НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____Сергій СТІРЕНКО

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Мобільний застосунок для моніторингу вартості товарів»

Виконав (-ла):
студент (-ка) 4 курсу, групи ІП-62
Ковальов Владислав Петрович _____

Керівник:
Доцент Габінет Артем Вікторович _____

Консультант з нормо контролю:
д.т.н, проф. Сімоненко Валерій Павлович _____

Рецензент: _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.

Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення високопродуктивних комп'ютерних систем та мереж»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____2020 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Ковальову Владиславу Петровичу

1. Тема проєкту «Мобільний застосунок для моніторингу вартості товарів», керівник проєкту доцент Габінет Артем Вікторович, затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки: аналіз існуючих програмних продуктів, вибір технологій і платформ, проектування та реалізація.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): принципова схема, схема бази даних, схема зв'язків між класами, схема прецедентів.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	06.02.2020	
2	Огляд предметної області	13.04.2020	
3	Огляд технології та платформи для розробки серверної частини	20.04.2020	
4	Огляд можливих рішень для написання клієнтської частини додатку	27.04.2020	
5	Розробка серверної частини та моделювання бази даних	06.05.2020	
6	Розробка клієнтської частини застосунку	20.05.2020	
7	Оформлення пояснювальної записки	25.05.2020	

Студент Владислав КОВАЛЬОВ

Керівник Артем ГАБІНЕТ

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з чотирьох розділів, містить 62 сторінок, 29 рисунків, 5 таблиць, 5 додатків, 13 джерел.

Метою розробки мобільного застосунку для моніторингу вартості товарів є створення додатку на смартфон, який допоможе покупцям продовольчих товарів відслідковувати наявність продуктів та їх вартість у різних мережах магазинів.

У першому розділі досліджено предметну область, а також визначено основні переваги та недоліки аналогів розроблюваного ПЗ. Крім того, висунуто вимоги до створюваного проекту.

У другому розділі описано переваги технологій, обраних для розробки, а саме Xamarin, ASP.NET Core, Entity Framework Core і SQL Server.

У третьому розділі подано архітектурні рішення щодо ПЗ, детально описано структуру БД, клієнтської та серверної частин.

У четвертому розділі подано ілюстровану інструкцію з роботі з програмнім забезпеченням.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, МОНІТОРИНГ, ВАРТІСТЬ, ПРОДОВОЛЬЧИЙ ТОВАР, C#.

ANNOTATION

Structure and scope of work. The explanatory note of the diploma project consists of four sections, contains 62 pages, 29 figures, 5 tables, 5 appendices, 13 sources.

The purpose of developing a mobile application for monitoring the value of goods is to create a smartphone application that will help grocery buyers to track the availability of products and their value in various stores.

In the first section the subject area is investigated, and advantages and disadvantages of analogues of the developed software are defined. In addition, requirements for the project are described

The second section describes the benefits of the technologies chosen for development, including Xamarin, ASP.NET Core, Entity Framework Core, and SQL Server.

The third section presents architectural solutions for software, describes the structure of the database, client and server parts.

The fourth section provides illustrated instructions for working with the software.

Keywords: MOBILE APPLICATION, MONITORING, COST, FOOD PRODUCT, C #.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467100.001 ВП	Відомість проєкту	1	
2	А4	ІАЛЦ.467100.002 ТЗ	Технічне завдання	4	
3	А4	ІАЛЦ.467100.003 ПЗ	Пояснювальна записка	62	
4	А3	ІАЛЦ.467100.004 Д1	Схема бази даних	1	
5	А3	ІАЛЦ.467100.005 Д2	Схема зв’язків між класами	1	
6	А3	ІАЛЦ.467100.006 Д3	Схема прецедентів	1	
7	А3	ІАЛЦ.467100.007 Д4	Принципова схема	1	
8	А4	ІАЛЦ.467100.008 Д5	Текст програми	35	

					ІАЛЦ.467100.001 ВП		
Зм.	*Арк.*	*ПІБ*	*Підп.*	*Дата*			
Розроб.		Ковальов В.П			Відомість дипломного проєкту	Лист	Листів
Керівн.		Габінет А.В.				1	1
Консульт.						КПІ ім.Ігоря Сікорського кафедра ОТ, гр. ІП-62	
Н/контр.		Сімоненко В.П.					
Зав. Каф.							

Технічне завдання до *дипломної* роботи

ЗМІСТ

					ІАЛЦ.467100.002 ТЗ		
Змн.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		Ковальов В.П			Мобільний застосунок для моніторингу вартості товарів. Технічне завдання	Літ.	Арк. 2 / Аркушів 4
Перевірив		Габінет А.В.					
						КПІ ім.Ігоря Сікорського кафедра ОТ, гр. ІП-62	
Н. Контр.		Сімоненко В.П.					
Зав.каф.							

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Найменування: «Мобільний застосунок для моніторингу вартості товарів». Галузь застосування – повсякденне життя, продовольчі магазини.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр програмної інженерії», затверджене кафедрою обчислювальної техніки Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення програмного забезпечення для покупців продовольчих магазинів, аби полегшити пошук товарів та порівняння цін на них у різних мережах.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є публікації в мережі Інтернет, науково-технічна література, документація програмних засобів, що використовуються при розробці.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до продукту

Користувач повинен мати змогу виконувати наступні дії:

- переглядати доступні товари та магазини;
- сканувати штрихкод товару та шукати товар за ним у магазинах;
- надсилати запит на зміну ціни товару за виявленням невідповідності даних дійсності у додатку.

Функціональні вимоги:

- зчитування штрихкоду товару за допомогою сканеру на базі камери смартфону;
- пошук за штрихкодом у базі даних інформації про товар;
- отримання списку наявних товарів та магазинів з бази даних;
- періодичне оновлення бази даних на основі наявних баз даних магазинів;
- демонстрація цін та мереж, в яких присутній товар;
- обробка запиту на зміну невірної ціни користувачем.

Вимоги до інтерфейсної складової:

- наявність кнопок для перегляду доступних магазинів та товарів;
- наявність зручного введення для пропозиції зміни ціни;
- наявність кнопки для відкриття сканеру штрихкода;
- інтуїтивна зрозумілість інтерфейсу

5.2. Вимоги до програмного забезпечення

Мінімальний набір програмного забезпечення для запуску системи:

- операційна система Windows версії 7 або вище, будь-який дистрибутив Linux (Ubuntu, Fedora, Red Hat) або MacOS
- .NET Core версії 3.1 або вище для запуску та виконання ASP.NET Core застосунків
- Мобільній пристрій з системою Android 5.0 “Lollipop” і вище

5.3. Вимоги до апаратної частини

Мінімальна необхідна кількість операційної пам’яті – 2 Гб, мінімальна кількість вільного місця на пристрої – 50 Мб, необхідне підключення до мережі Інтернет.

6. ЕТАПИ РОЗРОБКИ

	Дата
Затвердження теми роботи	06.02.2020
Огляд предметної області	13.04.2020
Огляд технології та платформи для розробки серверної частини	20.04.2020
Огляд можливих рішень для написання клієнтської частини додатку	27.04.2020
Розробка серверної частини та моделювання бази даних	06.04.2020
Розробка клієнтської частини застосунку	20.05.2020
Оформлення пояснювальної записки	25.05.2020

Пояснювальна записка
до дипломного проекту

на тему: МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ МОНІТОРИНГУ ВАРТОСТІ ТОВАРІВ

Київ – 2020 року

ЗМІСТ

СПИСОК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

Штрихко́д, штрихови́й код — спосіб запису даних, зручний для зчитування машиною.[1]

СУБД – система управління базами даних.

Xamarin – це платформа для розробки додатків із відкритим кодом від Microsoft для створення сучасних та ефективних програм для iOS та Android за допомогою C# та .NET. [3]

Asp.Net Core – це крос-платформна, високопродуктивна система з відкритим кодом для створення сучасних додатків. [4]

Entity Framework Core – це технологія для доступу до даних з відкритим кодом, легким, розширюваним і міжплатформованою версією технології Entity Framework. [5]

ВСТУП

Сьогодні кожен купує товари у продуктових магазинах. Більшість торгових точок даного типу є мережевими. На жаль, покупці часто зіштовхуються з проблемами відсутності цінників на полицях, не відповідності інформації у них дійсності, тощо. Деякі торгові марки постачаються лише в окремі магазини. Поширеною практикою є вибір торгової точки з більш дешевими товарами. Але намагання знайти найбільш дешевий товар у всіх магазинах великого міста є задачею, що потребує багато часу та фізичних зусиль. А запам’ятати, які торгові точки не продають певні товари, майже неможливо.

Нині технології невпинно йдуть уперед. Проте торгівельний бізнес не завжди встигає за ними або не має достатньо ресурсів, щоб встигнути. З іншого боку, деяким магазинам не вигідно оприлюднювати ціни на товар, адже існує шанс відтоку покупців у більш дешеві точки. Так скалося, що сучасні супермаркети й досі не мають власного сайту з товарами та цінами на них. Більш поширеною практикою є встановлення фізичних пристроїв для зчитування штрих-коду та відтворення ціни на товар на території магазину.

Перераховані проблеми можуть бути вирішені за допомогою спеціального мобільного додатку, основною функцією якого є сканування штрих-коду та демонстрація ціни на товар в магазинах, де він наявний. Завданням дипломної роботи буде розробка такого додатку.

РОЗДІЛ 1

Аналіз існуючих програмних продуктів

1.1 Загальні положення

Штриховий код - графічна інформація, яка наноситься на поверхню, маркування або упаковку виробів, що надає можливість зчитування її технічними засобами - послідовність чорних і білих смуг, або інших геометричних фігур [1].

Виділяють наступні способи кодування інформації:

- Лінійні. Це такі штрих-коди, читати які потрібно в єдиному напрямку (горизонтально). До найпоширеніших лінійних кодувань відносять:
    1. EAN (EAN-8 містить 8 цифрових символів, EAN-13 – 13)
    2. UPC (UPC-A, UPC-E)
    3. Code56
    4. Code128 (UPC / EAN-128)
- Двомірні. Такі штрихкоди виникли через потребу збільшення об’єму інформації, що підлягає кодуванню. Розшифровуються коди, як зрозуміло з їх назви, в двох вимірах (і вертикально, і горизонтально). Сьогодні існує велика кількість варіацій кодів з двома вимірами, наприклад:
    1. Data Matrix
    2. MaxiCode
    3. QR код

1.2 Змістовний опис і аналіз предметної області

Сфера торгівлі сьогодні найчастіше взаємодіє з кодами EAN та UPC. UPC кодування виникло першим. Це американська система, що складається з 12 цифрових символів. Згодом UPC стала досить популярною, тому скористатися системою захотіли представники європейських країн. На жаль, виникла проблема зайнятості діапазону кодів товарами американського виробництва. Розробники європейської системи EAN-13 мусили вирішити одразу дві проблеми. По-перше, потрібно було збільшити діапазон кодування. По-друге, існувала необхідність реєструвати товари незалежно від США, зберігаючи при цьому сумісність з UPC системою. Поява тринадцятого символу зліва від коду вирішила обидві проблеми. Нові коди, що належали до системи EAN-13, залишились сумісними з UPC, шляхом використання другої системи в якості підмножини першої з крайньою лівою цифрою 0.

Отже:

- система EAN-13 отримала незалежність від монопольного реєстратора;
- товар, виготовлений в Європі, не потребує додаткових зайвих полів чи створення іншого штрих-коду;
- не потрібно було змінювати коди товарів, виготовлених у США.

Будь-який код EAN-13 складається з п’яти секцій:

- Префікс, що позначає національну організацію GS1 (3 символи);
- Реєстраційний номер виробника (від 4 до 6 символів);
- Власне код товару (від 3 до 5 символів);
- Контрольна цифра (1 символ);
- Додаткова секція (є необов’язковою, іноді заповнюється знаком «>» або показником вільної секції).

Національна організація, яку позначає GS1 – це міжнародна організація, що є відповідальною за кодування одиниць логістики та ведення обліку за стандартами.

Реєстраційний номер виробника – це код, що позначає підприємство, що випускає або продає товар. Оскільки код містить від 4 до 6 символів, на рівні регіону є можливість зареєструвати 10000-1000000 підприємств.

Власне на товар виділяють всього від 3 до 5 символів. На довжину зони впливає безпосередньо реєстратор, обираючи базову довжину номеру виробника. Кількість найменувань, що можна зареєструвати, зважаючи на довжину коду, – 1000-100000 одиниць.

Останній символ коду є контрольним, він потрібен, аби перевірити, чи правильно код зчитав сканер.

1.3 Опис існуючих рішень

Найбільш схожим застосунком за порівняльним функціоналом додатку є веб сервіс hotline[2], котрий, проте, має достатньо відмінностей. Наприклад, hotline для пошуку використовує назву товару, а не штрих-код, та моніторить виключно непродовольчу продукцію. У майбутньому функціонал розробленого додатку можна буде розширити, зважаючи на зручність пошуку за ім’ям, а також підтягнути до бази даних інформацію про непродовольчі товари. У той же час основні недоліки hotline, такі як відсутність пошуку за штрих-кодом, мобільної версії та моніторингу харчової продукції мережевих магазинів, вирішені у власному додатку.

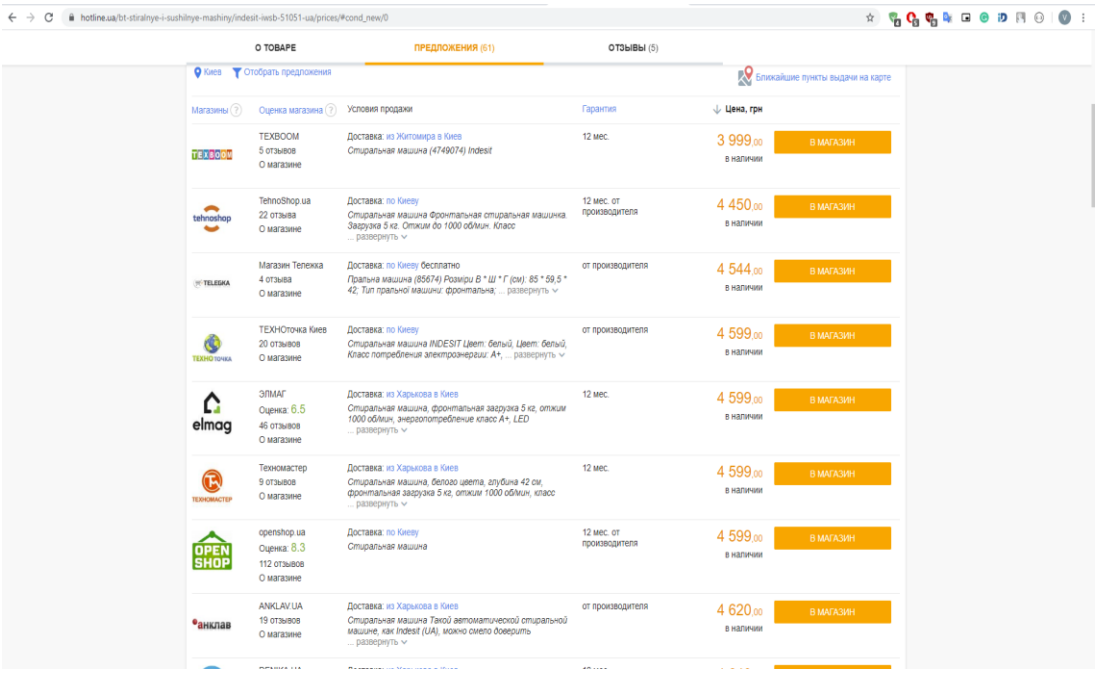


Рис. 1.1 – Моніторинг товарів за допомогою веб сервісу hotline

Аналоги за скануючим функціоналом можна розділити на дві категорії:

- Сканер з фізичною оболонкою
- Сканер в електронному девайсі

1.3.1 Сканер з фізичною оболонкою

Зазвичай фізичні сканери схожі за виглядом та функціоналом. Їх основний недолік – невелика кількість у порівнянні з кількістю покупців, а це означає, що їх потрібно шукати у магазині та витрачати на це час. Також фізичні сканери орієнтовані на продукцію однієї торгової точки.

Перераховані вище недоліки вирішуються за допомогою мобільного додатку, адже він завжди під рукою у смартфоні та не прев’язаний до конкретного магазину.

Рис. 1.2 – Фізичний сканер магазину АШАН

1.3.2 Сканер в електронному девайсі

Існує не так багато мобільних додатків, що мають функцію сканування штрих-коду. Додатки окремих мереж магазинів обмежуються моніторингом знижок або виступають у ролі мобільної картки лояльності.

Проте навіть за наявності сканеру в мобільному додатку, моніторинг проводиться в одній мережі магазинів. Аби володіти інформацією про декілька мереж, користувачу доведеться завантажити та встановити декілька додатків. Це потребує додаткових ресурсів: часу на пошук, пам’яті телефону. Розроблений мною додаток вирішує цю проблему.

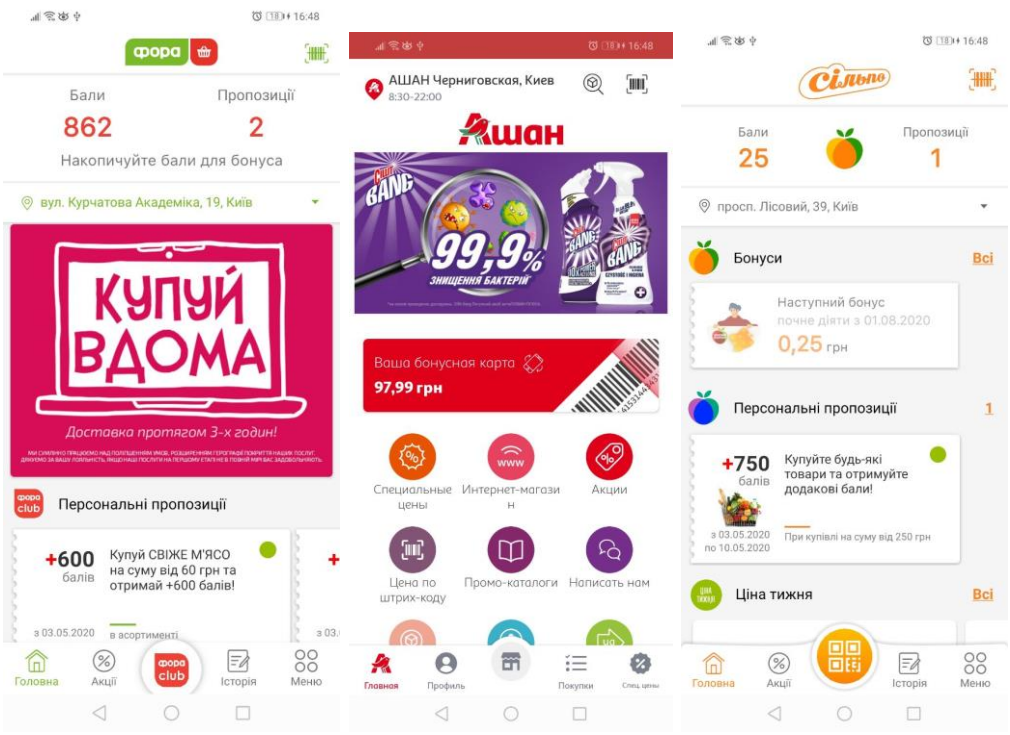


Рис. 1.3 – Мобільні додатки зі сканером товарів магазинів Фора, АШАН, Сільпо

Рис. 1.4 – Мобільний додаток без сканеру товарів магазину АТБ

1.4 Аналіз вимог до продукту

Кінцевим користувачем розроблюваного продукту є власник смартфону або планшету, на який забезпечення встановлено. Користувач має змогу виконувати наступні дії:

- переглядати доступні товари та магазини;
- сканувати штрихкод товару та шукати товар за ним у магазинах;
- надсилати запит на зміну ціни товару за виявленням невідповідності даних дійсності у додатку.

