МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних
інформаційних систем, д-р. техн. наук, проф.
_____Ю. П. Кондратенко
«____»_____2023 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

СИСТЕМА ФІНАНСОВОГО КОНТРОЛЮ ВЛАСНИХ КОШТІВ ШЛЯХОМ ЗБЕРІГАННЯ ЧЕКІВ В ЕЛЕКТРОННОМУ ВИГЛЯДІ

Спеціальність 122 «Комп'ютерні науки»

122 – БКР – 402.21910212

Виконав студент 4-го курсу, групи 402
_____ В. С. Кириленко
«19» червня 2023 р.

Керівник: канд. фіз.-мат. наук, доцент
_____І. В. Кулаковська
«19» червня 2023 р.

Миколаїв – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Рівень вищої освіти **бакалавр**
Спеціальність **122 «Комп'ютерні науки»**
(шифр і назва)
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем, д-р. техн. наук, проф.
_____ Ю. П. Кондратенко
«23» листопада 2022 р.

З А В Д А Н Н Я
на виконання кваліфікаційної роботи

Видано студенту групи 402 факультету комп'ютерних наук Кириленку Владиславу Сергійовичу.

1. Тема кваліфікаційної роботи «Система фінансового контролю власних коштів шляхом зберігання чеків в електронному вигляді».

Керівник роботи Кулаковська Інесса Василівна, канд. фіз.-мат. наук, доцент

Затв. наказом Ректора ЧНУ ім. Петра Могили від «17» березня 2023 р. № 59

2. Строк представлення кваліфікаційної роботи студентом «26» червня 2023 р.

3. Вхідні (початкові) дані до роботи: статистика проведених безготівкових та готівкових операцій в Україні; прогнози експертів щодо збільшення чи зменшення певних видів грошових операцій

Очікуваний результат: готовий застосунок для створення, збереження та перегляду електронних чеків у смартфоні користувача

4. Перелік питань, що підлягають розробці (зміст пояснювальної записки):

– дослідження методів розробки мобільних застосунків на базі операційної системи Android;

– порівняння мов програмування, фреймворків та бібліотек, за допомогою яких створюються мобільні застосунки;

– дослідити актуальність теми для проєкта;

– проаналізувати існуючі аналогічні проєкти;

– створення мобільного застосунку.

5. Перелік графічного матеріалу: презентація.

6. Завдання до спеціальної частини: «Вимоги щодо використання екранних пристроїв»

7. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис
Спеціальна частина з охорони праці	Боженко А. Л., викладач	

Керівник роботи __канд. фіз.-мат. наук І. В. Кулаковська_____
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ___Кириленко В. С._____
(прізвище та ініціали)

_____
(підпис)

Дата видачі завдання «_23_» _____листопада____ 2022 р.

КАЛЕНДАРНИЙ ПЛАН
виконання бакалаврської кваліфікаційної роботи

Тема: Система фінансового контролю власних коштів шляхом зберігання чеків в електронному вигляді

№	Найменування роботи	Початок	Закінчення	Примітки
1	Подання заяви на затвердження теми та керівників БКР	27.10.2022	27.10.2022	Виконано
2	Отримання завдання на виконання БКР	20.11.2022	20.11.2022	Виконано
3	Складання календарного плану роботи на весь період виконання БКР	03.12.2022	03.12.2022	Виконано
4	Отримання завдання на переддипломну практику	29.04.2023	29.04.2023	Виконано
5	Проходження переддипломної практики, збір та аналіз матеріалів до БКР	01.05.2023	14.05.2022	Виконано
6	Розробка звіту з переддипломної практики	15.05.2023	17.05.2023	Виконано
7	Виконання БКР: аналіз аналогічних додатків, огляд існуючих технологій, побудова моделей та зв’язків між ними, розробка застосунків (можна більш детально, або в декілька пунктів)	15.05.2023	19.06.2023	Виконано
8	Попередній захист БКР на засіданні комісії кафедри	29.05.2023	30.05.2023	Виконано
9	Доробка та остаточне оформлення БКР	02.06.2023	19.06.2023	Виконано
10	Подання БКР рецензенту	15.06.2023	17.06.2023	Виконано
11	Подання БКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	19.06.2023	22.06.2023	
12	Захист БКР перед екзаменаційною комісією (ЕК)	26.06.2023	29.06.2023	

Розробив студент Кириленко В. С. _____
(прізвище та ініціали) (*підпис*)

Керівник роботи канд. фіз.-мат. наук Кулаковська І. В. _____
(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«23» листопада 2022 р.

