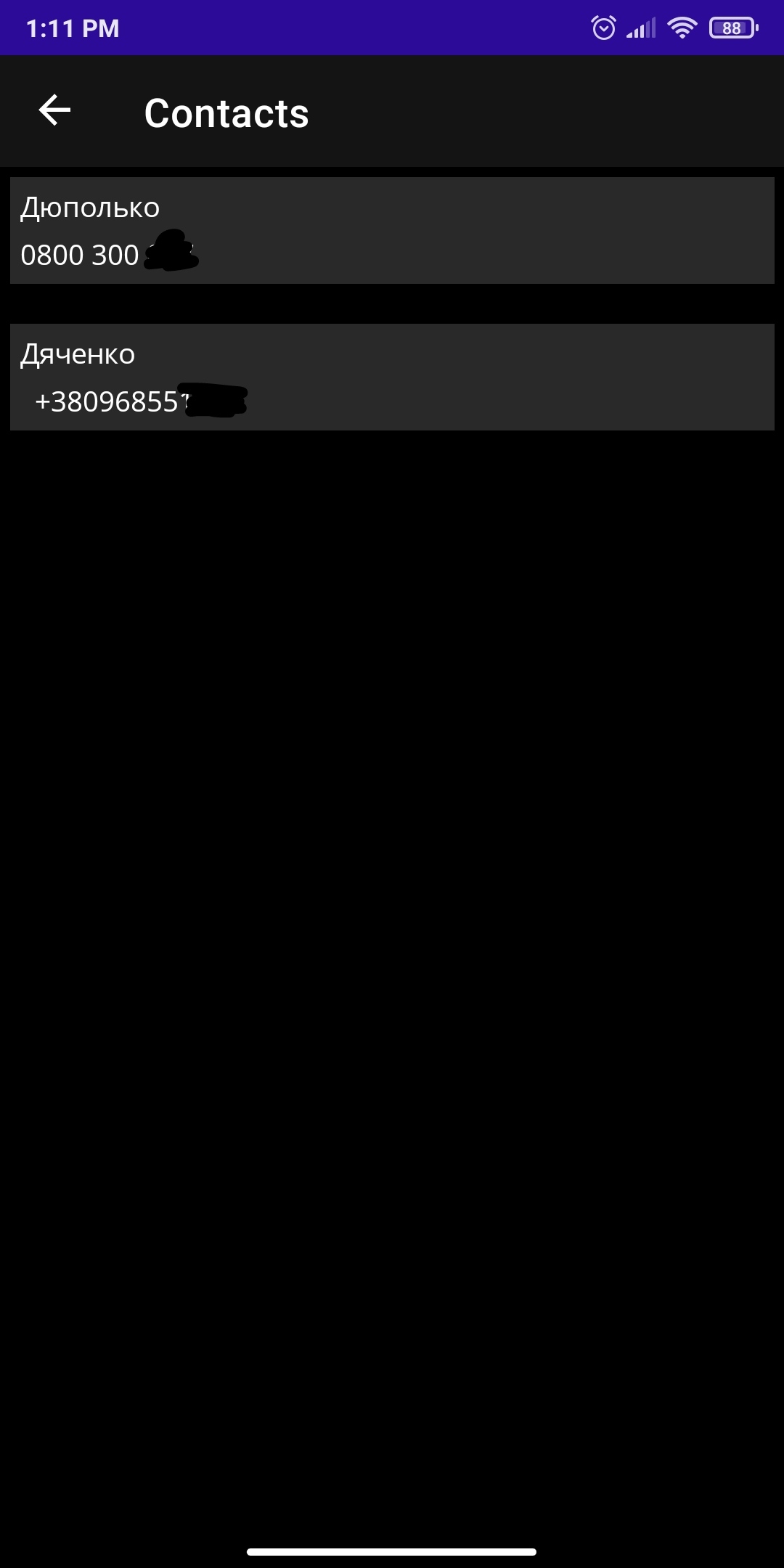


Рис.3

3. Інформація про автора

*UI*



Рис. 4

# **Додаток А**

using SQLite;

using System.Linq.Expressions;

namespace MAUISql.Data

{

public class DatabaseContext : IAsyncDisposable

{

private const string DbName = "BookLibrary.db3";

private static string DbPath => Path.Combine(FileSystem.AppDataDirectory, DbName);

private SQLiteAsyncConnection \_connection;

private SQLiteAsyncConnection Database =>

(\_connection ??= new SQLiteAsyncConnection(DbPath,

SQLiteOpenFlags.Create | SQLiteOpenFlags.ReadWrite | SQLiteOpenFlags.SharedCache));

private async Task CreateTableIfNotExists<TTable>() where TTable : class, new()

{

await Database.CreateTableAsync<TTable>();

}

private async Task<AsyncTableQuery<TTable>> GetTableAsync<TTable>() where TTable : class, new()

{

await CreateTableIfNotExists<TTable>();

return Database.Table<TTable>();

}

public async Task<IEnumerable<TTable>> GetAllAsync<TTable>() where TTable : class, new()

{

var table = await GetTableAsync<TTable>();

return await table.ToListAsync();

}

public async Task<IEnumerable<TTable>> GetFileteredAsync<TTable>(Expression<Func<TTable, bool>> predicate) where TTable : class, new()

{

var table = await GetTableAsync<TTable>();

return await table.Where(predicate).ToListAsync();

}

private async Task<TResult> Execute<TTable, TResult>(Func<Task<TResult>> action) where TTable : class, new()

{

await CreateTableIfNotExists<TTable>();

return await action();

}

public async Task<TTable> GetItemByKeyAsync<TTable>(object primaryKey) where TTable : class, new()