Базуючись на зазначених вище можливостях користувача, розроблюваний продукт має відповідати ряду функціональних вимог та вимог до інтерфейсної складової.

Функціональні вимоги до створюваного програмного забезпечення:

- зчитування штрихкоду товару за допомогою сканеру на базі камери смартфону;
- пошук за штрихкодом у базі даних інформації про товар;
- отримання списку наявних товарів та магазинів з бази даних;
- періодичне оновлення бази даних на основі наявних баз даних магазинів;
- демонстрація цін та мереж, в яких присутній товар;
- обробка запиту на зміну невірної ціни користувачем.

Вимоги щодо інтерфейсної складової програмного забезпечення:

- наявність кнопок для перегляду доступних магазинів та товарів;
- наявність зручного введення для пропозиції зміни ціни;
- наявність кнопки для відкриття сканеру штрихкода;
- інтуєтивна зрозумілість інтерфейсу.

Висновки до розділу 1

Виходячи з усього вищенаведеного, можна дійти висновку, що предметна область, що розглядається в рамках даної роботи, на сьогодні є досить затребуваною.

Оскільки діджиталізація в Україні та світі стає необхідною у більшості сфер життя, перехід до моніторингу продовольчих товарів з телефону є надзвичайно актуальним. Хоча сьогодні вже існують програмні рішення, які в тій чи іншій мірі вирішують поставлену проблему, вони мають недоліки, що вирішуються власним продуктом.

РОЗДІЛ 2

Вибір технологій і платформ

2.1 Xamarin

Xamarin - це фреймворк для кросплатформленної розробки мобільних додатків (iOS, Android, Windows Phone) з використанням мови C#[3]. Цей фреймворк дозволяє користувачу отримати доступ до функціоналу платформи SDK, а також нативних можливостей роботи з UI. Як результат, створений застосунок має ті ж переваги, що нативні додатки, до того ж є досить продуктивним.

У наш час популярності набули фреймворки, що підтримують HTML5 та JavaScript в якості засобів створення кросплатформенних застосунків на мобільні телефони. Програма додатку за способом розробки не відрізняється від базового мобільного сайту. Під час її створення використовуються бібліотеки JavaScript, такі як Jquery Mobile. Розроблена програма загортається в контейнер, що імітує нативний застосунок. Такі фреймворки мають очевидні недоліки. Так, користувач не може користуватися нативними засобами інтерфейсу. Навіть для використання кнопки «Повернутися» на iPhone, потрібно самотужки створити арт та налаштувати верстку. Інший недолік – урізання API, що його користувач застосовує для взаємодії з платформою. Так, до окремих функцій різних платформ доступу немає. Крім того, застосунок на базі подібних фреймворків використовує телефонний браузер для запуску. Отже, падає продуктивність і з’являються проблеми з відтворенням.

Xamarin базується на open-source версії .NET платформи, що називається Mono. Дана реалізація складається з компілятора мови C#, основних бібліотек цієї ж мови та середовища виконання. Проект створений, аби користувачі мали змогу використовувати будь-які платформи, не лише Windows, для запуску програм, написаних на C#.

iOS та Android відрізняються щодо виконання застосунків за способом їх прекомпіляції. Аби запустити програму, Android застосовує віртуальну Java-машину, що називається Dalvik. Нативні застосунки, написані на мові Java, мають скомпілюватись у байт-код, а вже він інтерпретується в процесорні команди Dalvik`ом саме під час виконання програми. Така компіляція має назву Just-in-time, що в перекладі означає компіляцію на льоту. iOS використовує відмінний спосіб компіляції – Ahead-of-Time, тобто модель компіляції до виконання. Xamarin обробляє такі відмінності, шляхом надання свого компілятора кожній з платформ. Таким чином розроблювані застосунки є нативними, незалежними від контексту браузера, а також використовують увесь функціонал платформи.

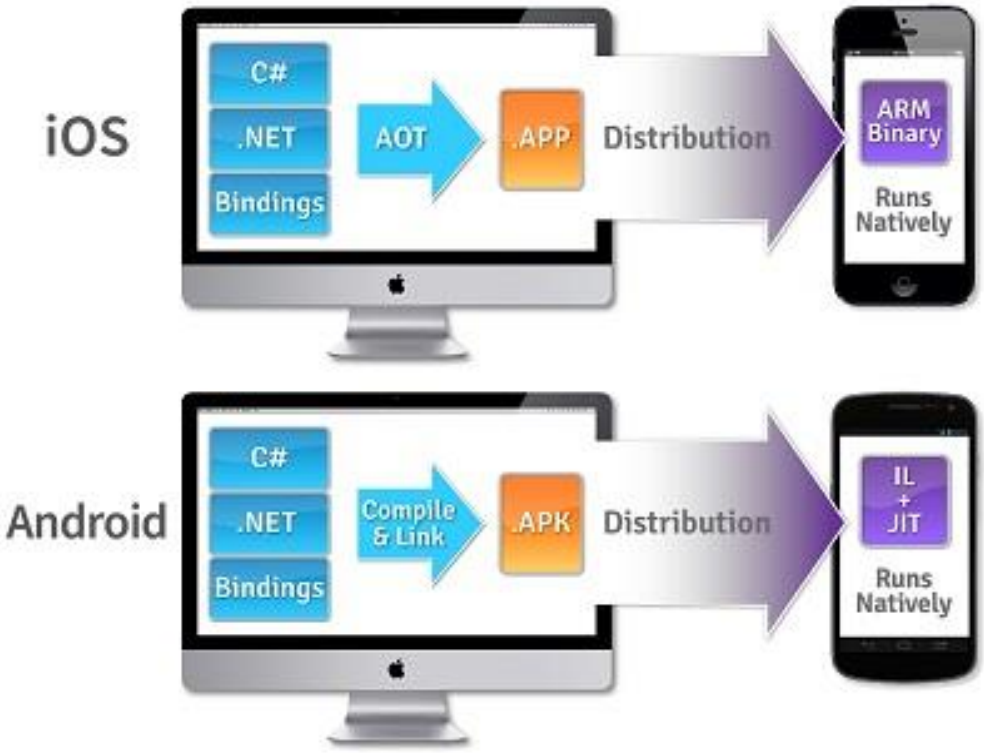


Рис. 2.1 – Способи попередньої компіляції IOS та Android

iOS компіляція є досить простою, адже потреби у віртуальній машині немає. Код програми компілюється в машинний попередньо за допомогою AOT компілятора Mono.

Ситуація з платформою Android є складнішою. Під час компіляції код, написаний на C #, перетворюється у байт-код. З байт-кодом працює віртуальна машина Mono, що також присутня у збірці додатку. Mono та Dalvik розроблені на мові C та працюють поверх ядра Linux. Якщо запустити застосунок на Android, дві віртуальні машини розпочнуть роботу паралельно, здійснюючи обмін даними за допомогою механізму wrapper`ів.

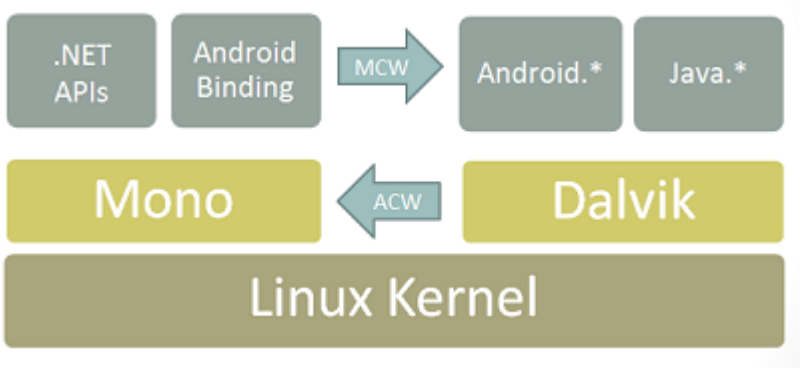


Рис. 2.2 – Компіляція на Android

Xamarin підтримує асинхронне програмування. Користувачі мають доступ до класів простору імен System.Threading.ThreadPool, а також System.Threading.Thread. Крім того, є можливість користуватися функціоналом Task Parallel Library.

На iOS Xamarin має трохи обмежені можливості. Це пов’язано з тим, що компіляція на платформі iOS відбувається без віртуальної машини, отже підтримка Generic є проблематичною. Аналізувати код та визначати допустимі конкретизації в різних класах або методах доводиться безпосередньо компілятору. Тому обмеження включають наступні:

- Бажано не використовувати методи, що є Virtual та Generic, адже компілятор може пропустити деякі варіанти використання;
- Заборонено створювати Generic нащадків базового класу Objective-C NSObject, що може негативно вплинути на архітектуру програми.

Xamarin дозволяє користуватися нативними засобами створення користувацького інтерфейсу усіх платформ. Якщо відбувається розробка на Android, працювати над інтерфейсом можна з просто з коду, або ж декларативним методом, описуючи його в XML. Під iOS також є можливість розробки користувацького інтерфейсу з коду. Також існують нативні інструменти роботи з UI – Storyboard або набір xib-файлів.

Користувацький інтерфейс має бути розроблений окремо під кожну платформу. Тобто, якщо розробляти застосунок під iOS та Android, код, що відтворює зовнішній вигляд додатку, доведеться писати двічі.

Типовий додаток, створений за допомогою Xamarin складається з наступних шарів:

- Data Layer (DL) - сховище даних, яким може виступати, база даних SqlLite або набір xml-файлів;
- Data Access Layer (DAL) - обгортка над сховищем, що необхідна для виконання CRUD-операцій;
- Business Layer (BL) - шар, в якому прописана бізнес-логіка застосунку;
- Service Access Layer (SAL) - шар, що відповідає за роботу з віддаленими сервісами (Rest, Json, WCF);
- Application Layer (AL) - шар, що містить код, який напряму залежить від платформи, тобто залежний від monodroid.dll та monotouch.dll бібліотек;
- User Interface Layer (UI) - шар, що містить логіку користувацького інтерфейсу.

Рис. 2.3 – Архітектура додатку

Кросплатформенними є всі шари, розташовані вище Application Layer. Частка переносимого коду досить сильно залежить від самого додатку, але вона не перевищує 50-60%. Інженери Xamarin це розуміють, тому прагнуть до збільшення цієї частки. У якості досягнень у вирішенні цієї проблеми можна розглядати бібліотеку Xamarin.Mobile. Вона надає єдиний для різних платформ API для роботи з камерою, контактами та геолокацією. Але використання цієї бібліотеки ніяк не обмежує вас в застосуванні платформо-залежного API, наприклад, за допомогою механізму делегатів.

На поточний момент технологія Xamarin є серйозним інструментом для вирішення складних завдань в області розробки мобільних додатків. Незважаючи на це, команда розробників не зупиняється і продовжує його активний розвиток та поліпшення. На мій погляд у технології велике майбутнє і з кожним днем число розробників, які використовують її в якості основного фреймворка для розробки буде неухильно зростати.

2.2 ASP.NET Core

ASP.NET Core – це технологія, представлена Microsoft. Платформа призначена для розробки веб-застосунків, наприклад, маленьких веб-сайтів або великих веб-порталів і навіть веб-сервісів[4].

Хоча ASP.NET Core вважається своєрідним вдосконаленням платформи ASP.NET, її було б хибно вважати просто новим релізом. Появу ASP.NET Core можна без перебільшень вважати революційним явищем, що якісно змінює існуючу платформу.

Створення ASP.NET Core розпочалося в 2014 році. У той час її називали ASP.NET vNext. Перший реліз з'явився у 2016 році влітку. Пізніше, аж у 2019 році було випущено ASP.NET Core 3.1.

Сьогодні ASP.NET Core є цілком open-source. Усі файли фреймворку будь-хто може знайти на GitHub.

ASP.NET Core працює над середовищем .NET Core, що є крос-платформенним та доступним до розгортання на таких ОС, як Linux, Windows чи Mac OS. Отже, фреймворк дозволяє розробку під різні платформи. Не зважаючи на те, що більшість розробників досі надають перевагу Windows, обмежень щодо вибору лише цієї операційної системи більше не існує. У той же час, традиційний IIS та Kestrel (мультиплатформенний веб-сервер) підходять для створення веб-додатків.

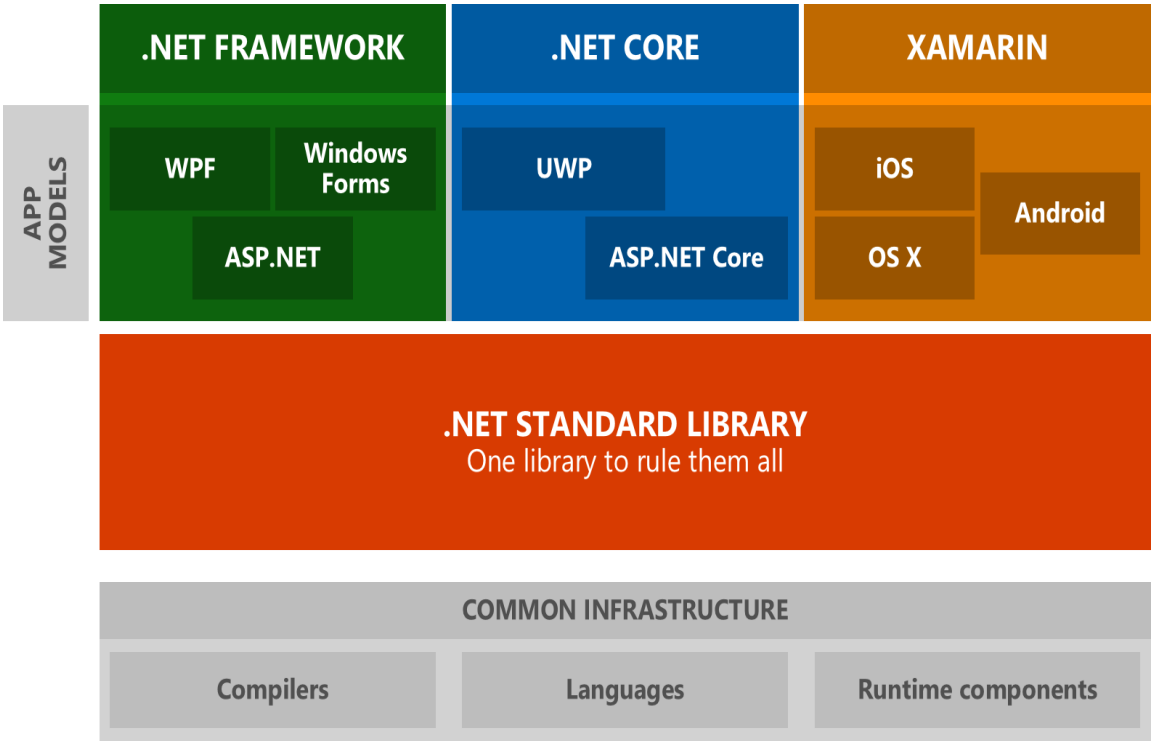


Рис. 2.4 – Екосистема .NET

Визначною перевагою фреймфорку є модульність. Ця особливість дозволяє підвантажити потрібні для розробки веб-застосунку пакети окремо, застосовуючи менеджер пакетів Nuget. Ще однією перевагою є відсутність необхідності користуватися System.Web.dll бібліотекою, що була обов'язковою у ранніх версіях платформи.

ASP.NET Core містить фреймворк MVC, який включає функціонал архітектури MVC, а також Web Pages та Web API. Раніше платформа підтримувала окрему реалізацію кожного з них, що спричиняло виникнення дублювань. Нині цю проблему вирішено і однаковий функціонал об'єднано в єдиний модуль ASP.NET Core MVC.

Створення такої моделі MVC – не єдине покращення функціоналу платформи. До плюсів фреймворку можна віднести появу тег-хелперів, що сприяють органічній взаємодії C# коду та html синтаксису.

Серед характеристик ASP.NET Core можна виділити розширюваність. Усі компоненти фреймворку є відносно незалежними. Отже, розробник може скористуватися наявною їх реалізацією, змінити її шляхом розширення та успадкування, чи розробити власний функціонал з нуля.

Фркеймворк має спрощену систему впровадження залежностей та налаштування конфігурацій. З’явився легковісий контейнер, відповідальний за управління залежностями. Розробнику більше немає потреби додавати в проект сторонні модулі, що раніше за це відповідали (наприклад, Ninject). Але можливісь використання додаткових пакетів також залишилась.

Розробляти проект можна за допомогою нових версій Visual Studio (2015 року і пізніших). До інструментаріїв для мультиплатворменної розробки належить Visual Studio Code.

Отже, основні відмінності ASP.NET Core від ASP.NET включають:

- Наявність модульного конвеєра HTTP-запитів;
- Можливість розгортати застосунок у два способи: як власний процес або на IIS;
- Застосування платформи .NET Core та її функціоналу;
- Завантаження пакетів за допомогою NuGet;
- Єдина модель веб-розробки, що включає як Web UI, так і Web API;
- Вбудовані засоби керування впровадженням залежностей;
- Розширюваність;
- Мультиплатформеність, тобто можливість створення та впровадження застосунків на Windows, Mac і Linux;
- Перехід на open source, можливість покращень.