АНОТАЦІЯ

бакалаврської кваліфікаційної роботи студента групи 402 ЧНУ ім. Петра Могили

Кириленка Владислава Сергійовича

Тема: «Система фінансового контролю власних коштів шляхом зберігання чеків в електронному вигляді»

Актуальність: рішення даної проблеми було б дуже корисним для сучасного суспільства, що допомогло б зекономити фінансові ресурси, допомогти екології менше страждати від вирубки лісів та продовжити шлях Міністерства цифрової трансформації України, який можна охарактеризувати так: «Все повинно бути під рукою і бути прозорим».

Об’єкт роботи: процес створення застосунку під операційну систему Android, використовуючи мову програмування C# за допомогою фреймворку Xamarin.Forms.

Предмет роботи: засоби для створення мобільного програмного застосунку для створення та дослідження можливостей щодо збереження електронних чеків.

Мета роботи: надання можливості створення та збереження електронних чеків на девайсі на платформі Android; дослідження проблеми впровадження такого рішення на законодавчому рівні.

У першому розділі проводиться аналіз необхідності імплементації представленого вирішення проблеми, порівняння поточного проєкту з аналогами та представлення технічного завдання. У другому розділі – опис зовнішніх інструментів, що використовуються у проєкті та їх зв’язок із проєктом, також зображуються моделі проєкту. У останньому розділі представлено програмно реалізацію мобільного та API застосунків.

Бакалаврська кваліфікаційна робота містить 82 сторінки, 0 таблиць, 35 рисунків, 32 посилань, 0 додатків.

Ключові слова: база даних, мобільний застосунок, API, Xamarin, Android, електронний чек, C#.

ABSTRACT

for bachelor's qualification work of a student of 402 group at Petro Mohyla Black Sea National University

Kyrylenko Vladyslav Serhiyovich

Topic: «A system of financial control of own funds by storing checks in electronic form»

Relevance: the solution to this problem would be very useful for modern society, which would help save financial resources, help the environment to suffer less from deforestation and continue the path of the Ministry of Digital Transformation of Ukraine, which can be described as follows: "Everything should be at hand and be transparent."

The object of the work is the process of creating an application for the Android operating system using the C# programming language using the Xamarin.Forms framework.

The subject of the work is tools to create a mobile software application to create and explore electronic check storage options.

The purpose of the work is creation of a mobile application for creating and saving electronic receipts on an Android device; study of the problem of implementation of such a decision at the legislative level.

The first section analyzes the need to implement the presented solution, compares the current project with similar ones and presents the technical task. In the second section - a description of the external tools used in the project and their connection with the project, the project models are also presented. The last section presents the software implementation of mobile and API applications.

Bachelor qualification work contains 82 pages, 0 tables, 35 figures, 32 references, 0 appendices.

Keywords: database, mobile software application, API, Xamarin, Android, electronic receipts, C#.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД – база даних

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

СУБД – система управління базами даних

API – application programming interface

DI – Dependency injection

HTTP – Hypertext Transfer Protocol

JSON – JavaScript object notation

UI – user interface

UX – user experience

ВСТУП

Під час написання бакалаврської кваліфікаційної роботи було проаналізовано та досліджено певні поставлені питання та проблеми. Враховуючи розвиток сучасних технологій, реалізація рішення проблеми зі зберіганням електронних чеків не є складною та довгою. Але при цьому, спостерігається ряд інших проблем, не пов’язаних безпосередньо зі сферою комп’ютерних наук, такі як наявність обладнання, бажання підприємців співпрацювати з приватними/державними компаніями по розробці програмного забезпечення (ПЗ), а також довіра своїх даних та вже наявних розробок, особливостей підприємства розробникам для подальшого об’єднання із новоствореним застосунком, об’єктивність заміни паперових чеків на електронні, враховуючи рівень діджиталізації країни та зацікавленість населення в переході на постійне або часте користування електронними чеками, складнощі щодо зміни поточного законодавства[1], наявність ресурсів задля досягнення поставленої мети тощо. Враховуючи вищезазначене, можна дійти висновку, що реалізація такої ідеї була б дуже корисною для сучасного суспільства, що допомогло б зекономити певні ресурси, допомогти екології менше страждати від вирубки лісів та продовжити шлях Міністерства цифрової трансформації України, який можна коротко охарактеризувати так: «Все повинно бути під рукою і бути прозорим».