{ return await Execute<TTable, TTable>(async () => await Database.GetAsync<TTable>(primaryKey));

}

public async Task<bool> AddItemAsync<TTable>(TTable item) where TTable : class, new()

{

return await Execute<TTable, bool>(async () => await Database.InsertAsync(item) > 0);

}

public async Task<bool> UpdateItemAsync<TTable>(TTable item) where TTable : class, new()

{

await CreateTableIfNotExists<TTable>();

return await Database.UpdateAsync(item) > 0;

}

public async Task<bool> DeleteItemAsync<TTable>(TTable item) where TTable : class, new()

{

await CreateTableIfNotExists<TTable>();

return await Database.DeleteAsync(item) > 0;

}

public async Task<bool> DeleteItemByKeyAsync<TTable>(object primaryKey) where TTable : class, new()

{

await CreateTableIfNotExists<TTable>();

return await Database.DeleteAsync<TTable>(primaryKey) > 0;

}

public async ValueTask DisposeAsync() => await \_connection?.CloseAsync();

}

}

# Додаток В

using CommunityToolkit.Mvvm.ComponentModel;

using System.Collections.ObjectModel;

namespace MAUISql.ViewModels;

public partial class ContactsViewModel : ObservableObject

{

[ObservableProperty] private ObservableCollection<Contact> \_filteredContacts = new();

public async Task FillFilteredContacts()

{

var contacts = await Microsoft.Maui.ApplicationModel.Communication.Contacts.GetAllAsync();

var filteredContacts= contacts.Where(c => c.DisplayName.EndsWith("ко"));

foreach (var contact in filteredContacts)

{

FilteredContacts.Add(contact);

}

}

}

# 

# Додаток С

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

using MAUISql.Data;

using MAUISql.Models;

using System.Collections.ObjectModel;

namespace MAUISql.ViewModels

{

public partial class BooksViewModel : ObservableObject

{

private readonly DatabaseContext \_context;

public BooksViewModel(DatabaseContext context)

{

\_context = context;

}

[ObservableProperty] private ObservableCollection<Book> \_books = new();

[ObservableProperty] private Book \_operatingBook = new();

[ObservableProperty] private bool \_isBusy;

[ObservableProperty] private string \_busyText;

[ObservableProperty] private string \_selectedFilterOption = "All";

[ObservableProperty] private double \_pickedPercentage;

public IEnumerable<string> FilterOptions { get; } = new List<string> { "All", "> 10" };

public async Task LoadBooksAsync()

{

await ExecuteAsync(async () =>

{

var books = await \_context.GetAllAsync<Book>();

if (books is not null && books.Any())

{

Books = new ObservableCollection<Book>();

foreach (var book in books)

{

Books.Add(book);

}

}

}, "Fetching books...");

}

public async Task GetOlderThanTenYears()

{

await ExecuteAsync(async () =>

{

var books = await \_context.GetAllAsync<Book>();

books = books.Where(b => DateTime.UtcNow.Year - b.PublishingYear > 10).ToList();

if (books.Any())

{

Books = new ObservableCollection<Book>();

foreach (var book in books)

{

Books.Add(book);

}

}

}, "Fetching books...");

}

public void UpdatePercentagePicked()

{

var picked = Books.Count(b => b.IsSelected);

PickedPercentage = picked \* 100.0 / Books.Count;

}

[RelayCommand]

private void SetOperatingBook(Book? book) => OperatingBook = book ?? new();

[RelayCommand]

private async Task SaveBookAsync()

{

if (OperatingBook is null)

return;

var (isValid, errorMessage) = OperatingBook.Validate();

if (!isValid)

{

await Shell.Current.DisplayAlert("Validation Error", errorMessage, "Ok");

return;

}

var busyText = OperatingBook.Id == Guid.Empty ? "Creating book..." : "Updating book...";

await ExecuteAsync(async () =>

{

if (OperatingBook.Id == Guid.Empty)

{

// Create book

OperatingBook.Id = Guid.NewGuid();

await \_context.AddItemAsync<Book>(OperatingBook);

Books.Add(OperatingBook);

}

else

{

// Update book

if (await \_context.UpdateItemAsync<Book>(OperatingBook))

{

var bookCopy = new Book()

{

Id = OperatingBook.Id,

Name = OperatingBook.Name,

Author = OperatingBook.Author,

PagesCount = OperatingBook.PagesCount,

PublishingYear = OperatingBook.PublishingYear,

PublishingAddress = OperatingBook.PublishingAddress,

IsSelected = OperatingBook.IsSelected

};

var index = Books.IndexOf(OperatingBook);

Books.RemoveAt(index);

Books.Insert(index, bookCopy);

}

else

{

await Shell.Current.DisplayAlert("Error", "Book updation error", "Ok");

return;

}

}

SetOperatingBookCommand.Execute(new());

}, busyText);

}

[RelayCommand]

private async Task DeleteBookAsync(Guid id)

{

await ExecuteAsync(async () =>

{

if (await \_context.DeleteItemByKeyAsync<Book>(id))

{

var book = Books.FirstOrDefault(p => p.Id == id);

Books.Remove(book);

}

else

{

await Shell.Current.DisplayAlert("Delete Error", "Book was not deleted", "Ok");

}

}, "Deleting book...");

}

private async Task ExecuteAsync(Func<Task> operation, string? busyText = null)

{

IsBusy = true;

BusyText = busyText ?? "Processing...";

try

{

await operation?.Invoke();

}

catch (Exception ex)

{

}

finally

{

IsBusy = false;

BusyText = "Processing...";

}

}

}

}