Роутинг

У ASP.NET Core MVC активно використовується ASP.NET Core роутинг, потужний компонент URL-маппінга, що дозволяє створювати додатки зі зрозумілими і чіткими URL. Користувач визначає патерни іменування URL, що підходять для SEO і генерування посилань, не беручи до уваги те, як організовані файли на сервері. Можна визначати роутинг, використовуючи зручний шаблонний синтаксис, при якому підтримуються обмеження, значення за замовчуванням і додаткові значення.

Роутинг, заснований на угоді, дозволяє глобально визначити ті формати URL, які приймає додаток, і визначає, як ці формати узгоджуються з конкретними методами контролера. Після отримання вхідного запиту механізм роутинга парсить URL, пов'язує його з одним з певних URL форматів, а потім викликає відповідний метод контролера.

```
routes.MapRoute(name: "Default", template: "{controller=Home}/{action=Index}/{id?}");
```

Рис. 2.5 – Шаблону роутингу

Атрибутивний роутинг дозволяє вказати роутовую інформацію, передавши контролерам і методам дії атрибути, які визначають роути додатку. Це означає, що визначення розташовуються поруч з контролером і діями, з якими вони пов'язані.

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
  [HttpGet("{id}")]
  public IActionResult GetProduct(int id)
  {
    ...
  }
}
```

Рис. 2.6 – Зразок атрибутного роутингу

ASP.NET Core MVC конвертує дані клієнтського запиту (значення з форм, роутовані дані, параметри рядка запиту, HTTP заголовки) в об'єкти, які може обробити контролер. В результаті цього контролер не повинен з'ясовувати, що за вхідні дані йому прийшли; дані просто передаються його методам у вигляді параметрів.

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null) { ... }
```

Рис. 2.7 – Приклад передачі параметрів в контролер

ASP.NET Core MVC конвертує дані клієнтського запиту Web API та представляє спосіб побудови програми ASP.NET, який спеціально заточений для роботи в стилі REST (Representation State Transfer або "передача стану відтворення"). REST-архітектура передбачає застосування таких методів або типів запитів HTTP для взаємодії з сервером: GET, POST, PUT, DELETE.

Найчастіше REST-стиль особливо зручний при створенні будь-якого роду Single Page Application, які нерідко використовують спеціальні javascript-фреймворки типу Angular, React або Vue.js. По суті Web API представляє собою веб-службу, до якої можуть звертатися інші додатки. Причому ці програми можуть представляти будь-яку технологію і платформу: веб-додатки, мобільні або десктопні клієнти.

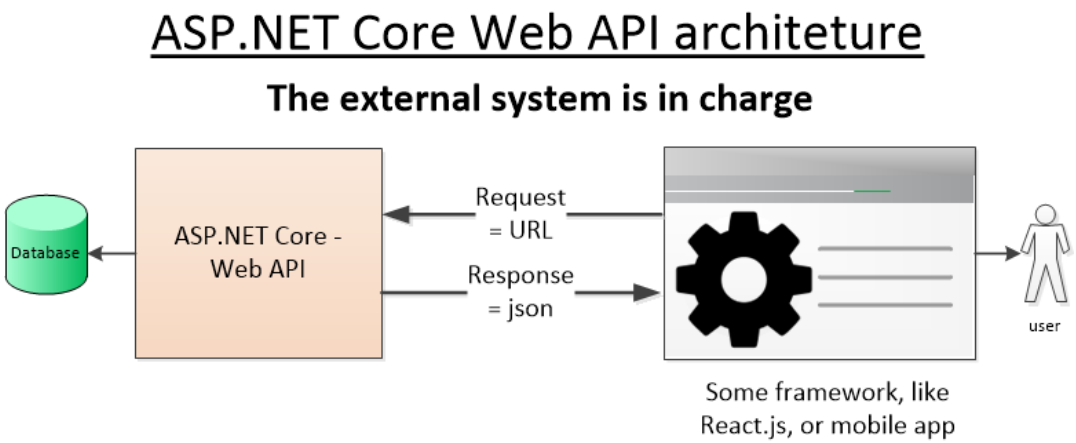


Рис. 2.8 – Архітектура ASP.NET Core Web API

2.3 Entity Framework Core

Entity Framework Core (EF Core) – це технологія Microsoft, що характеризується об’єктно-орієнтованістю, розширюваністю, легковісістю та використовується для роботи з даними. EF Core є ORM-інструментом (object-relational mapping - відображення даних на реальні об'єкти)[5]. Отже, EF Core можна використовувати в роботі з базами даних, але збільшивши рівень абстракції. Так, фреймворк дозволяє абстрагуватися від БД та окремих таблиць, обробляючи дані незалежно від особливостей сховища. Фізичний рівень передбачає роботу з таблицями, індексами та ключами. Фреймворк же пропонує концептуальний рівень, тобто зміну об’єктів.

Entity Framework Core дозволяє працювати з великою кількістю СУБД, якщо для них існують необхідні провайдери.

Microsoft пропонує вбудовані провайдери для таких СУБД: SQLite, MS SQL Server, PostgreSQL. Робота з MySQL відбувається за допомогою стороннього постачальника.

Перевагою EF Core є і те, що її API для управління даними є універсальним. Отже, зміна СУБД вплине на проект незначною мірою. Так, доведеться переналаштувати конфігурацію та підключення до тих чи інших провайдерів. У той же час класи, що змінюють дані, дадають та видаляють їх з бази даних, змін не потребують.

Хоча Entity Framework Core перейняв більшість функціоналу від попередніх версій (наприклад, Entity Framework 6), фреймворк не можна вважати усього лише черговою версією. Навпаки, це – нова технологія з використанням деяких базових принципів роботи. Таким чином, EF Core має власну систему версій.

Як засіб роботи з даними фреймворк може бути застосований на варіативних платформах .NET, включаючи ASP.NET Core, Windows Forms, UWP, WPF. Ще однією перевагою фреймворку є його крос-платформенність, тобто можливість використання його як на Windows, так і на Mac OS або Linux.

Основним поняттям Entity Framework є entity, тобто сутність у перекладі. Сутність складає множина даних, що стосуються конкретного об’єкта. Отже, фреймворк призначений для роботи з об’єктами та їх наборами, а не з таблицями.

Особливістю Entity Framework Core, як ORM технології, є застосування LINQ запитів для отримання наборів даних з бази. Так, LINQ допомагає створювати запити на вибірку об'єктів, а Entity Framework при виконанні запиту перетворює їх у вирази, які може зрозуміти конкретна система управління базами даних (найчастіше це SQL вирази).

2.4 SQL Server

SQL Server є однією з найбільш популярних систем управління базами даних у світі[13]. СУБД використовують для розробки великої кількості проектів, включаючи як прості додатки, так і великі системи.

SQL Server розробила Microsoft. Перший реліз відбувся у 1987 році.

SQL Server довго залишався системою управління базами даних виключно для Windows, але версія 16 виправила даний недолік, зробивши систему доступною для Linux.

Характерні особливості SQL Server включають:

- Висока продуктивність. SQL Server відрізняється швидкістю;
- Надійність. SQL Server шифрує дані;
- Легкість використання. Дана СУБД є досить простою для роботи та адміністрування у порівнянні з іншими;

Головним об’єктом в MS SQL Server, як і в будь-якій іншій СУБД, є база даних. БД – це сховище даних, що є організованими у деякий спосіб. Фізично БД найчастіше є файлом, що зберігається на жорсткому диску. Система управління (management system) потрібна, аби адмініструвати базу даних.

Для організації баз даних MS SQL Server використовує реляційну модель. Ця модель була розроблена в 1970 році Едгаром Коддом. А на сьогодні вона фактично є стандартом для організації баз даних.

Реляційна модель передбачає зберігання даних у вигляді таблиць, кожна з яких складається з рядків і стовпців. Кожен рядок зберігає окремий об'єкт, а в стовпчиках розміщуються атрибути цього об'єкта.

Аби ідентифікувати рядок у таблиці, використовується primary key (первинний ключ). Первинний ключ може складатися з одного або декількох стовпців. Посилання на конкретний рядок в таблиці стає можливим через використання первинного ключа. Отже, для кожного рядка існує свій первинний ключ.

Ключі також необхідні для зв’язку однієї таблиці з іншою, тобто для встановлення зв’язків у БД.

Structured Query Language (SQL мова) використовується для роботи з БД. Клієнт, яким може виступати деяка зовнішня програма, створює та надсилає запит мовою SQL, використавши необхідне API. СУБД обробляє та перетворює запит, надсилаючи назад результат його виконання.

Спершу SQL розроблялась IBM System / R СУБД. Мова мала назву SEQUEL (Structured English Query Language). Зрештою, ні система управління, ні мова, опублікованими не були.

Перша СУБД була створена Relational Software Inc. у 1979 році. Вона мала назву Oracle та використовувала мову SQL. Пізніше проект став настільки успішним, що компанія змінила назву на Oracle.

Пізніше з’явилися інші системи БД, що застосовували SQL. Як результат, Американський Національний Інститут Стандартів (скорочено ANSI) кодифікував мову у 1989 році. Тоді ж було вперше опубліковано її стандарт. Звісно, згодом стандарт іноді змінювався і доповнювався. Останнє його оновлення відбулося в 2011 році. Але, не зважаючи на наявність стандарту, нерідко виробники СУБД використовують свої власні реалізації мови SQL, які трохи відрізняються одна від одної.

Виділяють два різновиди мови SQL: PL-SQL і T-SQL. PL-SQL використовується в таких СУБД як Oracle і MySQL. T-SQL (Transact-SQL) застосовується в SQL Server.

Висновки до розділу 2

В даному розділі наведено перелік основних технологій, застосованих під час розробки програмного забезпечення. Також описано переваги Xamarin, ASP.NET Core, Entity Framework Core і SQL Server та обґрунтовано їх вибір для створення програми.

РОЗДІЛ 3

Проектування та реалізація

Архітектура даного мобільного застосунку складається з трьох частин: бази даних, серверу, що отримує дані з бази даних, та мобільного додатку, який взаємодіє з сервером.

3.1 База даних

Короткий опис структури бази даних наведено в таблицях.

Таблиця 3.1 Структура бази даних

Назва таблиці	**Опис**
Companies	Список компаній
Stores	Список магазинів компаній
Prices	Список цін на проекти
Goods	Список продуктів

Таблиця 3.2 Опис таблиці Companies

Назва поля	**Тип у базі даних**	**Опис**
RetailName	int	Первинний ключ – унікальний ідентифікатор запису в таблиці
ImageUrl	nvarchar	Посилання на картинку, що є логотипом компанії

Таблиця 3.3 Опис таблиці Stores

Назва поля	**Тип у базі даних**	**Опис**
Id	int	Первинний ключ
Name	navarchar	Назва магазину
RetailName	int	Зовнішній ключ на таблицю Companies
Address_City	navarchar	Місто розташування магазину
Address_Street	navarchar	Вулиця розташування магазину
Address_Building	navarchar	Номер будинку розташування магазину
Coords	navarchar	Координати магазину
LastUpdate	datetime2	Остання дата оновлення магазину

Таблиця 3.4 Опис таблиці Prices

Назва поля	**Тип у базі даних**	**Опис**
StoreId	int	Зовнішній ключ на таблицю Stores та частина первинного ключу
Price	decimal	Ціна продукту
GoodBarcode	nvarchar	Зовнішній ключ на таблицю Goods (містить штрих-код продукту) та частина первинного ключу

Таблиця 3.5 Опис таблиці Goods

Назва поля	**Тип у базі даних**	**Опис**
Barcode	nvarchar	Первинний ключ, що містить штрихкод
Name	nvarchar	Назва продукту
ImageUrl	nvarchar	Посилання на зображення продукту

3.2 Серверна частина

У будь-якому типі проектів ASP.NET Core, як і в проекті консольного застосунку, присутній файл Program.cs, в якому визначено однойменний клас Program і з якого починається виконання програми.

Щоб запустити додаток ASP.NET Core, необхідний об'єкт IHost, в рамках якого розгортається веб-додаток. Для створення IHost застосовується об'єкт IHostBuilder.

Метод ConfigureWebHostDefaults() в якості параметра приймає делегат Action<IWebHostBuilder>. А за допомогою послідовного виклику ланцюжка методів у об'єкта IWebHostBuilder проводиться ініціалізація веб-сервера для розгортання веб-додатка. Зокрема, в даному випадку у IWebHostBuilder викликається метод UseStartup().

Метод ConfigureLogging() використовується для налаштувань логування (вибирається мінімальний рівень логування). Метод UseNLog() підключає бібліотеку NLog до об’єкту IHost.

```
public class Program
{
    0 references | vladkovaliov, 11 days ago | 2 authors, 2 changes
    public static void Main(string[] args)
    {
        var logger = NLogBuilder.ConfigureNLog("nlog.config").GetCurrentClassLogger();
        try
        {
            logger.Debug("App started.");
            CreateHostBuilder(args)
                .Build()
                .Run();
        }
        catch (Exception ex)
        {
            logger.Error(ex, "Stopped App because of unhandled exception");
            throw;
        }
        finally
        {
            NLog.LogManager.Shutdown();
        }
    }

    1 reference | vladkovaliov, 11 days ago | 2 authors, 2 changes
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseContentRoot(Directory.GetCurrentDirectory())
                          .UseIISIntegration()
                          .UseStartup<Startup>();
            })
            .ConfigureLogging(logging =>
            {
                logging.ClearProviders();
                logging.SetMinimumLevel(LogLevel.Information);
            })
            .UseNLog();
}
```

Рис. 3.1 – Клас Program

Клас Startup є вхідною точкою в додаток ASP.NET Core. Цей клас відповідає за конфігурацію додатку, налаштовує сервіси, які додаток буде використовувати, встановлює компоненти для обробки запиту або middleware.

3.2.1 Доступ до бази даних

Для доступу до бази використовується технологія EF Core.

У будь-якому додатку, що працює з БД через Entity Framework, необхідно мати контекст (клас, похідний від DbContext). В даному випадку таким контекстом є клас AppDbContext.

Також в класі визначено декілька властивостей: Goods, Prices, Stores, Companies, які будуть зберігати набір об'єктів Good, Price, Store, Company відповідно. У класі контексту даних набір об'єктів представляє клас DbSet <T>. Через цю властивість буде здійснюватися зв'язок з таблицею об'єктів в базі.

Варто відзначити, що за замовчуванням у проекті немає бази даних. Тому в конструкторі класу контексту визначений виклик методу Database.EnsureCreated(), який при створенні контексту автоматично перевірить наявність бази даних і, якщо вона відсутня, створить її.

Крім того, для налаштувань підключення потрібно передати параметр DbContextOptions в базовий клас.

Всі сутності, з якими працює Entity Framework, визначаються у вигляді класів моделей. При цьому EF Core використовує ряд умовностей для зіставлення класів моделей з таблицями. Наприклад, назви стовпців повинні відповідати назвам властивостей і т.п. В цьому випадку Entity Framework зможе зіставити стовпці таблиці і властивості моделі.

Однак, за допомогою таких механізмів, як Fluent API і анотації даних ми можемо додати додаткові правила конфігурації або перевизначити використані умовності.

Fluent API представляє набір методів, які визначають зіставлення між класами і їх властивостями і таблицями та їх стовпцями. Як правило, функціонал Fluent API задіюється при перевизначенні методу OnModelCreating.

Для налаштування відносин між моделями за допомогою Fluent API застосовуються спеціальні методи: HasOne, HasMany, WithOne, WithMany. Методи HasOne і HasMany встановлюють навігаційну властивість для сутності, для якої проводиться конфігурація. Далі можуть слідувати виклики методів WithOne і WithMany, що ідентифікують навігаційну властивість на стороні пов'язаної сутності. Методи HasOne / WithOne застосовуються для звичайної властивості навігації, що представляє одиночний об'єкт, а методи HasMany /

WithMany використовуються для навігаційних властивостей, що представляють колекції. Зовнішній ключ встановлюється за допомогою методу HasForeignKey.

Як правило, кожен тип зіставляється з окремою таблицею. Однак бувають випадки, коли певний клас існує не сам по собі, а несе деяку додаткову інформацію по відношенню до іншої головної моделі. І в версії 2.0 в Entity Framework Core була додана функціональність власних типів. Дана функціональність передбачає, що у з декількох типів один є власником (owner). Інші ж залежні або власні типи (owned entity types) є частиною типу-власника і без нього існувати не можуть.