Мета даної роботи полягає у тому, запропонувати рішення проблеми централізованого зберігання електронних чеків в одному застосунку, при цьому ще й відповідності застосунку законодавству України, а також показати прототип застосунку для вирішення поставленої задачі.

Об’єкт роботи: процес створення застосунку під операційну систему Android, використовуючи мову програмування C# за допомогою фреймворку Xamarin.Forms.

Предмет роботи: засоби для створення мобільного програмного застосунку для створення та дослідження можливостей щодо збереження електронних чеків.

Так як створення застосунку – справа досить кропітка і потребує знань з різних галузей однієї сфери (а іноді, для досягнення більших результатів, ще й з інших сфер, професій), то в результаті створюється міцний зв'язок із більш творчими (але все ж технічними професіями) як UI/UX-дизайнер, що дозволяє створити більш привабливе та приємне оку користувача дизайн застосунку, що, в свою чергу, підвищує користувацьку аудиторію; маркетолог – для поширення застосунку серед цільової аудиторії та інші.

Задля досягнення поставлених цілей було використано інструменти мобільної розробки, такі як мова програмування C# та мобільний фреймворк від Microsoft – Xamarin.Forms, для зберігання даних та операцій над ними було використано MS SQL Server.

Завдання роботи:

– дослідження методів розробки мобільних застосунків на базі операційної системи Android;

– порівняння мов програмування, фреймворків та бібліотек, за допомогою яких створюються мобільні застосунки;

– дослідити актуальність теми для проєкта;

– проаналізувати існуючі аналогічні проєкти;

– створення мобільного застосунку.

1 АНАЛІЗ НЕОБХІДНОСТІ ІМПЛЕМЕНТАЦІЇ. ПРОВЕДЕННЯ АНАЛОГІЇ. ПОСТАНОВКА ЗАВДАННЯ

1.1 Дослідження та аналіз статистики щодо необхідності вирішення проблеми

В першу чергу, перед тим, як вирішувати певну проблеми або надавати відповідь на певне питання, спочатку треба з’ясувати, чи варте воно того, або, більш офіційною мовою – з’ясувати актуальність даної проблеми. Для цього, треба звернутися до джерел статистики. Найкраще для цього підійде статистика, що відтворює рівень діджиталізації громадян України, що значитиме, чи варто впроваджувати нові технології в цій сфері.

Стрімка діджиталізація нашої країни та поява подібних застосунків, а також зростання кількості покупок, здійснених безготівковим шляхом (оплата через Інтернет, переказ на особисті картки, оплата через термінали, зокрема смартфони з вбудованими NFC-модулями), все частіше ставлять це питання на порядок денний. Застосунки для онлайн-банкінгу, такі як Приват24, monobank та розроблений державою багатофункціональний застосунок Дія, набувають все більшого значення, користувачі цих застосунків намагаються автоматизувати якомога більше функцій і перенести їх на свої смартфони для виконання різних видів операцій. Вони намагаються скоротити час і ресурси, що витрачаються на безготівкові платежі, подачу і підготовку документів, масове зберігання документів, швидке отримання довідок та інших документів, подачу податкових декларацій, сплату податків тощо. За останніми опублікованими даними, станом на липень 2022 року кількість транзакцій (безготівкових зняття коштів), здійснених в Україні та за її межами з використанням платіжних карток, емітованих українськими банками, сягнула 637 мільйонів, загальною вартістю 573,7 мільярда гривень. Порівняно з довоєнним січнем 2022 року кількість транзакцій дещо зменшилася (на 3,4%), а сума операцій зросла приблизно на третину (на 31,5%). Під час війни українці продовжували надавати перевагу безготівковим операціям з

платіжними картками. Так, у травні 2022 року частка безготівкових операцій з платіжними картками за вартістю становила майже 69% від загального обсягу операцій з платіжними картками (у січні 2022 року - 63%). Водночас частка за кількістю залишилася майже незмінною - 92% (у січні 2022 року - 91%). Іншими словами, 92 з кожних 100 операцій за картками є безготівковими. Аналізуючи розподіл безготівкових операцій за типами, можна побачити, що, як і раніше, більшість карткових операцій у травні 2022 року здійснювалася готівкою:

– у кількісному вираженні 53,9% (317 млн) транзакцій було здійснено в торговельній мережі. Це майже чверть усіх безготівкових операцій (24,5%);

– перекази з картки на картку склали 57% (або 225,2 млрд грн). Цей показник становив 14%.