У методі OwnsOne() вказується навігаційна властивість, яке представляє залежний тип.

```
public class AppDbContext : DbContext
{
    0 references | vladkovaliov, 11 days ago | 2 authors, 2 changes
    public AppDbContext(DbContextOptions<AppDbContext> dbContextOptions) : base(dbContextOptions)
    {
        Database.EnsureCreated();
    }

    8 references | Rutakamekiar, 15 days ago | 1 author, 1 change
    public DbSet<GoodEntity> Goods { get; set; }
    6 references | Rutakamekiar, 15 days ago | 1 author, 1 change
    public DbSet<PriceEntity> Prices { get; set; }
    8 references | Rutakamekiar, 15 days ago | 1 author, 1 change
    public DbSet<StoreEntity> Stores { get; set; }
    4 references | vladkovaliov, 10 days ago | 1 author, 1 change
    public DbSet<CompanyEntity> Companies { get; set; }

    0 references | vladkovaliov, 10 days ago | 2 authors, 2 changes
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<GoodEntity>()
                    .HasMany(x => x.Prices)
                    .WithOne()
                    .HasForeignKey(x => x.GoodBarcode);

        modelBuilder.Entity<PriceEntity>()
                    .HasOne(x => x.Store)
                    .WithMany(x => x.Prices)
                    .HasForeignKey(x => x.StoreId);

        modelBuilder.Entity<CompanyEntity>()
                    .HasMany(x => x.Stores)
                    .WithOne(x => x.Company)
                    .HasForeignKey(x => x.RetailName);

        modelBuilder.Entity<StoreEntity>().OwnsOne(x => x.Address);
        base.OnModelCreating(modelBuilder);
    }
}
```

Рис. 3.2 – клас AppDbContext

Для підключення бази даних потрібно додати клас AppDbContext до IServiceCollection в класі Startup.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContextPool<AppDbContext>( optionsAction: options =>
    {
        options.UseSqlServer(Configuration.GetConnectionString( name: "Database"));
    });
}
```

Рис. 3.3 – Підключення бази даних

Dependency injection (DI) або впровадження залежностей представляє механізм, який дозволяє зробити взаємодіючи в додатку об'єкти слабко пов’язаними. Такі об'єкти пов'язані між собою через абстракції, наприклад, інтерфейси, що робить всю систему більш гнучкою, адаптованою і легко розширюваною.

Нерідко для установки залежностей в подібних системах використовуються спеціальні контейнери - IoC-контейнери (Inversion of Control). Такі контейнери служать свого роду фабриками, які встановлюють залежності між абстракціями і конкретними об'єктами і, як правило, управляють створенням цих об'єктів.

І якщо раніше в ASP.NET 4 та попередніх версіях була необхідність використовувати різні зовнішні IoC-контейнери для установки залежностей, такі як Ninject, Autofac, Unity, Windsor Castle, StructureMap, то ASP.NET Core вже має вбудований контейнер впровадження залежностей, який представлений інтерфейсом IServiceProvider. А самі залежності ще називаються сервісами, власне тому контейнер можна назвати провайдером сервісів. Цей контейнер відповідає за зіставлення залежностей з конкретними типами і за впровадження залежностей у різні об'єкти.

За встановлення сервісів в додатку відповідає метод ConfigureServices, визначений у класі Startup.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc()
            .AddNewtonsoftJson(options =>
            {
                options.SerializerSettings.Converters.Add(new StringEnumConverter());
                options.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;
                options.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
            });
    services.AddDbContextPool<AppDbContext>(options =>
    {
        options.UseSqlServer(Configuration.GetConnectionString("Database"));
        options.EnableSensitiveDataLogging();
    });

    var mappingConfig = new MapperConfiguration(mc =>
    {
        mc.AddProfile(new ContractMappingProfile());
    });

    var mapper = mappingConfig.CreateMapper();
    services.AddSingleton(mapper);
    services.AddSingleton<UpdateDbService>();

    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1",
                     new OpenApiInfo
                     {
                         Title = "Goods API",
                         Version = "v1"
                     });
    });
}
```

Рис. 3.4 – Встановлення сервісів

У ASP.NET Core, як і у інших фреймворках для розробки великих веб-застосунків, існує можливість додавати в клас залежності від інших сервісів. Це може здійснюватися п’ятьма способами:

- вказання параметрів методу Configure (лише для класу Startup)
- впровадження залежності через конструктор класу (для всіх класів, крім Startup
- вказання параметрів методу Invoke у так званих middleware-компонентах (список таких компонентів вказується у класі Startup)
- через властивість ApplicationServices об’єкту IApplicationBuilder (передається як параметр у метод Configure класу Startup)
- через властивість RequestServices контексту HttpContext (працює також для middleware-компонентів, патерн, який використовується при цьому, отримав назву Service Locator)

Вбудована в ASP.NET Core система впровадження залежностей використовує конструктори класів для передачі всіх залежностей. Відповідно, і в конструкторі контролера ми можемо отримати залежність. Конструктори є найкращим варіантом для отримання залежностей.

```
public class GoodController : ControllerBase
{
    private readonly AppDbContext _context;
    private readonly IMapper _mapper;
    private readonly ILogger<GoodController> _logger;
    private readonly UpdateDbService _updateDbService;

    0 references | vladkovaliov, 12 days ago | 1 author, 1 change
    public GoodController(AppDbContext context,
                          IMapper mapper,
                          ILogger<GoodController> logger,
                          UpdateDbService updateDbService)
    {
        _context = context;
        _mapper = mapper;
        _logger = logger;
        _updateDbService = updateDbService;
    }
```

Рис. 3.5 – Отримання залежностей через конструктор

Використовуючи різні методи впровадження залежностей, можна керувати життєвим циклом створюваних сервісів. Сервіси, які створюються механізмом Depedency Injection, можуть представляти один з наступних типів:

- Transient: при кожному зверненні до сервісу створюється новий об'єкт сервісу. Протягом одного запиту може бути кілька звернень до сервісу, відповідно при кожному зверненні буде створюватися новий об'єкт. Подібна модель життєвого циклу найбільш підходить для легких сервісів, які не зберігають даних про стан.
- Scoped: для кожного запиту створюється свій об'єкт сервісу. Тобто якщо протягом одного запиту є кілька звернень до одного сервісу, то при всіх цих зверненнях буде використовуватися один і той же об'єкт сервісу.
- Singleton: об'єкт сервісу створюється при першому зверненні до нього, всі наступні запити використовують один і той же раніше створений об'єкт сервісу

Для створення кожного типу сервісу призначений відповідний метод AddTransient (), AddScoped () або AddSingleton ().

3.2.2 API контролер

Перш за все до контролера застосовується атрибут [ApiController], який дозволяє використовувати ряд додаткових можливостей, зокрема, для прив'язки моделі. Також до контролера застосовується атрибут маршрутизації, який вказує, як контролер буде зіставлятися з запитами.

Контролер API призначений переважно для обробки запитів протоколу HTTP: Get, Post, Put, Delete, Patch, Head, Options. В даному випадку для кожного типу запитів в контролері визначено свої методи. Так, метод Get () обробляє запити типу GET і повертає колекцію об'єктів з бази даних.

Якщо запит Get містить параметр id (ідентифікатор об'єкта), то він обробляється іншим методом - Get (int id), який повертає об'єкт по переданому id.

```
[ApiController]
[Route(template: "[controller]")]

public class PriceController : ControllerBase
{
    private readonly AppDbContext _context;


    public PriceController(AppDbContext context)
    {
        _context = context;
    }

    [HttpPost(template: "setOrUpdatePrice")]

    public async Task<IActionResult> SetPricesByBarcode(PriceEntity priceEntity)
    {
        if (_context.Prices.Any(x => x.GoodBarcode == priceEntity.GoodBarcode &&
                                     x.StoreId == priceEntity.StoreId))
        {
            _context.Prices.Update(priceEntity);
        }
        else
        {
            await _context.Prices.AddAsync(priceEntity);
        }

        await _context.SaveChangesAsync();
        return NoContent();
    }
}
```

Рис. 3.6 - Приклад API контролеру

Запити типу Post обробляються методом Post (Customer customer), який отримує з тіла запиту відправлені дані і додає їх у базу даних.

Метод Put (Customer customer) обробляє запити типу Put - отримує дані з запиту і змінює ними об'єкт у базі даних.

Метод Delete (int id) обробляє запити типу Delete, тобто запити на видалення, - отримує із запиту параметр id і по цьому ідентифікатору видаляє об'єкт з бази даних.

При використанні Web API стан обробки запиту на сервері ми можемо контролювати за допомогою статусних кодів:

- 200: статус Ok. Вказує на вдале виконання запиту.
- 201: статус Created. Вказує на успішне створення об'єкта, як правило, використовується в запитах POST.
- 204: статус NoContent - запит пройшов успішно, наприклад, після видалення.
- 400: статус BadRequest - помилка при виконанні запиту.
- 401: статус Unathorized - користувач не авторизований.
- 403: статус Forbidden - доступ заборонений.
- 404: статус NotFound - ресурс не знайдений.

Відправляючи певний статусний код, ми вже даємо клієнту знати про характер виниклої помилки або статусу запиту.

3.3 Мобільний додаток

У Xamarin.Forms візуальний інтерфейс складається зі сторінок. Сторінка являє собою об'єкт класу Page, вона займає весь простір екрану. Тобто те, що ми бачимо на екрані мобільного пристрою, - це сторінка. Додаток може мати одну або кілька сторінок.

Сторінка в якості вмісту приймає один з контейнерів компонування, в який поміщаються стандартні візуальні елементи типу кнопок і текстових полів, а також інші елементи компоновки.

За замовчуванням весь інтерфейс створюється в класі App, який розташовується в файлі App.xaml.cs і представляє поточний додаток.

Робота класу App починається з конструктора, де спочатку викликається метод InitializeComponent(), який виконує ініціалізацію об'єкта, а потім встановлюється властивість MainPage. Через цю властивість клас App встановлює головну сторінку додатка.

Але даний спосіб не єдиний. Xamarin.Forms дозволяє створювати візуальний інтерфейс як за допомогою коду C#, так і декларативним шляхом за допомогою мови xaml, аналогічно html, або комбінуючи ці підходи.

Хоча ми можемо створювати весь інтерфейс в коді мовою C#, кращим способом є його опис в XAML. XAML представляє мову розмітки на основі xml для створення об'єктів декларативним чином. Власне тому при створенні проекту вже за замовчуванням в нього додаються два файли MainPage.xaml і MainPage.xaml.cs.

3.3.1 Переваги використання XAML

По-перше, за допомогою XAML ми можемо відокремити графічний інтерфейс від логіки додатку, завдяки чому над різними частинами додатка можуть щодо автономно працювати різні фахівці: над інтерфейсом - дизайнери, над кодом логіки - програмісти.

По-друге, XAML дозволяє описати інтерфейс більш ясним і зрозумілим способом, такий код набагато простіше підтримувати і оновлювати.

В цілому XAML дозволяє організувати весь призначений для користувача інтерфейс у вигляді набору сторінок подібно до того, як це робиться в HTML.

3.3.2 Структура файлу XAML

Файлом з розміткою на xaml є звичайний файл xml, де в першому рядку розміщено стандартне визначення xml-файлу. Далі визначено елемент ContentPage, який представляє сторінку і всередині якого знаходиться мітка з текстом.

У визначенні кореневого елемента ContentPage підключаються чотири простору імен за допомогою атрибутів xmlns.

Простір імен http://xamarin.com/schemas/2014/forms визначає більшість типів з Xamarin Forms, які застосовуються для побудови графічного інтерфейсу.

Простір імен http://schemas.microsoft.com/winfx/2009/xaml визначає ряд типів XAML і типи CLR. Так як тільки один простір імен може бути базовим, цей простір використовується з префіксом x: xmlns: x. Це означає, що ті властивості елементів, які укладені в цьому просторі імен, будуть використовуватися з префіксом x - x: Name або x: Class

3.3.3 Взаємодія XAML і C#

При додаванні нової сторінки XAML в проект також одночасно додається файл коду C#. Так, при створенні проекту в нього за замовчуванням додається файл з графічним інтерфейсом в XAML - MainPage.xaml і файл MainPage.xaml.cs, де, як передбачається, описана логіка програми, пов'язана з розміткою з файлу XAML.

Файли XAML дозволяє визначити інтерфейс вікна, але для створення логіки додатка, наприклад, для визначення обробників подій елементів управління, нам все одно доведеться скористатися кодом C#.

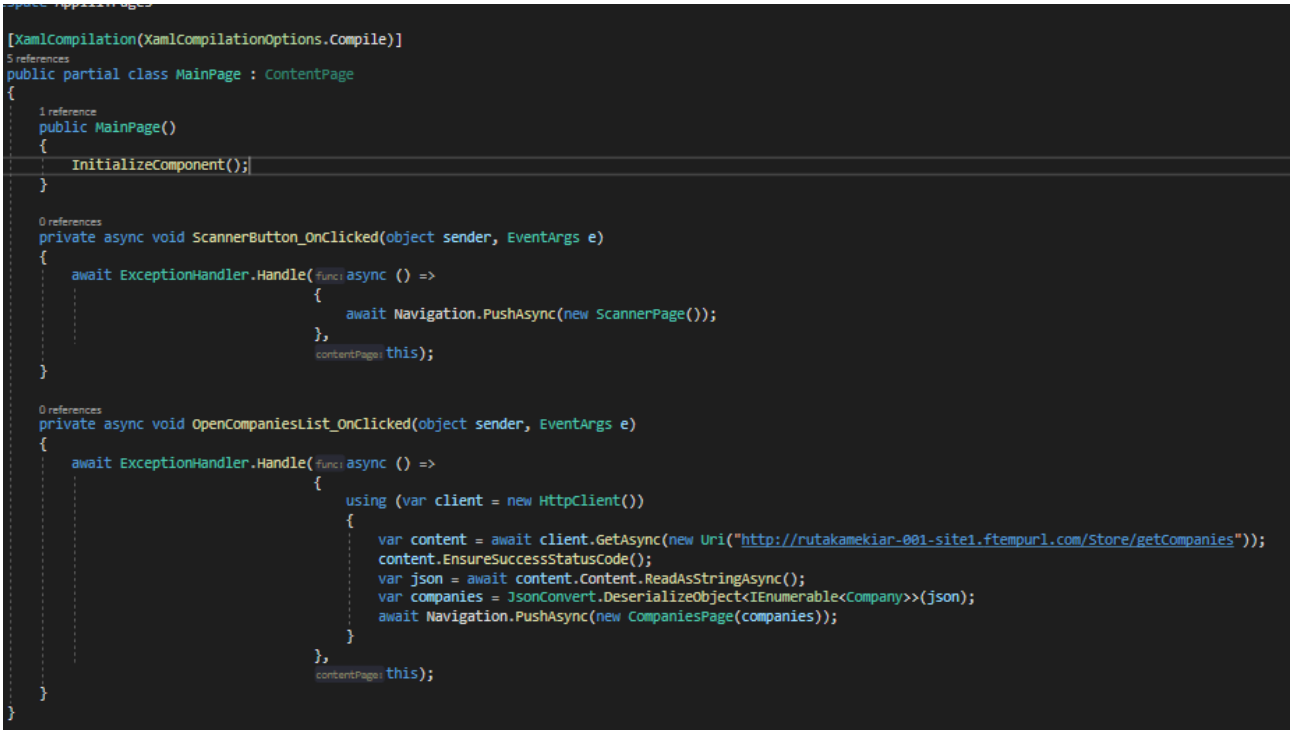
```xml
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             x:Class="App111.Pages.MainPage"
             NavigationPage.HasNavigationBar="False">
    <StackLayout BackgroundColor="GhostWhite">
        <Frame BackgroundColor="Coral" HeightRequest="80" CornerRadius="25" Margin="10"
               BorderColor="Black">
            <Label Text="Check price" HorizontalOptions="Center" FontSize="40" TextColor="GhostWhite"></Label>
        </Frame>

        <Button BackgroundColor="Coral" Text="Our stores" WidthRequest="100" HeightRequest="100" CornerRadius="50"
                TextColor="GhostWhite"  BorderColor="Black" BorderWidth="1" HorizontalOptions="Center"
                Clicked="OpenCompaniesList_OnClicked"></Button>
        <Button BackgroundColor="Coral" Text="Our goods" WidthRequest="100" HeightRequest="100" CornerRadius="50"
                TextColor="GhostWhite"  BorderColor="Black" BorderWidth="1" HorizontalOptions="Center"></Button>

        <Button Margin="0,250,0,0" Text="Search by barcode" Clicked="ScannerButton_OnClicked"
                HorizontalOptions="Center" FontSize="15"
                WidthRequest="200" BackgroundColor="Coral" CornerRadius="20" BorderColor="Black" BorderWidth="1"
                TextColor="GhostWhite">
        </Button>
    </StackLayout>
</ContentPage>
```

Рис. 3.7 – MainPage.xaml

Властивості елементів визначаються у вигляді атрибутів, наприклад, Text = "Search by barcode!". Події також визначаються як атрибути. Наприклад, за допомогою атрибута Clicked визначається вбудоване для події натискання: Clicked = "ScannerButton_OnClicked". А щоб визначити обробник, потрібно перейти в файл MainPage.xaml.cs і визначити в класі MainPage даний метод.



```
[XamlCompilation(XamlCompilationOptions.Compile)]
5 references
public partial class MainPage : ContentPage
{
    1 reference
    public MainPage()
    {
        InitializeComponent();
    }

    0 references
    private async void ScannerButton_OnClicked(object sender, EventArgs e)
    {
        await ExceptionHandler.Handle(func: async () =>
                                      {
                                          await Navigation.PushAsync(new ScannerPage());
                                      },
                                      contentPage: this);
    }

    0 references
    private async void OpenCompaniesList_OnClicked(object sender, EventArgs e)
    {
        await ExceptionHandler.Handle(func: async () =>
                                      {
                                          using (var client = new HttpClient())
                                          {
                                              var content = await client.GetAsync(new Uri("http://rutakamekiar-001-site1.ftempurl.com/Store/getCompanies"));
                                              content.EnsureSuccessStatusCode();
                                              var json = await content.Content.ReadAsStringAsync();
                                              var companies = JsonConvert.DeserializeObject<IEnumerable<Company>>(json);
                                              await Navigation.PushAsync(new CompaniesPage(companies));
                                          }
                                      },
                                      contentPage: this);
    }
}
```

Рис. 3.8 – MainPage.xaml.cs

3.3.4 Контейнери компонування

У Xamarin можемо використовувати ряд елементів. Їх об'єднує те, що всі вони успадковані від загального класу View і тому мають ряд загальних властивостей.

Крім звичайних елементів типу кнопок і текстових полів, в Xamarin Forms також є контейнери, які дозволяють скомпонувати вміст, розташувати його певним чином.

Для визначення вмісту сторінки клас сторінки ContentPage має властивість Content. За замовчуванням цій властивості присвоюється один елемент Label.

Але властивість Content має обмеження - для неї можна встановити тільки один елемент. І щоб розташувати на сторінці відразу кілька елементів, нам треба використовувати один з елементів компоновки.

Елемент компонування представляє класи, які успадковуються від базового класу Layout <T>:

- StackLayout;
- AbsoluteLayout;
- RelativeLayout;
- Grid;
- FlexLayout.

Всі елементи компонування мають властивість Children, що дозволяє задати або отримати вкладені елементи.

3.3.5 Навігація

Для підтримки навігації на кожній сторінці Page в Xamarin Forms визначено властивість Navigation. Ця властивість представляє інтерфейс INavigation, в якому є такі методи:

- Task PushAsync (Page page)
- Task PushModalAsync (Page page)

Як параметр тут передається об'єкт Page - сторінка, на яку треба здійснити перехід.

Другий метод має в своїй назві слово "Modal" і здійснює перехід на модальну сторінку. Зазвичай модальні сторінки використовуються, коли з додатком потрібно отримати деяку інформацію від користувача. При цьому до отримання інформації не можна повертатися на попередню сторінку.

Також є методи для повернення на попередню сторінку:

- Task <Page> PopAsync ()
- Task <Page> PopModalAsync ()

3.3.6 Взаємодія з сервером

Взаємодія з сервером є однією з невід'ємних можливостей багатьох мобільних додатків. Додаток може отримувати дані з сервера, або, навпаки, відправляти деякі дані. Але щоб взаємодіяти з сервером, необхідно підключення до мережі, яка не завжди може бути доступною в силу тих чи інших причин. За допомогою спеціальних API ми можемо перевірити стан підключення до інтернету.

```
1  using Android.Net;
2
3  var cm = (ConnectivityManager)GetSystemService(Application.ConnectivityService);
4  bool isConnected = cm.ActiveNetworkInfo.IsConnected;
```

Рис. 3.9 – Перевірка стану мережі на Android

Використовуючи механізм впровадження залежностей можна побудувати кроссплатформенную абстракцію, через яку при взаємодії з мережею буде отримано стан підключення.

Крім того, можна використовувати ConnectivityPlugin - спеціальний плагін для перевірки підключення, а також для отримання додаткових даних про стан мережі.

Платформи мають ряд обмежень при роботі з мережею, які потрібно враховувати. Зокрема, для отримання стану мережі в Android, треба задати відповідні дозволи. Для цього в проекті для Android у властивостях потрібно перейти на вкладку Android Manifest і додати дозвіл для пунктів ACCESS_NETWORK_STATE і ACCESS_WIFI_STATE.

Для взаємодії з веб-сервісами в .NET використовується клас HttpClient з простору імен System.Net.Http. За допомогою його методів можна посилати певний запит до сервера.

3.3.6.1 Робота з HttpClient

HttpClient надає наступні методи:

- GetAsync (): відправляє запит типу Get;
- DeleteAsync (): відправляє запит типу Delete;
- PostAsync (): відправляє запит типу Post;
- PutAsync (): відправляє запит типу Put;
- SendAsync (): відправляє запит будь-якого типу в залежності від налаштувань заголовків запиту. Вище згадані методи є окремими випадками даного методу;
- GetByteArrayAsync (): відправляє масив байтів в запиті типу Get;
- GetStreamAsync (): відправляє потік Stream в запиті типу Get;
- GetStringAsync (): відправляє рядок в запиті типу Get.

При відправці запиту для його налаштування можна застосовувати об'єкт класу HttpRequestMessage. Зокрема, для конфігурації запиту можна використовувати такі властивості класу:

- Content: відправляються дані;
- Headers: заголовки http;
- Method: метод відправки (Get / Post / Put / Delete);
- RequestUri: адреса сервера.

Дані запиту і відповіді представлені у вигляді об'єкта класу HttpContent. Цей клас є абстрактним, і фактично ми будемо працювати з його похідними класами:

- StringContent: застосовується, якщо відправка або отримання даних відбувається у вигляді рядка;
- ByteArrayContent: застосовується, якщо відправка або отримання даних відбувається у вигляді масиву байтів;
- StreamContent: застосовується, якщо відправка або отримання даних відбувається, як правило, у вигляді файлу, наприклад, зображення.

Якщо потрібно отримати дані у вигляді рядка, масиву байт або файлу, то відповідно можна використовувати один з методів GetStringAsync(), GetByteArrayAsync(), GetStreamAsync(), або можна обробляти властивість Content об'єкта HttpResponseMessage.

```
using (var client = new HttpClient())
{
    var content = await client.GetAsync(new Uri("http://rutakamekiar-001-site1.ftempurl.com/Good/getByBarcode/"
                                                + result.Text)); // Task<HttpResponseMessage>
    if (content.StatusCode == HttpStatusCode.NotFound)
    {
        await DisplayAlert(title: "Error",
                           message: $"Good with barcode {result.Text} not found.",
                           cancel: "Ok");
    }
    else
    {
        var json = await content.Content.ReadAsStringAsync();
        var good = JsonConvert.DeserializeObject<Good>(json);
        await Navigation.PushAsync(new GoodPage(good));
    }

    ActivityIndicator.IsRunning = false;
    zxing.IsAnalyzing = true;
}
```

Рис. 3.10 – Приклад використання HttpClient

3.4 Публікація серверу

Традиційно додатки ASP.NET розгорталися на веб-сервері IIS. Однак оскільки ASP.NET Core має кроссплатформенную природу, треба було відв'язати ASP.NET Core від IIS і в цілому від Windows. І на даний момент ASP.NET Core підтримує розгортання додатку на таких веб-серверах як IIS і IIS Express, а також надає можливість запускати додаток без IIS в рамках власного процесу за допомогою двох додаткових http-серверів, які представлені разом з ASP.NET Core:

- IIS HTTP Server;
- Microsoft.AspNetCore.Server.HttpSys (або просто WebListener);
- Microsoft.AspNetCore.Server.Kestrel (або просто Kestrel).

HTTP.sys працює тільки на платформі Windows, а Kestrel є кросплатформним.

Крім того, якщо додаток використовує Kestrel, то в якості проксі-сервера він може використовувати також IIS, Apache і Nginx. Тобто Apache, Nginx або IIS отримуватимуть запити і перенаправлятимуть їх до додатку, що працює на Kestrel. Схема, при якій запити йдуть не напряму на Kestrel, а проходять через IIS / Apache / Nginx, дозволяє задіяти можливості, що є у цих веб-серверів, але відсутні у Kestrel.

3.4.1 IIS і IIS Express

За замовчуванням додатки ASP.NET працюють з сервером IIS. Однак слід зауважити, що в порівнянні з попередніми версіями ASP.NET при роботі з ASP.NET Core IIS не використовує інфраструктуру System.Web, що значно підвищує продуктивність програми. Крім того, IIS є досить функціональним, має безліч можливостей з адміністрування та управління сервером і розміщеними на ньому додатками.

3.4.2 AspNetCoreModule

Хостування додатків ASP.NET Core на IIS відбувається за допомогою нативного модуля IIS під назвою AspNetCoreModule, який налаштований таким чином, щоб перенаправляти запити на веб-сервер Kestrel. Цей модуль управляє запуском зовнішнього процесу dotnet.exe, в рамках якого хостується додаток, і перенаправляє всі запити від IIS до цього процесу. При отриманні першого запиту до додатку AspNetCoreModule запускає процес для цього додатку і перезапускає процес, якщо додаток падає.

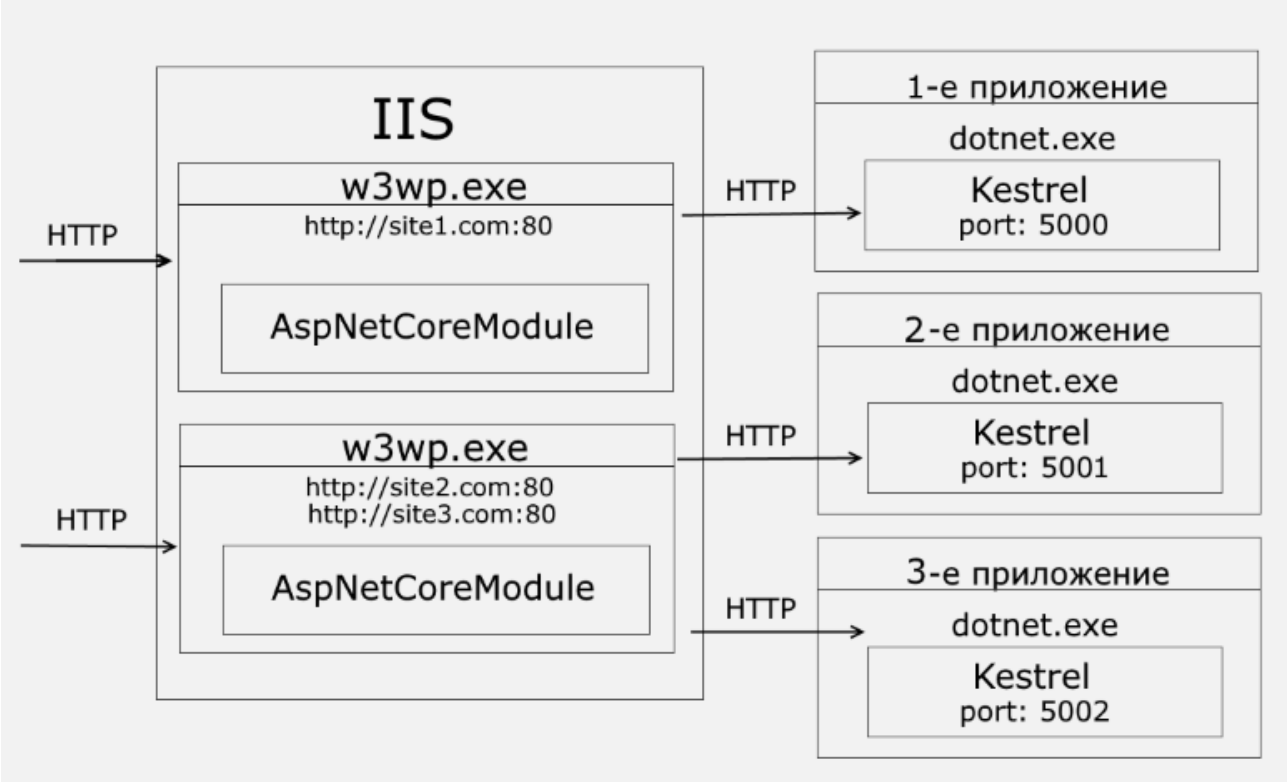


Рис. 3.11 – AspNetCoreModule та IIS

Принцип роботи зображено на рисунку 3.11. Всі вхідні запити потрапляють на драйвер Http.Sys, який перенаправляє їх на порт 80 (стандартний) або на порт 443 (для SSL-з'єднань). Після цього запит обробляється власне додатком ASP.NET Core, який запущено на будь-якому порту, відмінному від 80 та 443. Веб-сервер отримує запит та перетворює його разом з усіма вхідними параметрами на об'єкт HttpContext для передачі в middleware-обробник ASP.NET Core. Після цього виконується обробка запиту додатком і повернення IIS результату, який буде повернено клієнту.

Щоб використовувати даний модуль, його необхідно встановити разом з пакетом ASP.NET Core Server Hosting Bundle. Проте наразі встановлення модуля спростилося, оскільки при використанні середовища розробки Visual Studio воно здійснюється автоматично для IIS Express / IIS, тому зазвичай жодних додаткових дій виконувати не потрібно. Для використання модуля безпосередньо в додатку також необхідно з використанням пакетного менеджера Nuget завантажити пакет Microsoft.AspNetCore.Server.IISIntegration.

Інтеграція з IIS виконується шляхом виклику метода UseIISIntegration об’єкта WebHostBuilder. При виклику здійснюється пошук змінних оточення, необхідних для конфігурації та їх налаштування.

3.4.3 Kestrel

Kestrel – багатоплатформений веб-сервер, що базується на бібліотеці асинхронного введення / виведення libuv. Використовується у проектах на ASP.NET Core за замовчуванням.

При запуску застосунку викликається метод UseKestrel класу WebHostBuilder, що вмикає підтримку Kestrel і дозволяє виконати його налаштування. Навіть якщо не вказувати виклик цього методу у стартовому класі програми, він викликається фреймворком самостійно з стандартними параметрами.

Серед недоліків цього сервера можна вказати необхідність використання зовнішніх проксі-серверів (IIS, Apache, Nginx) для запуску у мережі Інтернет. Але цей підхід також надає і переваги, серед яких можливість приховати додатки, якщо вони не повинні бути доступні безпосередньо, а також можливість керувати навантаженням на сервер.

3.4.4 HTTP.sys

HTTP.sys (раніше – WebListener) – сервер для виконання програм, написаних з використання фреймворку ASP.NET Core (працює лише для операційної системи Windows). Його перевага полягає у відсутності необхідності використовувати додаткові проксі-сервери (IIS, Apache) для коректного запуску, на відміну від Kestrel. Це пов’язано з тим, що Http.Sys дозволяє забезпечити безпеку і надійність самостійно, без жодних допоміжних ресурсів.

3.4.5 Вибір хостингового серверу

Основними перевагами SmarterASP.NET є його безкоштовність на пробний період та орієнтація на asp.net сервери. Ці особливості роблять даний хостинг доступним для розробників, що працюють без команди, а також зручним для інтеграції з розроблюваним додатком.

Основними недоліками безкоштовного серверу є низька пропускна спроможність та відносно повільна обробка запитів, але даними особливостями можна знехтувати за відсутності реальних користувачів та великого навантаження системи.

Висновки до розділу 3

В даному розділі було подано опис архітектури розроблюваного програмного забезпечення. Також наведено структуру БД з детальнім описом таблиць. Крім того, описано архітектурні та програмні рішення, застовповані до клієнтського додатку та серверної частини проекту.

РОЗДІЛ 4

Огляд розробленого продукту

На головній сторінці додатку розташовуються 2 кнопки. Перша – “OUR STORES” та друга – “SEARCH BY BARCODE”.

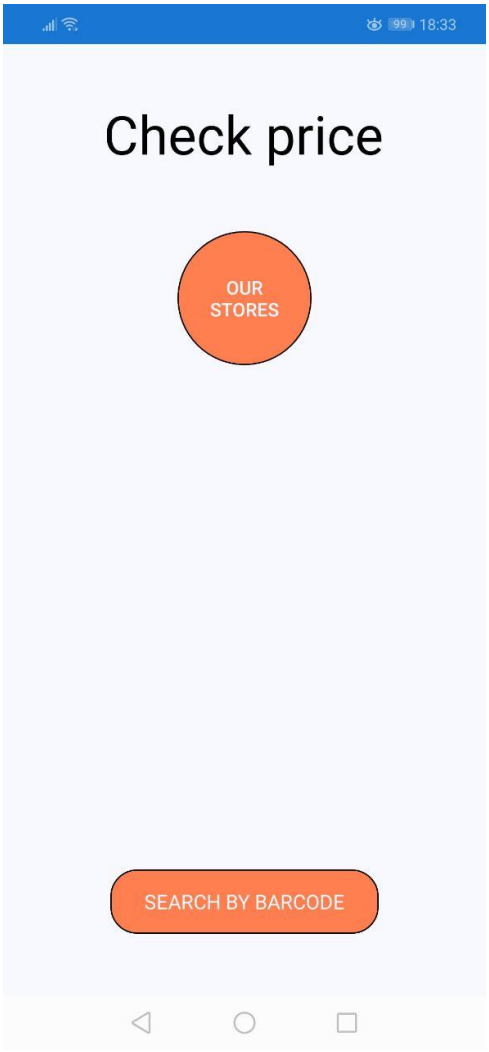


Рис. 4.1 – Головна сторінка

При натисканні на першу кнопку відкривається сторінка з переліком мереж магазинів, при наступному подвійному натисканні на одну з мереж, відкривається список назв та адрес магазинів цієї мережі.

Рис. 4.2 – Список підтримуваних мереж магазинів.

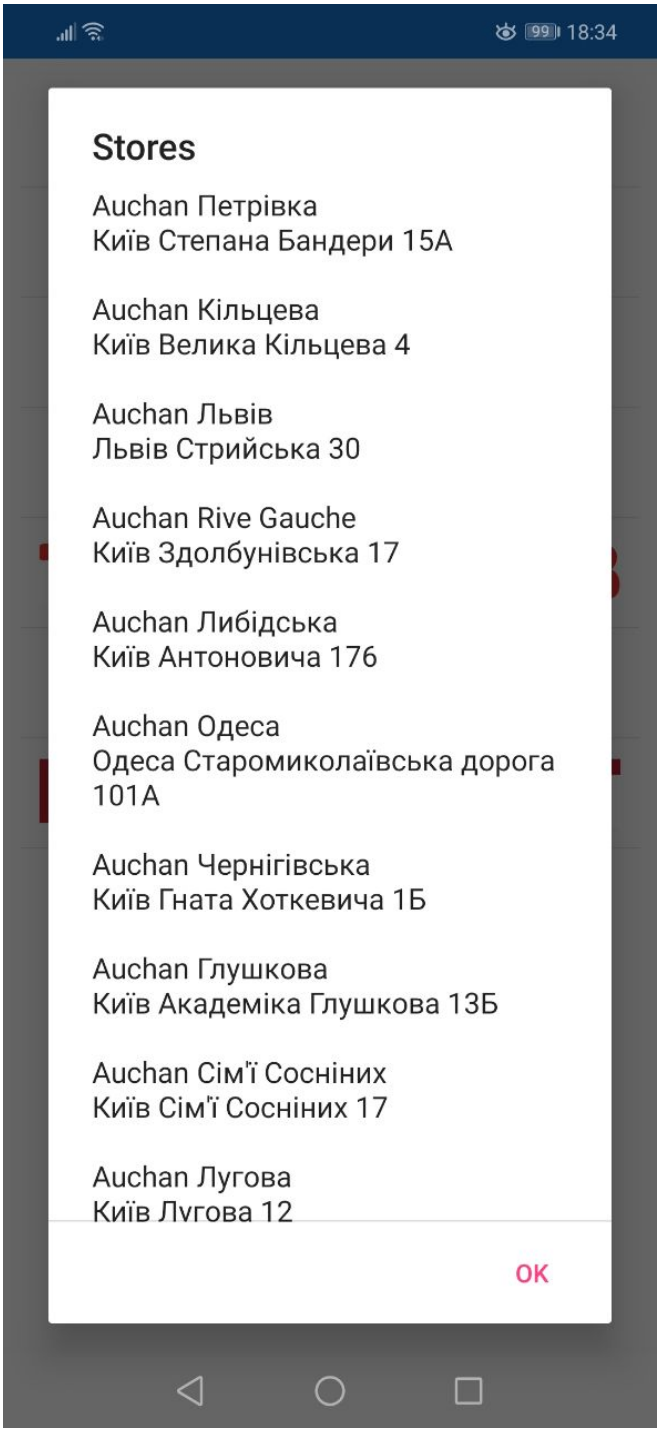


Рис. 4.3 – Результат подвійного натискання по одній з іконок магазинів

При натисканні на кнопку “SEARCH BY BARCODE” відкривається камера, яка буде зчитувати штрихкод з товару.

Рис. 4.4 – Режим зчитування штрихкоду

Якщо товар не знайдено, то з’явиться відповідне повідомлення.

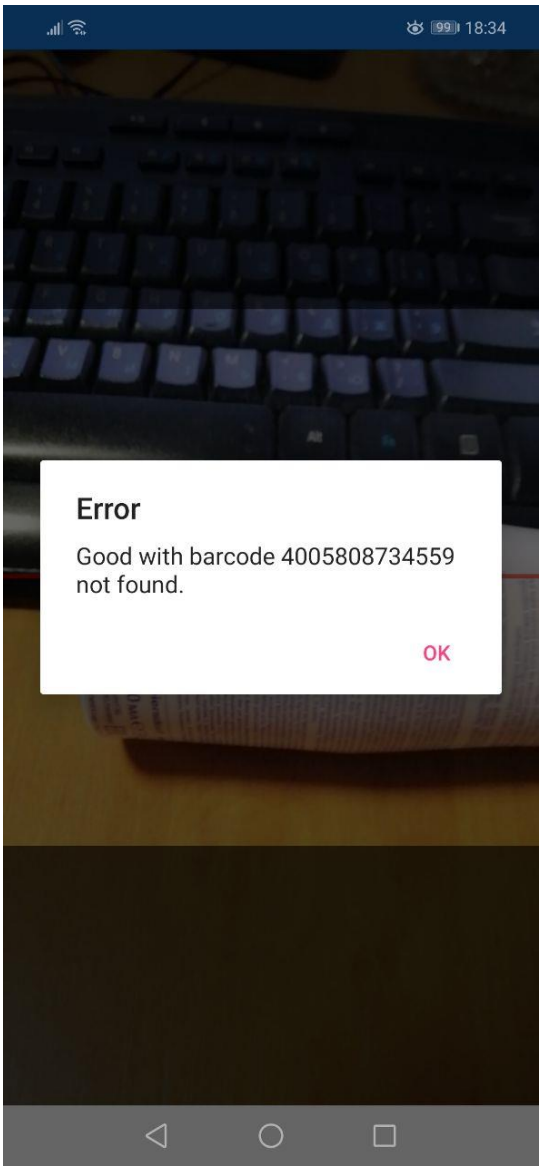


Рис. 4.5. – Повідомлення про відсутність товару

При вдалому пошуку відкриється нове вікно з відповідним товаром, де буде представлена картинка товару, його назва, штрихкод та список мереж магазинів з мінімальною та максимальною ціною в них.