Незважаючи на зменшення загальної кількості безконтактних та токенізованих платіжних карток (-6% та -15% відповідно), їх популярність серед українців залишається дуже високою. На безконтактні та токенізовані платіжні картки припадає 60% діючих карток (58% у січні цього року). Загалом, кожна шоста платіжна картка є токенізованою (порівняно з кожною сьомою на початку січня), а кожна друга - безконтактною (так само, як і на початку січня). Як наслідок, менше п'ятої частини транзакцій у торговельній мережі було здійснено шляхом фізичного зчитування даних з картки (18,2% за сумою та 16% за кількістю). Решта транзакцій були безконтактними (з використанням таких інструментів, як безконтактні картки або смартфони). Загальна вартість цих транзакцій у травні цього року склала 79,2 млрд грн.[2]

Отже, виходячи з даної статистики, можна помітити, як українці активно замінюють готівкові операції та безготівкові операції за допомогою картки на безготівкову оплату за допомогою смартфонів та інших гаджетів з модулем NFC. Можна тільки уявити, скільки дерев можна зберегти, замінивши паперові чеки на електронні, що централізовано зберігатимуться в смартфоні, які можна буде використовувати так само, як і паперові. Такі чеки важко втратити, достатньо було б відкрити застосунок, в якому вони всі зберігаються. Але важливо створити

централізований застосунок, наприклад «Дія», що вважався б єдиним легітимним застосунком для зберігання будь-яких чеків покупок на території України. Також не треба забувати про підготовку законодавчої бази, яка б спонукала підприємців переходити саме на можливість видачі електронних чеків (наприклад, запровадити зменшення податків для таких підприємців, або навпаки – збільшення податків для тих, хто досі відмовляється від впровадження нових технологій), а також на дозвіл покупців використовувати електронні чеки замість паперових, наприклад при поверненні/обміні товару тощо.

Отже, можна дійти до висновків, що громадяни нашої держави показують високий рівень готовності до даних технологій, адже популярність готівкових операцій стає з кожним роком нижче, при чому з кожним роком приріст безготівкових операцій йде стрімко вгору, особливо з використанням пристроїв з модулями NFC, найчастіше – це смартфони, що для нас і є цільовою аудиторію – приваблення до користування саме володарів смартфонів, які часто використовують безконтактні методі оплати товару в магазинах.

Про це також каже спільне дослідження агентства мобільного маркетингу LEAD9 та Київського міжнародного інституту соціології (КМІС) показало як українці користуються смартфонами та мобільними застосунками. За їх даними, 85% українців у віці 18-30 років користуються смартфонами з сенсорними екранами. Порівнюючи з попередніми роками, частка проникнення смартфонів в Україні за останні два роки виросла на 26%. Активніше смартфони використовують жителі великих і середніх міст - 54-55%. В маленьких містах таких близько 43%, а в селах - 32%. При цьому 91% українських власників сенсорних смартфонів користується мобільними застосунками[3].

Що стосується покупок, то інформацію про товари в 50% випадків люди шукали через смартфон, а 43% оформлювали покупку теж за допомогою мобільних гаджетів.

1.2 Дослідження поточної ситуації шляхом порівняння аналогічних застосунків

Якщо проаналізувати поточну ситуацію, актуальну на травень 2023 року, деякі магазини (наприклад, мережа магазинів краси «EVA») створили власні застосунки, які дозволяють зберігати чеки в телефоні і вважаються рівними паперовим, але лише в рамках цих магазинів.