Рис. 4.6. – Знайдений товар

При подвійному натисканні на назву мережі магазинів, відкривається вікно з переліком магазинів (назва, адреса та ціна у цьому магазині) у яких даний товар наявний.

На кожній зі сторінок наявна відповідна кнопка для повернення на головну сторінку.

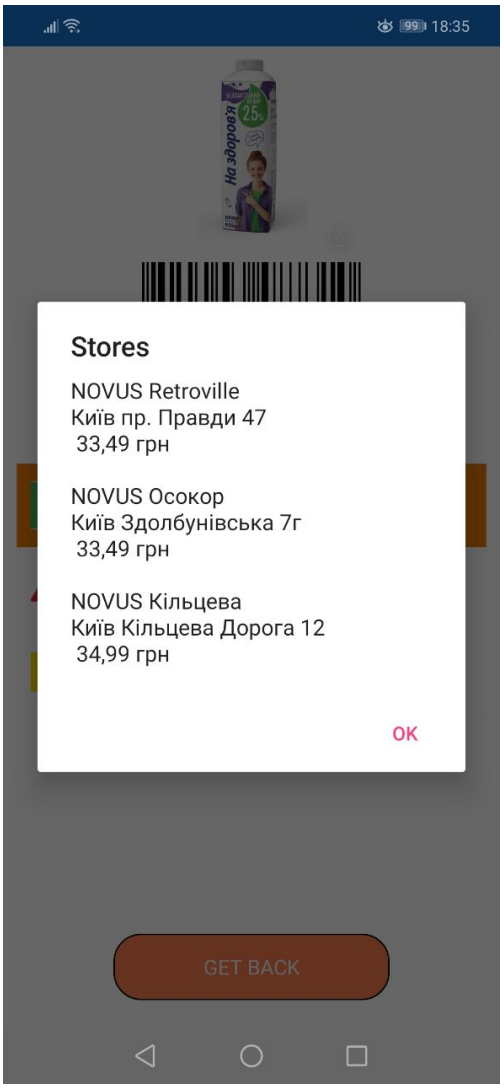


Рис.4.7. – Список назв та адрес магазинів, та ціна на даний товар у них

Висновки до розділу 4

В даному розділі подано опис варіантів використання програмного забезпечення у вигляді короткої інструкції користувача. Також наведено ілюстрації до кожного з варіантів.

ВИСНОВОК

Дипломний проект призначено для розробки мобільного застосунку для моніторингу вартості товарів.

Метою розробки є створення застосунку на смартфон, що допоможе покупцям дізнаватися актуальні ціни на продовольчі товари у різних продуктових мережах за штрихкодом, а також відслідковувати наявність таких товарів в асортименті магазинів.

Хоча розроблене програмне забезпечення має деякі недоліки, зокрема роботу лише з продовольчими товарами та пошук лише за штрихкодом, архітектура програми дозволяє легко розширити функціонал додатку у майбутньому.

У розділі «Аналіз існуючих програмних продуктів» було виконано огляд предметної області, зокрема розглянуто види та особливості штрихкодів. Також було проведено аналіз додатків та приладів з аналогічним функціоналом. Базуючись на проаналізованих даних, було визначено вимоги до розроблюваного застосунку.

У розділі «Вибір технологій і платформ» було подано опис засобів програмування, застосованих для розробки. Було визначено їх особлівості та переваги.

У розділі «Проектування та реалізація» було представлено архітектуру додатку. Також було описано структуру бази даних, подано особливості реалізації клієнтської (мобільного додатку) та серверної частин.

У розділі «Огляд розробленого продукту» було описано дії, що користувач може виконати з розробленим додатком. Окрім цього, подано ілюстрації роботи програмного забезпечення.

Результатом даного дипломного проекту є мобільний застосунок для моніторингу вартості товарів, що складається з мобільного додатку та серверної частини.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Штрихкод: https://uk.wikipedia.org/wiki/%D0%A8%D1%82%D1%80%D0%B8%D1%85-%D0%BA%D0%BE%D0%B4
2. Сервіс hotline: https://hotline.ua/
3. Xamarin: https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin
4. Asp.Net Core:https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1
5. EF Core: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx
6. Asp.Net Core: https://metanit.com/sharp/aspnet5/
7. Web.Api: https://metanit.com/sharp/aspnet5/23.1.php
8. EF Core: https://metanit.com/sharp/entityframeworkcore/1.1.php
9. Asp.Net Core: https://habr.com/ru/post/312226/
10. Прайс Марк C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов.
11. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд.
12. SQL Server: https://uk.wikipedia.org/wiki/Microsoft_SQL_Server
13. SQL Server: https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-ver15?view=sql-server-ver15

Додатки

ДОДАТОК 1

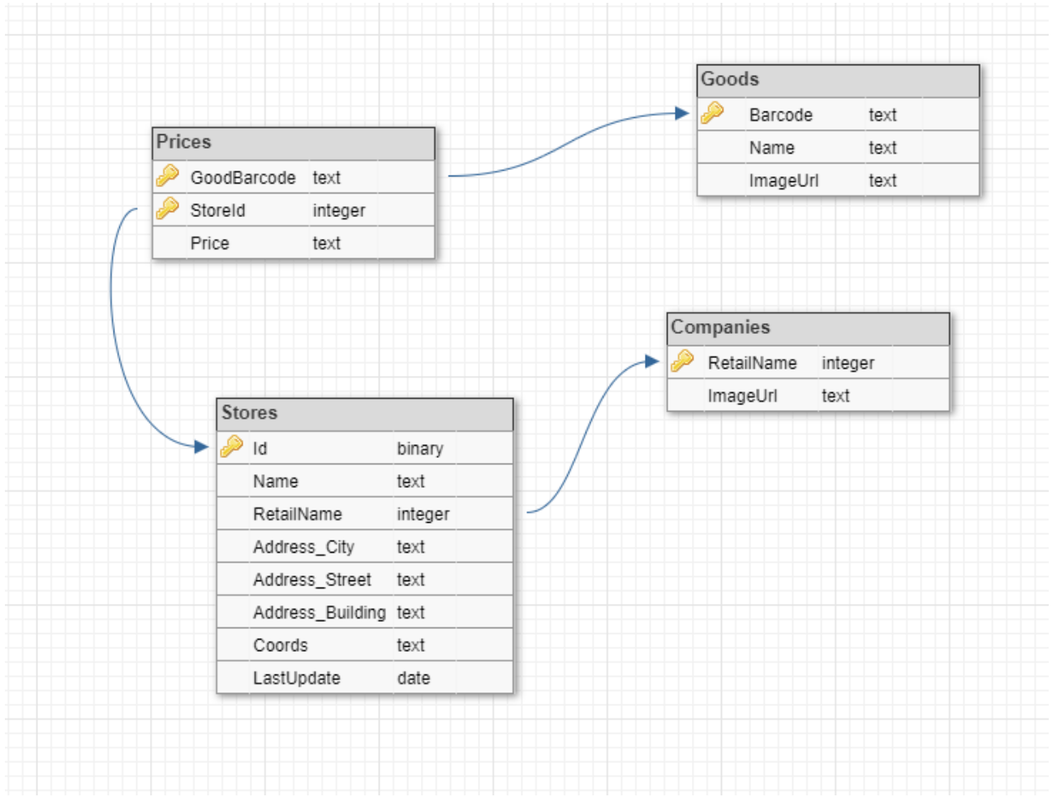
МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ МОНІТОРИНГУ ВАРТОСТІ ТОВАРІВ

Схема бази даних

ІАЛЦ.467100.004 Д1

Аркушів 1

2020



					ІАЛЦ.467100.004 Д1	
					Схема бази даних	Літ. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Ковальов В.П				Арк. / Аркушів
Перевір.		Габінет А.В.			Дипломна робота	КПІ ім.Ігоря Сікорського, ФІОТ кафедра ОТ, гр. ІП-62
Н. контр.		Сімоненко В.П				
Затверд.						

ДОДАТОК 2

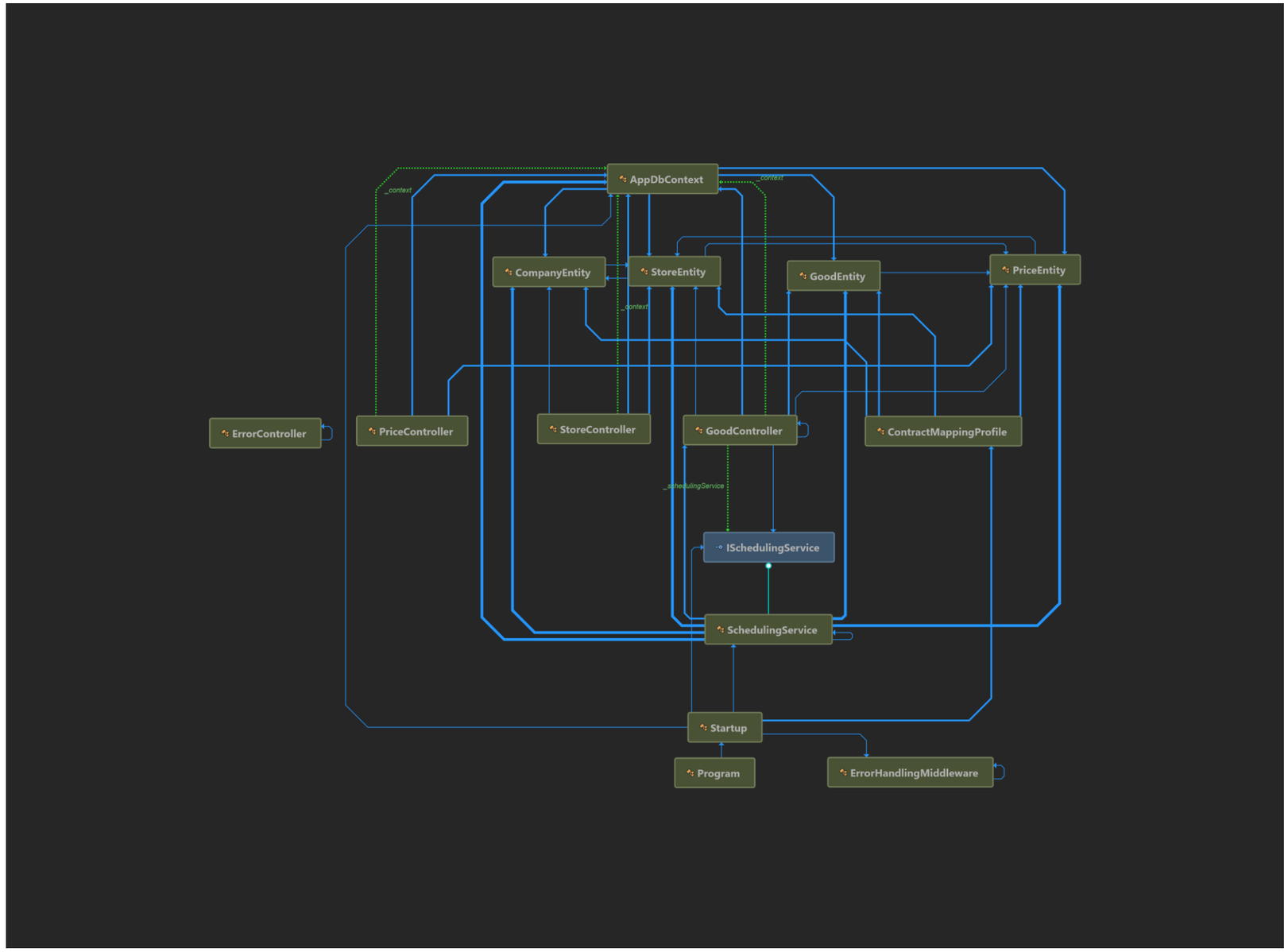
МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ МОНІТОРИНГУ ВАРТОСТІ ТОВАРІВ

Схема зв’язків між класами

ІАЛЦ.467100.005 Д2

Аркушів 1

2020



					ІАЛЦ.467100.005 Д2	
					Схема зв’язків між класами	Літ. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Ковальов В.П				Арк. Аркушів
Перевір.		Габінет А.В.				
Н. контр.		Сімоненко В.П			Дипломна робота	КПІ ім.Ігоря Сікорського, ФІОТ кафедра ОТ гр. ІП-62
Затверд.						

ДОДАТОК З

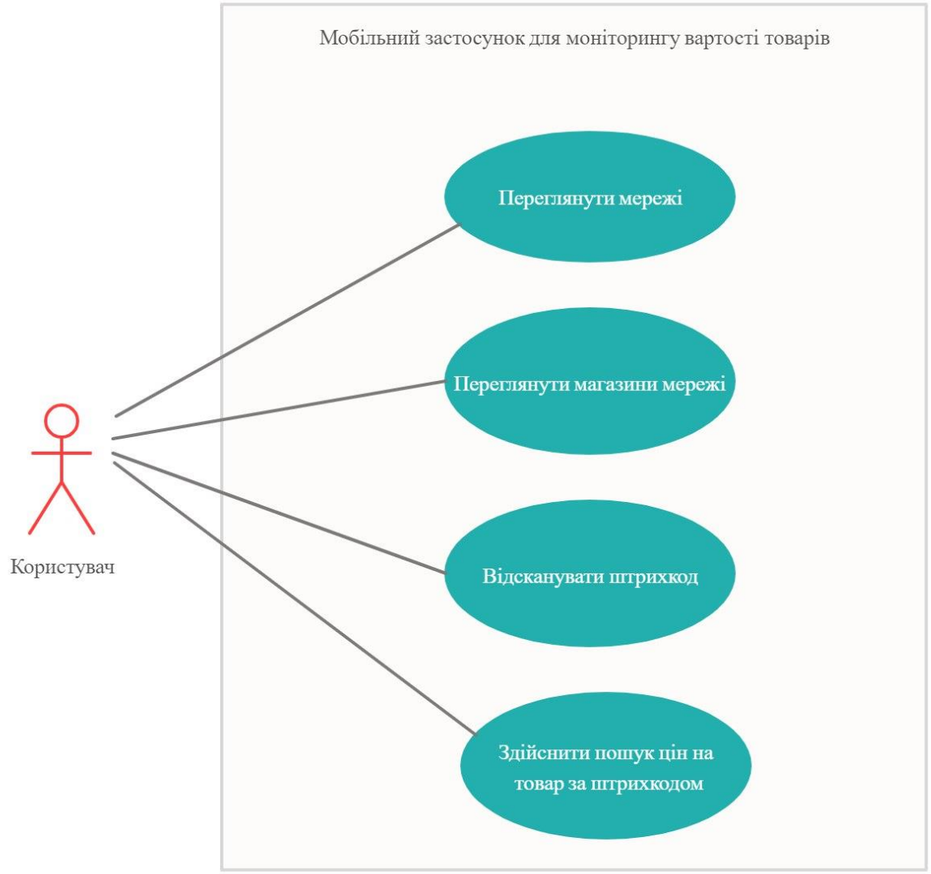
МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ МОНІТОРИНГУ ВАРТОСТІ ТОВАРІВ

Схема прецедентів

ІАЛЦ.467100.006 ДЗ

Аркушів 1

2020



					ІАЛЦ.467100.006 ДЗ	
					Схема прецедентів	Літ. / Маса / Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Ковальов В.п				
Перевір.		Габінет А.В.				Арк. / Аркушів
Н. контр.		Сімоненко В.П			Дипломна робота	КПІ ім.Ігоря Сікорського, ФІОТ кафедра ОТ гр. ІП-62
Затверд.						

ДОДАТОК 4

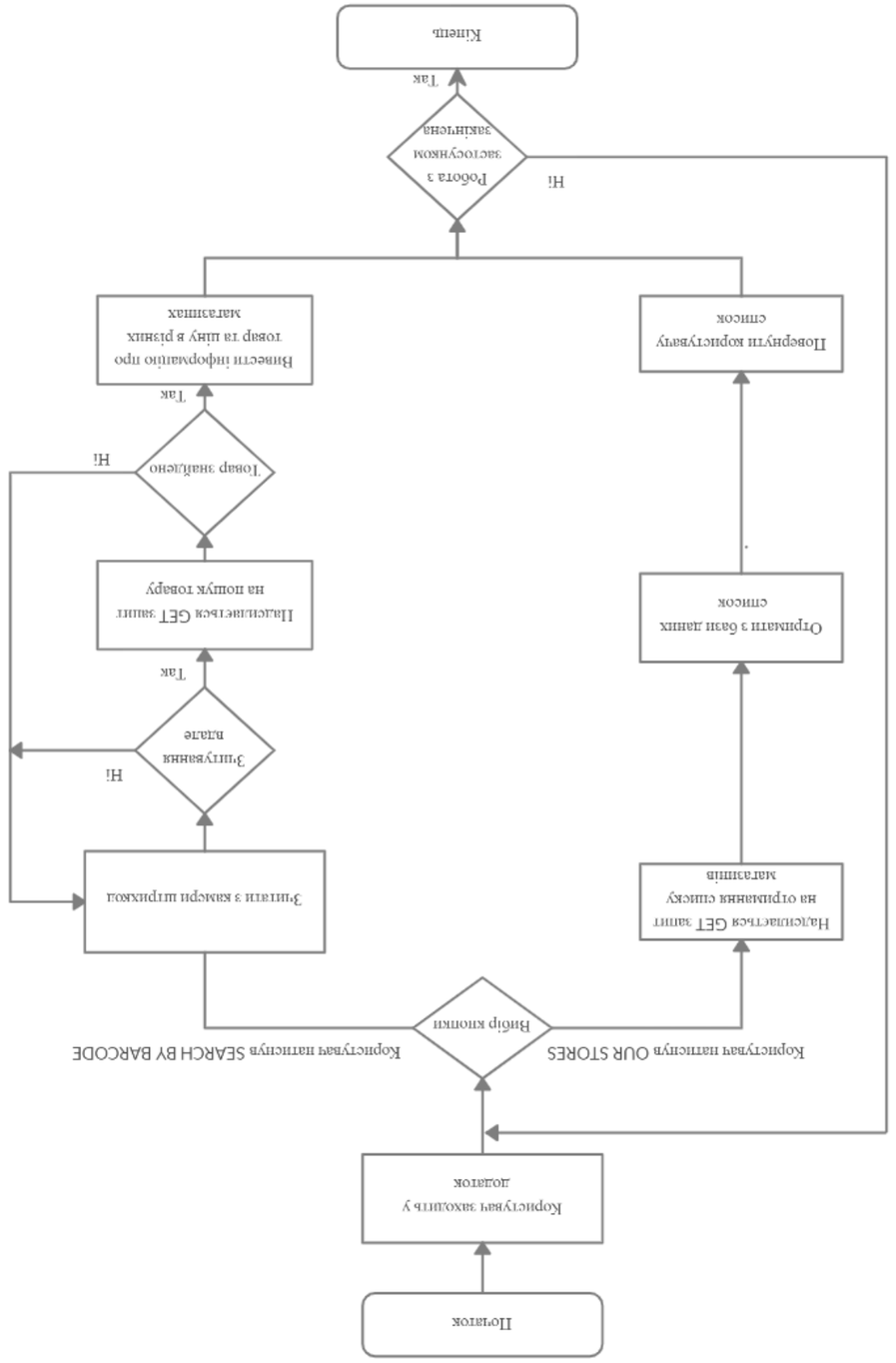
МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ МОНІТОРИНГУ ВАРТОСТІ ТОВАРІВ

Принципова схема

ІАЛЦ.467100.007 Д4

Аркушів 1

2020



Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.467100.007 Д4		
Розроб.		Ковальов В.п			Принципова схема	Літ.	Маса	Масштаб
Перевір.		Гаєцет А.В.						
						Арк.	Аркушів
Н. контр.		Сімоненко В.П					
Затверд.					Дипломна робота	КПІ ім.Ігоря Сікорського, ФІОТ кафедра ОТ гр. ІП-62	

ДОДАТОК 5

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ МОНІТОРИНГУ ВАРТОСТІ ТОВАРІВ

Текст програми

ІАЛЦ.467100.008 Д5

Аркушів 35

2020

GoodsMonitoring.dll

```
public class Program
   {
      public static void Main(string[] args)
      {
         var logger =
NLogBuilder.ConfigureNLog("nlog.config").GetCurrentClassLogger();
         try
         {
            logger.Debug("App started.");
            CreateHostBuilder(args)
               .Build()
               .Run();
         }
         catch (Exception ex)
         {
            logger.Error(ex, "Stopped App because of unhandled exception");
            throw;
         }
         finally
         {
            NLog.LogManager.Shutdown();
         }
      }

      public static IHostBuilder CreateHostBuilder(string[] args) =>
         Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
               webBuilder.UseContentRoot(Directory.GetCurrentDirectory())
                     .UseIISIntegration()
                     .UseStartup<Startup>();
            })
            .ConfigureLogging(logging =>
            {
               logging.ClearProviders();
               logging.SetMinimumLevel(LogLevel.Information);
            })
            .UseNLog();
   }
public class Startup
   {
```

```
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc()
                    .AddNewtonsoftJson(options =>
                    {
                        options.SerializerSettings.Converters.Add(new
StringEnumConverter());
                        options.SerializerSettings.ReferenceLoopHandling =
ReferenceLoopHandling.Ignore;
                        options.SerializerSettings.ContractResolver = new
CamelCasePropertyNamesContractResolver();
                    });
            services.AddDbContextPool<AppDbContext>(options =>
            {

options.UseSqlServer(Configuration.GetConnectionString("Database"));
                options.EnableSensitiveDataLogging();
            });

            var mappingConfig = new MapperConfiguration(mc =>
            {
                mc.AddProfile(new ContractMappingProfile());
            });

            var mapper = mappingConfig.CreateMapper();
            services.AddSingleton(mapper);
            services.AddSingleton<ISchedulingService, SchedulingService>();

            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1",
                        new OpenApiInfo
                        {
                            Title = "Goods API",
                            Version = "v1"
```

```
                    });
            });
        }

        public void Configure(IApplicationBuilder app,
                              IWebHostEnvironment env,
                              IHostApplicationLifetime appLifetime)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseHttpsRedirection();
            app.UseSwagger();
            app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Goods API"));
            app.UseRouting();
            app.UseAuthorization();
            app.UseMiddleware<ErrorHandlingMiddleware>();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
            appLifetime.ApplicationStarted.Register(() =>
            {
                var schedulingService =
app.ApplicationServices.GetService<ISchedulingService>();
                schedulingService.Startup();
            });

            appLifetime.ApplicationStopped.Register(() =>
            {
                var schedulingService =
app.ApplicationServices.GetService<ISchedulingService>();
                schedulingService.Shutdown();
            });
        }
    }
public class AppDbContext : DbContext
    {
```

```
        public AppDbContext(DbContextOptions<AppDbContext>
dbContextOptions) : base(dbContextOptions)
        {
            Database.EnsureCreatedAsync();
        }

        public DbSet<GoodEntity> Goods { get; set; }
        public DbSet<PriceEntity> Prices { get; set; }
        public DbSet<StoreEntity> Stores { get; set; }
        public DbSet<CompanyEntity> Companies { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<GoodEntity>()
                    .HasMany(x => x.Prices)
                    .WithOne()
                    .HasForeignKey(x => x.GoodBarcode);

            modelBuilder.Entity<PriceEntity>()
                    .HasOne(x => x.Store)
                    .WithMany(x => x.Prices)
                    .HasForeignKey(x => x.StoreId);

            modelBuilder.Entity<CompanyEntity>()
                    .HasMany(x => x.Stores)
                    .WithOne(x => x.Company)
                    .HasForeignKey(x => x.RetailName);

            modelBuilder.Entity<PriceEntity>()
                    .HasKey(x => new { x.GoodBarcode, x.StoreId });
            modelBuilder.Entity<StoreEntity>().OwnsOne(x => x.Address);
            base.OnModelCreating(modelBuilder);
        }
    }
public class CompanyEntity
    {
        [Key]
        public StoreRetailName RetailName { get; set; }

        public string ImageUrl { get; set; }

        public ICollection<StoreEntity> Stores { get; set; }
```

```
    }
public class GoodEntity
    {
        [Key]
        [StringLength(13)]
        public string Barcode { get; set; }

        public string Name { get; set; }

        public string ImageUrl { get; set; }

        public ICollection<PriceEntity> Prices { get; set; }
    }
public class PriceEntity
    {
        public int StoreId { get; set; }

        public StoreEntity Store { get; set; }

        public decimal Price { get; set; }

        public string GoodBarcode { get; set; }
    }
public class StoreEntity
    {
        [DatabaseGenerated(DatabaseGeneratedOption.None)]
        public int Id { get; set; }

        public string Name { get; set; }

        [JsonProperty("retail_chain")]
        public StoreRetailName RetailName { get; set; }

        public CompanyEntity Company { get; set; }

        public Address Address { get; set; }

        public string Coords { get; set; }

        public DateTime LastUpdate { get; set; }
```

```
        public IEnumerable<PriceEntity> Prices { get; set; }
    }

public class ContractMappingProfile : Profile
    {
        public ContractMappingProfile()
        {
            CreateMap<GoodEntity, Good>()
                .ForMember(x => x.Companies,
                        x => x.MapFrom(y => y.Prices.Select(z =>
z.Store.Company).DistinctBy(z => z.RetailName)))
                .ReverseMap();
            CreateMap<StoreEntity, Store>().ReverseMap();
            CreateMap<CompanyEntity, Company>().ReverseMap();

            CreateMap<PriceEntity, PriceModel>().ReverseMap();
        }
    }

public class ErrorHandlingMiddleware
    {
        private readonly RequestDelegate _next;
        private readonly ILogger<ErrorHandlingMiddleware> _logger;

        public ErrorHandlingMiddleware(RequestDelegate next,
ILogger<ErrorHandlingMiddleware> logger)
        {
            _next = next;
            _logger = logger;
        }

        public async Task InvokeAsync(HttpContext context)
        {
            try
            {
                await _next(context);
            }
            catch (Exception ex)
            {
                await HandleExceptionAsync(context, ex);
            }
        }
```

```
        private async Task HandleExceptionAsync(HttpContext context, Exception
ex)
        {
            var code = (int)HttpStatusCode.InternalServerError; // 500 if unexpected

            context.Response.ContentType = "text/html";
            context.Response.StatusCode = code;
            var message = ex.StackTrace + "\n" + ex.Message + "\n" +
ex.InnerException?.Message;
            _logger.LogError(ex, ex.Message);
            await context.Response.WriteAsync(message);
        }
    }
public interface ISchedulingService
    {
        void Startup();

        void Shutdown();
    }


public class SchedulingService : ISchedulingService
    {
        private readonly IServiceProvider _serviceProvider;
        private readonly ILogger<SchedulingService> _logger;
        private readonly object _lock;


        public SchedulingService(IServiceProvider serviceProvider,
ILogger<SchedulingService> logger)
        {
            _serviceProvider = serviceProvider;
            _logger = logger;
            _lock = new object();
        }
```

```
public void Startup()
{
    JobManager.JobException += info =>
    {
        lock (_lock)
        {
            _logger.LogError(info.Exception.Message);
        }
    };
    Reload();
}

public void Shutdown()
{
    lock (_lock)
    {
        JobManager.StopAndBlock();
    }
}

public void Reload()
{
    lock (_lock)
    {
        JobManager.StopAndBlock();
        var registry = new Registry();
        using var scope = _serviceProvider.CreateScope();
        registry.Schedule(() => UpdateDatabase().Wait())
```

```
                .ToRunOnceAt(DateTime.Now)
                .AndEvery(1)
                .Days()
                .At(0, 1);
            registry.Schedule(() => UpdateStores().Wait())
                .ToRunOnceAt(DateTime.Today.AddDays(1))
                .AndEvery(7)
                .Days()
                .At(0, 0);
            JobManager.RemoveAllJobs();
            JobManager.Initialize(registry);
        }
    }

    public async Task UpdateStores()
    {
        using var client = new HttpClient();
        client.DefaultRequestHeaders.Add("Accept-Language", "uk-
UA;q=0.5");
        var response = await client.GetAsync("https://stores-
api.zakaz.ua/stores");
        var stores =
JsonConvert.DeserializeObject<IEnumerable<StoreEntity>>(await
response.Content.ReadAsStringAsync());
        await DoWithContextAsync(async context =>
        {
            foreach (StoreRetailName retailName in
Enum.GetValues(typeof(StoreRetailName)))
```

```
            {
                var companyEntity = new CompanyEntity
                {
                    RetailName = retailName,
                    ImageUrl = retailName switch
                    {
                        StoreRetailName.Auchan                    =>
"https://auchan.zakaz.ua/static/i/auchan-logo.svg",
                        StoreRetailName.Novus                     =>
"https://novus.zakaz.ua/static/i/novus-logo.svg",
                        StoreRetailName.Furshet                   =>
"https://furshet.zakaz.ua/static/i/furshet-logo.svg",
                        StoreRetailName.Metro                     =>
"https://metro.zakaz.ua/static/i/metro-logo.svg",
                        StoreRetailName.Tavriav                   =>
"https://tavriav.zakaz.ua/static/i/tavriav-logo.svg",
                        StoreRetailName.Megamarket                =>
"https://megamarket.zakaz.ua/static/i/megamarket-logo.svg",
                        StoreRetailName.Vostorg                   =>
"https://vostorg.zakaz.ua/static/i/vostorg-logo.svg",
                        _ => throw new ArgumentOutOfRangeException()
                    }
                };

                if (!context.Companies.Any(x => x.RetailName == retailName))
                {
                    await context.Companies.AddAsync(companyEntity);
                }
```

```
                else
                {
                    context.Companies.Update(companyEntity);
                }
            }

            await context.SaveChangesAsync();

            foreach (var storeEntity in stores)
            {
                if (context.Stores.Any(x => x.Id == storeEntity.Id))
                {
                    context.Stores.Update(storeEntity);
                    context.Entry(storeEntity).Property(x                    =>
x.LastUpdate).IsModified = false;
                }
                else
                {
                    await context.Stores.AddAsync(storeEntity);
                }
            }

            await context.SaveChangesAsync();
        });
    }

    private async Task UpdateDatabase()
    {
```

```
DoWithLog(x => x.LogInformation("Goods and prices start to update at {0}", DateTime.UtcNow));

var stores = await DoAndReturnWithContextAsync(async context => await context.Stores.ToArrayAsync());
for (var i = 0; i < stores.Length; i++)
{
   var store = stores[i];
   var storeId = store.Id;
   DoWithLog(x => x.LogInformation("Start update store {0} with id {1} and index {2}", store.Name, store.Id, i));
   try
   {
      var goods = new List<GoodEntity>();
      var prices = new List<PriceEntity>();
      using (var client = new HttpClient())
      {
         client.DefaultRequestHeaders.Add("Accept-Language", "uk-UA;q=0.5");

         var categories = await GetStoreCategories(client, storeId);

         foreach (var category in categories)
         {
            var response = await client.GetAsync($"https://stores-api.zakaz.ua/stores/{storeId}/categories/{category}/products");
            var goodArr = JObject.Parse(await response.Content.ReadAsStringAsync())["results"];
```

```
                foreach (var good in goodArr)
                {
                    var barcode = good["ean"].Value<string>();
                    var goodEntity = new GoodEntity
                    {
                        Barcode = barcode.Substring(barcode.Length - 13),
                        Name = good["title"].Value<string>(),
                        ImageUrl = good["img"]["s1350x1350"].Value<string>()
                    };
                    var priceEntity = new PriceEntity
                    {
                        Price = good["price"].Value<decimal>() / 100,
                        StoreId = storeId,
                        GoodBarcode = goodEntity.Barcode
                    };
                    goods.Add(goodEntity);
                    prices.Add(priceEntity);
                }
            }
        }

        await UpdateStore(store);

        await UpdateGoods(goods);
        await UpdatePrices(prices);
        await UpdateStore(store);
        DoWithLog(x => x.LogInformation("Goods and prices with store id
{0} and name {1} updated successfully", store.Id, store.Name));
```

```
                var i1 = stores.Length - i + 1;
                DoWithLog(x => x.LogInformation("{0} stores left", i1));
            }
            catch (Exception e)
            {
                DoWithLog(x => x.LogError(e, "Goods and prices with store id {0} and name {1} updated unsuccessfully", store.Id, store.Name));
            }
        }

        DoWithLog(x => x.LogInformation("Goods and prices updated at {0}", DateTime.UtcNow));
    }

    private async Task<IEnumerable<string>> GetStoreCategories(HttpClient client, int storeId)
    {
        var categories = new List<string>();
        try
        {
            var response = await client.GetAsync($"https://stores-api.zakaz.ua/stores/{storeId}/categories/");
            var json = await response.Content.ReadAsStringAsync();
            var jArray = JArray.Parse(json);

            foreach (var category in jArray)
            {
                AddCategory(category);
```

```
        }

        void AddCategory(JToken category)
        {
            categories.Add(category["id"].Value<string>());
            var children = category["children"];
            if (children.HasValues)
            {
                foreach (var child in children)
                {
                    AddCategory(child);
                }
            }
        }
    }
    catch (Exception e)
    {
        DoWithLog(x => x.LogError(e, "Get categories end unsuccessfully"));
        throw;
    }

    return categories;
}

private async Task UpdatePrices(IEnumerable<PriceEntity> prices)
{
    var uniqPrices = prices.DistinctBy(x => x.GoodBarcode).ToArray();
    await DoWithContextAsync(async context =>
```

```
            {
                foreach (var priceEntity in uniqPrices)
                {
                    if (context.Prices.Any(x => x.GoodBarcode == priceEntity.GoodBarcode && x.StoreId == priceEntity.StoreId))
                    {
                        context.Prices.Update(priceEntity);
                    }
                    else
                    {
                        await context.Prices.AddAsync(priceEntity);
                    }
                }

                await context.SaveChangesAsync();
                DoWithLog(x => x.LogInformation(uniqPrices.Length + " prices saved"));
            });
        }

        private async Task UpdateGoods(IEnumerable<GoodEntity> goods)
        {
            var uniqGoods = goods.DistinctBy(x => x.Barcode).ToArray();
            await DoWithContextAsync(async context =>
            {
                foreach (var goodEntity in uniqGoods)
                {
                    if (context.Goods.Any(x => x.Barcode == goodEntity.Barcode))
```

```
                {
                    context.Goods.Update(goodEntity);
                }
                else
                {
                    await context.Goods.AddAsync(goodEntity);
                }
            }

            await context.SaveChangesAsync();
            DoWithLog(x => x.LogInformation(uniqGoods.Length + " goods saved"));
        });
    }

    private async Task UpdateStore(StoreEntity store)
    {
        await DoWithContextAsync(async context =>
        {
            store.LastUpdate = DateTime.UtcNow;
            context.Stores.Update(store);
            await context.SaveChangesAsync();
        });
    }

    private async Task DoWithContextAsync(Func<AppDbContext, Task> func)
    {
```

```
            using var scope = _serviceProvider.CreateScope();
            var context = scope.ServiceProvider.GetService<AppDbContext>();
            await func.Invoke(context);
        }

        private async Task<T> DoAndReturnWithContextAsync<T>(Func<AppDbContext, Task<T>> func)
        {
            using var scope = _serviceProvider.CreateScope();
            var context = scope.ServiceProvider.GetService<AppDbContext>();
            return await func.Invoke(context);
        }

        private void DoWithLog(Action<ILogger<SchedulingService>> action)
        {
            using var scope = _serviceProvider.CreateScope();
            var logger = scope.ServiceProvider.GetService<ILogger<SchedulingService>>();
            action.Invoke(logger);
        }
    }


    [ApiController]
    [Route("[controller]")]
    public class GoodController : ControllerBase
    {
        private readonly AppDbContext _context;
        private readonly IMapper _mapper;
        private readonly ILogger<GoodController> _logger;
        private readonly ISchedulingService _schedulingService;
```

```
        public GoodController(AppDbContext context,
                              IMapper mapper,
                              ILogger<GoodController> logger,
                              ISchedulingService schedulingService)
        {
            _context = context;
            _mapper = mapper;
            _logger = logger;
            _schedulingService = schedulingService;
        }

        [HttpPost("getPage")]
        public async Task<IActionResult> Get(PageSettings pageSettings)
        {
            return Ok(await _context.Goods.Include(x =>
x.Prices).ToPageAsync(pageSettings));
        }

        [HttpGet("getByBarcode/{barcode}")]
        public async Task<IActionResult> GetByBarcode(string barcode)
        {
            var goodEntity = await _context.Goods
                                .Include(x => x.Prices)
                                .ThenInclude(x => x.Store)
                                .ThenInclude(x => x.Company)
                                .FirstOrDefaultAsync(x => x.Barcode == barcode);
            if (goodEntity == null)
            {
                return NotFound();
            }

            var good = _mapper.Map<Good>(goodEntity);
            good.Prices = good.Prices.OrderBy(x => x.Price).ToList();
            return Ok(good);
        }

        [HttpPost("addOrUpdate")]
        public async Task<IActionResult> AddOrUpdate(GoodEntity goodEntity)
        {
            if (goodEntity.Barcode.Length != 13)
            {
                return BadRequest("Barcode must have 13 characters");
```

```
            }

            if (_context.Goods.Any(x => x.Barcode == goodEntity.Barcode))
            {
                _context.Goods.Update(goodEntity);
            }
            else
            {
                await _context.Goods.AddAsync(goodEntity);
            }

            await _context.SaveChangesAsync();
            return NoContent();
        }

        [HttpGet("updateDatabase")]
        public IActionResult UpdateDatabase()
        {
            _schedulingService.Startup();
            return Ok();
        }
    }


[ApiController]
    [Route("[controller]")]
    public class StoreController : ControllerBase
    {
        private readonly AppDbContext _context;
        private readonly IMapper _mapper;

        public StoreController(AppDbContext context, IMapper mapper)
        {
            _context = context;
            _mapper = mapper;
        }

        [HttpPost("getPage")]
        public async Task<IActionResult> Get(PageSettings pageSettings)
        {
            return Ok(await _context.Stores.ToPageAsync(pageSettings));
        }
```

```
        [HttpGet("getCcompanies")]
        public async Task<IActionResult> Get()
        {
            var enumerable = await _context.Companies.Include(x =>
x.Stores).ToArrayAsync();
            return Ok(_mapper.Map<IEnumerable<Company>>(enumerable));
        }

        [HttpGet("getByRetailName/{retailName}")]
        public IActionResult Get(StoreRetailName retailName)
        {
            var store = _context.Stores.Where(x => x.RetailName == retailName);
            if (!store.Any())
            {
                return NotFound();
            }

            return Ok(store);
        }

        [HttpGet("getById/{id}")]
        public async Task<IActionResult> GetByBarcode(int id)
        {
            var store = await _context.Stores.FirstOrDefaultAsync(x => x.Id == id);
            if (store == null)
            {
                return NotFound();
            }

            return Ok(store);
        }
    }


[ApiController]
    [Route("[controller]")]
    public class PriceController : ControllerBase
    {
        private readonly AppDbContext _context;
        private readonly ILogger<PriceController> _logger;
```

```
        public PriceController(AppDbContext context,
                               ILogger<PriceController> logger)
        {
            _context = context;
            _logger = logger;
        }

        [HttpPost("setOrUpdatePrice")]
        public async Task<IActionResult> SetPricesByBarcode(PriceEntity
priceEntity)
        {
            if (_context.Prices.Any(x => x.GoodBarcode ==
priceEntity.GoodBarcode && x.StoreId == priceEntity.StoreId))
            {
                _context.Prices.Update(priceEntity);
            }
            else
            {
                await _context.Prices.AddAsync(priceEntity);
            }

            await _context.SaveChangesAsync();
            return NoContent();
        }

        [HttpGet("sendMailToFixPrice/{barcode}/{storeId}")]
        public async Task<IActionResult> SendMailToFixPrice(string barcode, int
storeId)
        {
            var price = await _context.Prices.FirstOrDefaultAsync(x => x.StoreId ==
storeId && x.GoodBarcode == barcode);
            if (price == null)
            {
                return NotFound();
            }

            SendEmail("somemail@mail.ru", "Please change price in your store",
$"Barcode is {barcode}, price on your website is {price.Price}.");

            return NoContent();
        }
```

```
        private void SendEmail(string email, string subject, string message)
        {
            _logger.LogInformation($"{subject}\n{message}");
        }
    }
```