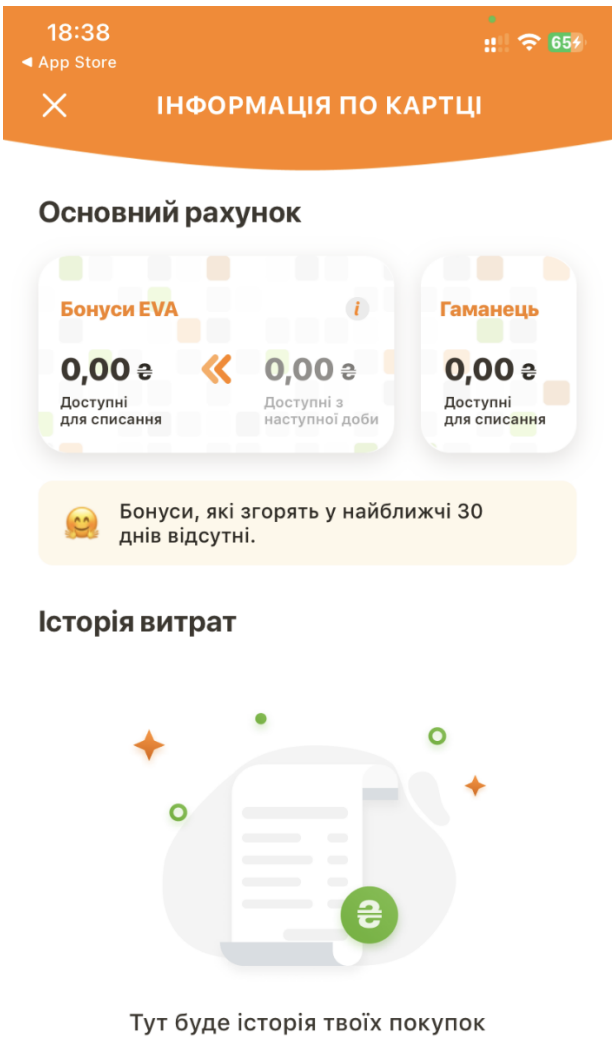

Рисунок 1.1 – Електронні чеки в застосунку «EVA» в розділі «Історія витрат»

Також було розглянуто аналогічний універсальний застосунок для зберігання чеків під назвою «Dext». Реєстрація в даному застосунку відбувається за допомогою авторизації через Apple ID (для користувачів пристроїв на базі операційної системи iOS), Google Account та звичайну реєстрацію через електронну пошту та пароль[4]

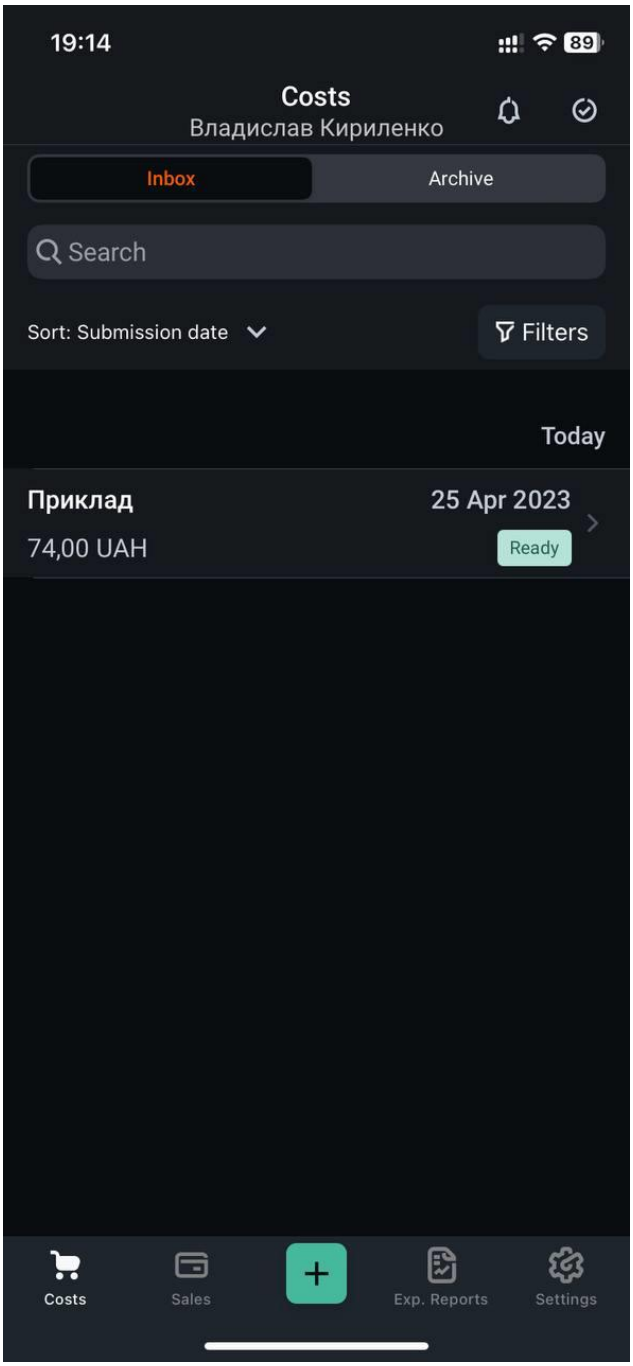

Рисунок 1.2 – Зберігання чеків в застосунку «Dext»

Рисунок 1.3 – Операція сканування чеку

Але все ж, даний застосунок далекий від цілі, яку прагне досягти проєкт практиканта, адже вищезазначений застосунок зберігає паперові чеки шляхом їх сканування через камеру смартфона, а також не вважається легітимною заміною паперових чеків. Тому слід розглянути і дослідити концепцію проєкту, який міг би частково або повністю вирішити дану проблему.

Переходячи до наступного завдання, а саме вивчення методів розробки, мов програмування, фреймворків для створення мобільного застосунку, було досліджено, що наразі мобільні застосунки можна створити за допомогою таких мов програмування як Java, Javascript, Swift, Kotlin, Dart, C#/.NET та інші. Найголовнішим і, напевно, єдиним чинником, який вплинув на вибір мови програмування, був поточний рівень знань в певній мові програмування.

1.3 Вибір та порівняння сучасних технологій для розробки мобільних застосунків

Наступним питанням було аналізування технологій для написання мобільних застосунків на вибраній мові. Для цього, треба звернутися до певних форумів, ресурсів, передових розробників і подивитися, що зараз є популярним і практичним. Проаналізувавши офіційну документацію розробки мобільних застосунків[5], авторські блоги[6-8], форуми розробників різного рівня та різної сфери діяльності[9], курси[10], книги[11], та збірки відео[12], а також виходячи з власного досвіду та набору знань, було вирішено обрати інструменти розробки, пов’язані з C#/.NET. Серед них найпопулярніші це Xamarin.Forms та .NET MAUI. Характерно те, що остання технологія є більш покращеною та новою версією технології Xamarin.Forms. Але, враховуючи наявність навчальних матеріалів, підтримку даної технології розробниками та популярність, було обрано більш перевірений, хоча й більш старий варіант – Xamarin.Forms.

Для початку розробки за допомогою даного фреймворку, було прийнято рішення розпочати вивчення теоретичного матеріалу за допомогою офіційної документації фреймворку[13], простих прикладів реалізації певних елементів на офіційному репозиторії розробника Github[14], офіційного форума Xamarin.Forms[15], офіційного блогу Xamarin.Forms[16], а також до певних курсів[17].

Технологія дуже комфортна для розробки тим, що розділяє дизайн застосунку та його функціонал шляхом розробки за допомогою XAML (спеціальна мова

розмітки для мов програмування з сімейства мов Microsoft) та C#. При цьому, вони можуть розробник може поєднувати ці два «інструменти» для розробки власного проєкту, тобто він може писати дизайн сторінки суто на C#, або суто на XAML, а може поєднувати це, використовуючи обидва варіанти. Хоча ця технологія підтримує розробку на двох популярних мобільних операційних системах – Android та iOS, було обрано для розробки саме Android, тому що ця операційна система має менше обмежень для розробника, а також може девайс, що має цю операційну систему, може бути приєднаний до ПК і налагоджений для того, щоб дивитися результат написаного коду в режимі онлайн. Режим налагодження на iPhone доступний лише при розробці на ПК не базі операційної системи macOS (обмеження внаслідок безпеки, обмін ключами безпеки може бути лише серед девайсів одного «сімейства»).