Contracts.dll

```
public class Address
    {
        public string City { get; set; }

        public string Street { get; set; }

        public string Building { get; set; }

        public override string ToString()
        {
            return $"{City} {Street} {Building}";
        }
    }
public class Company
    {
        public StoreRetailName RetailName { get; set; }

        public string ImageUrl { get; set; }

        public IEnumerable<Store> Stores { get; set; }
    }
public class Good
    {
        public string Barcode { get; set; }

        public string Name { get; set; }

        public string ImageUrl { get; set; }

        public IList<PriceModel> Prices { get; set; }

        public IList<Company> Companies { get; set; }
    }
public class PageSettings
```

```
    {
        [Range(1, int.MaxValue)]
        public int CurrentPage { get; set; }

        [Range(1, int.MaxValue)]
        public int PageSize { get; set; }
    }
public class PriceModel
    {
        public decimal Price { get; set; }

        public string GoodBarcode { get; set; }

        public Store Store { get; set; }
    }
public class ResultPage<T>
    {
        public ResultPage()
        {
            Results = new List<T>();
        }

        public int CurrentPage { get; set; }

        public int PageCount { get; set; }

        public int PageSize { get; set; }

        public int RowCount { get; set; }

        public int FirstRowOnPage => ((CurrentPage - 1) * PageSize) + 1;

        public int LastRowOnPage => Math.Min(CurrentPage * PageSize,
RowCount);

        public IList<T> Results { get; set; }
    }
public class Store
    {
        public int Id { get; set; }
```

```
        public string Name { get; set; }

        public StoreRetailName RetailName { get; set; }

        public Address Address { get; set; }

        public string Coords { get; set; }

        public DateTime LastUpdate { get; set; }
    }
public enum StoreRetailName
    {
        Auchan,
        Novus,
        Furshet,
        Metro,
        Tavriav,
        Megamarket,
        Vostorg
    }
```

App.dll

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:forms="clr-
namespace:FFImageLoading.Svg.Forms;assembly=FFImageLoading.Svg.Forms
"
             mc:Ignorable="d"
             NavigationPage.HasNavigationBar="False"
             x:Class="App111.Pages.CompaniesPage">
    <ContentPage.Content>
        <StackLayout>
            <ScrollView Padding="10">
                <StackLayout>
                    <ListView HasUnevenRows="True"
                              ItemsSource="{Binding Companies}">
                        <ListView.ItemTemplate>
                            <DataTemplate>
```

```
                        <ViewCell>
                            <StackLayout>
                                <forms:SvgCachedImage HeightRequest="40"
Source="{Binding ImageUrl}"
                                                      VerticalOptions="Center"
                                                      Margin="10">
                                    <forms:SvgCachedImage.GestureRecognizers>
                                        <TapGestureRecognizer Tapped="OpenStores"
                                                              CommandParameter="{Binding
RetailName}"
                                                              NumberOfTapsRequired="2" />
                                    </forms:SvgCachedImage.GestureRecognizers>
                                </forms:SvgCachedImage>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </StackLayout>
    </ScrollView>
    <Button Text="Get Back" Clicked="GetMain_OnClicked"
            HorizontalOptions="Center" FontSize="15"
            WidthRequest="200" BackgroundColor="Coral"
CornerRadius="20"
            BorderColor="Black" BorderWidth="1"
            TextColor="GhostWhite"
            Margin="0,0,0,20"></Button>
  </StackLayout>
 </ContentPage.Content>
</ContentPage>


[XamlCompilation(XamlCompilationOptions.Compile)]
 public partial class CompaniesPage : ContentPage
 {
  public IEnumerable<Company> Companies { get; }

  public CompaniesPage(IList<Company> companies)
  {
   for (var i = 0; i < companies.Count; i++)
   {
    companies[i].ImageUrl = companies[i].RetailName + ".jpg";
```

```
            }

            Companies = companies;
            InitializeComponent();
            BindingContext = this;
        }

        private async void GetMain_OnClicked(object sender, EventArgs e)
        {
            await Navigation.PopToRootAsync();
        }

        private async void OpenStores(object sender, EventArgs e)
        {
            var retailName =
(StoreRetailName)Enum.Parse(typeof(StoreRetailName),
((TappedEventArgs)e).Parameter.ToString());
            var y = Companies.First(x => x.RetailName ==
retailName).Stores.Select(x => $"{x.Name} \n{x.Address}\n");
            await DisplayAlert("Stores", string.Join("\n", y), "Ok");
        }
    }
```

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:forms="clr-
namespace:ZXing.Net.Mobile.Forms;assembly=ZXing.Net.Mobile.Forms"
             xmlns:common="clr-
namespace:ZXing.Common;assembly=zxing.portable"
             xmlns:svg="clr-
namespace:FFImageLoading.Svg.Forms;assembly=FFImageLoading.Svg.Forms
"
             mc:Ignorable="d"
             x:Class="App111.Pages.GoodPage"
             NavigationPage.HasNavigationBar="False"
             BackgroundColor="DeepSkyBlue">
    <StackLayout BackgroundColor="White">
        <StackLayout>
            <Image Source="{Binding Good.ImageUrl}"
```

```
                HorizontalOptions="Center"
                HeightRequest="150"></Image>
            <forms:ZXingBarcodeImageView IsVisible="True"
BarcodeFormat="EAN_13"
                                HorizontalOptions="CenterAndExpand"
                                BarcodeValue="{Binding Good.Barcode}">
              <forms:ZXingBarcodeImageView.BarcodeOptions>
                <common:EncodingOptions  Height="100" Width="600"
PureBarcode="True" />
              </forms:ZXingBarcodeImageView.BarcodeOptions>
            </forms:ZXingBarcodeImageView>
            <Label Text="{Binding Good.Barcode}" FontSize="10"
TextColor="Black" HorizontalTextAlignment="Center" ></Label>
            <Label Padding="10" Text="{Binding Good.Name}" FontSize="20"
TextColor="Black" HorizontalTextAlignment="Center" />
        </StackLayout>
        <ScrollView Padding="10">
            <StackLayout>
              <ListView HasUnevenRows="True"
                    ItemsSource="{Binding PriceViewModel.PriceView}">
                <ListView.ItemTemplate>
                  <DataTemplate>
                    <ViewCell>
                      <StackLayout>
                        <Grid Margin="10">
                          <Grid.ColumnDefinitions>
                            <ColumnDefinition />
                            <ColumnDefinition />
                          </Grid.ColumnDefinitions>
                          <svg:SvgCachedImage HeightRequest="40"
Source="{Binding Item2}" Grid.Column="0" VerticalOptions="Center">
                            <svg:SvgCachedImage.GestureRecognizers>
                              <TapGestureRecognizer Tapped="OpenStores"
                                                    CommandParameter="{Binding
Item3}"
                                                    NumberOfTapsRequired="2" />
                            </svg:SvgCachedImage.GestureRecognizers>
                          </svg:SvgCachedImage>
                          <Label Text="{Binding Item1}" TextColor="Black"
HorizontalTextAlignment="Center"
                                 Grid.Column="1"
VerticalTextAlignment="Center" />
```

```
                        </Grid>
                    </StackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</StackLayout>
</ScrollView>
<Button Text="Get Back" Clicked="GetMain_OnClicked"
        HorizontalOptions="Center" FontSize="15"
        WidthRequest="200" BackgroundColor="Coral" CornerRadius="20"
        BorderColor="Black" BorderWidth="1"
        TextColor="GhostWhite"
        Margin="0,0,0,20"></Button>
</StackLayout>
</ContentPage>
```

```
[XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class GoodPage : ContentPage
    {
        public Good Good { get; set; }

        public PriceViewModel PriceViewModel { get; set; }

        public GoodPage(Good good)
        {
            InitializeComponent();
            PriceViewModel = new PriceViewModel
            {
                PriceView = new List<Tuple<string, string, string>>()
            };
            for (var i = 0; i < good.Companies.Count(); i++)
            {
                good.Companies[i].ImageUrl = good.Companies[i].RetailName +
".jpg";
            }
            foreach (StoreRetailName variable in
Enum.GetValues(typeof(StoreRetailName)))
            {
                var priceModels = good.Prices.Where(x => x.Store.RetailName ==
variable).DistinctBy(x => x.Price).ToArray();
                if (priceModels.Any())
                {
```

```
                var retailName = priceModels.First().Store.RetailName;
                var img = good.Companies.FirstOrDefault(x => x.RetailName ==
retailName)?.ImageUrl;
                PriceViewModel.PriceView.Add(priceModels.Length >= 2
                                        ? new Tuple<string, string,
string>($"{priceModels.First().Price} - {priceModels.Last().Price} грн", img,
variable.ToString())
                                        : new Tuple<string, string,
string>(priceModels.First().Price.ToString(CultureInfo.CurrentCulture) + "
грн", img, variable.ToString()));
            }
        }

        Good = good;
        BindingContext = this;
    }

    private async void GetMain_OnClicked(object sender, EventArgs e)
    {
        await ExceptionHandler.Handle(async () => await
Navigation.PopToRootAsync(), this);
    }

    private async void OpenStores(object sender, EventArgs e)
    {
        var retailName =
(StoreRetailName)Enum.Parse(typeof(StoreRetailName),
((TappedEventArgs)e).Parameter.ToString());
        var y = Good.Prices
                .Where(x => x.Store.RetailName == retailName)
                .Select(x => $"{x.Store.Name} \n{x.Store.Address}\n {x.Price}
грн\n");
        await DisplayAlert("Stores", string.Join("\n", y), "Ok");
    }
}


<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
```

```
            xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
            mc:Ignorable="d"
            x:Class="App111.Pages.MainPage"
            NavigationPage.HasNavigationBar="False">
    <StackLayout BackgroundColor="GhostWhite">
        <Label Text="Check price" HorizontalOptions="Center" FontSize="40"
TextColor="Black" Margin="0,40,0,40"></Label>
        <Button BackgroundColor="Coral" Text="Our stores"
WidthRequest="100" HeightRequest="100" CornerRadius="50"
                TextColor="GhostWhite"  BorderColor="Black" BorderWidth="1"
HorizontalOptions="Center"
                Clicked="OpenCompaniesList_OnClicked"></Button>
        <Button Margin="0,370,0,0" Text="Search by barcode"
Clicked="ScannerButton_OnClicked"
                HorizontalOptions="Center" FontSize="15"
                WidthRequest="200" BackgroundColor="Coral" CornerRadius="20"
BorderColor="Black" BorderWidth="1"
                TextColor="GhostWhite">
        </Button>
    </StackLayout>
</ContentPage>


[XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            BindingContext = this;
        }

        private async void ScannerButton_OnClicked(object sender, EventArgs e)
        {
            await ExceptionHandler.Handle(async () =>
                                    {
                                        await Navigation.PushAsync(new ScannerPage());
                                    },
                                    this);
        }
```

```
        private async void OpenCompaniesList_OnClicked(object sender,
EventArgs e)
        {
            await ExceptionHandler.Handle(async () =>
                                    {
                                        using (var client = new HttpClient())
                                        {
                                            var content = await client.GetAsync(new
Uri("http://rutakamekiar-001-site1.ftempurl.com/Store/getCcompanies"));
                                            content.EnsureSuccessStatusCode();
                                            var json = await
content.Content.ReadAsStringAsync();
                                            var companies =
JsonConvert.DeserializeObject<IList<Company>>(json);
                                            await Navigation.PushAsync(new
CompaniesPage(companies));
                                        }
                                    },
                                    this);
        }
    }


<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:d="http://xamarin.com/schemas/2014/forms/design"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             xmlns:zxing="clr-
namespace:ZXing.Net.Mobile.Forms;assembly=ZXing.Net.Mobile.Forms"
             mc:Ignorable="d"
             x:Class="App111.Pages.ScannerPage"
             NavigationPage.HasNavigationBar="False">
    <StackLayout>
        <Grid HorizontalOptions="FillAndExpand"
VerticalOptions="FillAndExpand" AbsoluteLayout.LayoutFlags="All"
AbsoluteLayout.LayoutBounds="0,0,1,1">
            <Grid.RowDefinitions>
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <zxing:ZXingScannerView Grid.Row="0" x:Name="zxing"
```

```
                    IsScanning="true" HeightRequest="300"

OnScanResult="ZXingScannerView_OnOnScanResult" />
        <zxing:ZXingDefaultOverlay Grid.Row="0"
                                   ShowFlashButton="False"
                                   Opacity="0.9" />
        <ActivityIndicator Grid.Row="0"
                           Scale="2"
                           Color="Coral"
                           x:Name="ActivityIndicator"
                           IsEnabled="False"
                           VerticalOptions="CenterAndExpand"
                           HorizontalOptions="CenterAndExpand" />
    </Grid>
  </StackLayout>
</ContentPage>


[DesignTimeVisible(false)]
  public partial class ScannerPage : ContentPage
  {
    public ScannerPage()
    {
      InitializeComponent();
    }

    private void ZXingScannerView_OnOnScanResult(Result result)
    {
      zxing.IsAnalyzing = false;
      Device.BeginInvokeOnMainThread(async () =>
      {
        var barcode = string.Empty;
        await ExceptionHandler.Handle(async () =>
                                      {
                                        ActivityIndicator.IsRunning = true;
                                        using (var client = new HttpClient())
                                        {
                                          var content = await client.GetAsync(new
Uri("http://rutakamekiar-001-site1.ftempurl.com/Good/getByBarcode/"
                                                                              + result.Text));
                                          barcode = result.Text;
```

```
                                    if (content.StatusCode ==
HttpStatusCode.NotFound)
                                    {
                                        await DisplayAlert("Error",
                                                $"Good with barcode
{result.Text} not found.",
                                                "Ok");
                                    }
                                    else if (content.IsSuccessStatusCode)
                                    {
                                        var json = await
content.Content.ReadAsStringAsync();
                                        var good =
JsonConvert.DeserializeObject<Good>(json);
                                        await Navigation.PushAsync(new
GoodPage(good));
                                    }
                                    else
                                    {
                                        var message = await
content.Content.ReadAsStringAsync();
                                        throw new HttpRequestException($"Failed
with status {content.StatusCode}, reason {content.ReasonPhrase}, message
{message}");
                                    }

                                    ActivityIndicator.IsRunning = false;
                                    zxing.IsAnalyzing = true;
                                }
                            },
                            this,
                            $"Failed with barcode {barcode}");
            });
        }
    }
```