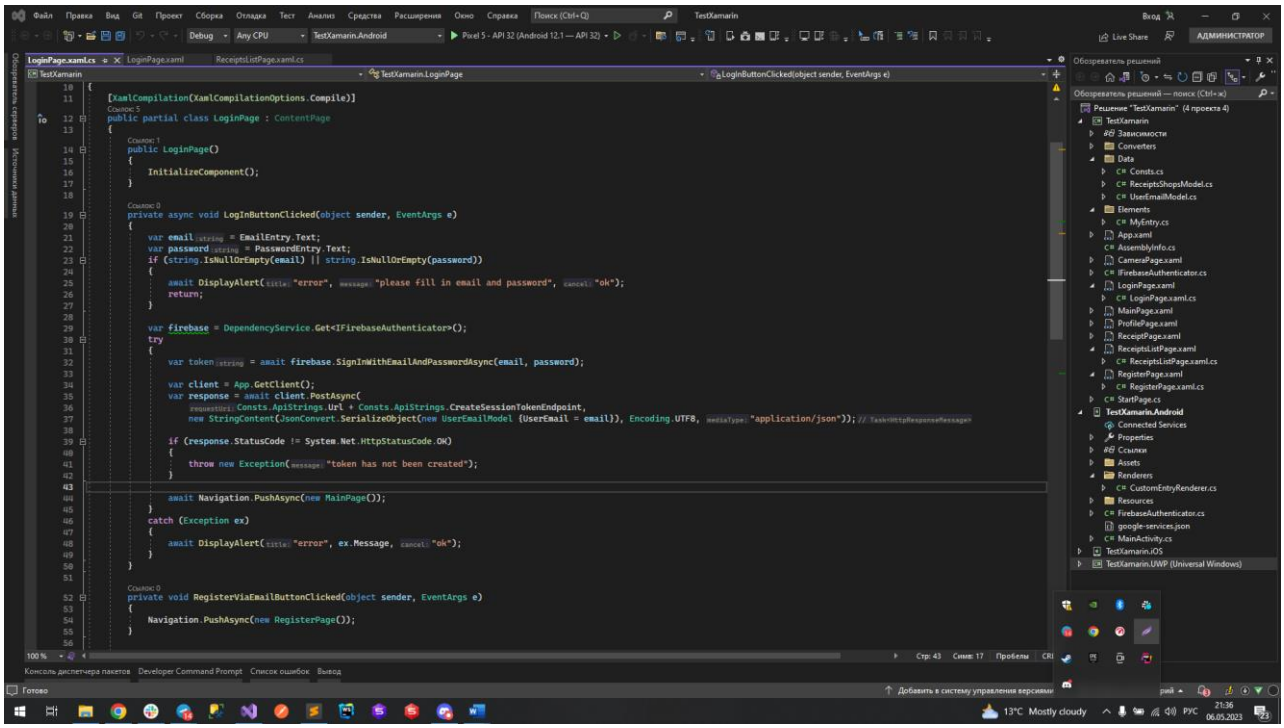

Рисунок 1.4 – Вибір середовища розробки мобільного застосунку – Visual Studio

Варто також додати, що вибір платформи розробки залежить не лише від

уподобань розробника чи команди розробників, але й від інших факторів. По-перше, від цільового користувача. Залежно від умов життя, таких як дохід і спосіб мислення, необхідно з'ясувати, для кого буде розроблятися застосунок, оскільки у них може бути багато пристроїв на базі необхідної операційної системи. Наприклад, в Японії, Північній Америці та деяких країнах Західної Європи, включаючи Великобританію, більшість населення користується продукцією Apple. Якщо ви плануєте зосередитися на цих регіонах, вам слід розглянути можливість розробки вашого застосунку на платформі iOS, щоб охопити цільову аудиторію. З іншого боку, користувачі Центральної та Східної Європи, Азії, Південної Америки та Африки здебільшого використовують пристрої на платформі Android.

Монетизація також є важливим фактором при виборі, оскільки причиною розробки застосунку зазвичай є заробіток для команди в обраному проекті. Залежно від операційної системи, на якій написано застосунок, команда проекту зароблятиме різну кількість грошей. З моменту свого заснування компанія Apple позиціонує себе як виробник висококласних пристроїв для багатих. Користувачі пристроїв Apple на платформі iOS за статистикою мають в чотири-п'ять разів більшу купівельну спроможність, ніж користувачі на платформі Android. Крім того, власники продуктів Apple набагато частіше купують застосунки та замовляють платні додаткові функції. Якщо ви плануєте продавати товари через свій застосунок, вам слід врахувати, що користувачі пристроїв на iOS переважають конкурентів на Android у цьому сегменті. Різниця становить близько 6%, що є вигідним, оскільки користувачі iOS більш схильні купувати продукти на своїх мобільних пристроях. Така різниця не є великою, але вона існує, і її слід враховувати при виборі платформи для майбутніх програм.

Але ще один факт пов'язаний з вартістю розробки застосунків: незалежно від того, чи працює він на iOS або Android, розробка мобільних застосунків коштує недешево. Тож для якої операційної системи варто замовити застосунок, щоб заощадити кошти та отримати очікувані результати?

Як правило, робота iOS-фахівців добре оплачується, а опублікувати готовий

продукт в App Store складніше і дорожче. Але оскільки iOS є дуже специфічною операційною системою і працює лише на обмеженій кількості пристроїв Apple, розробникам набагато простіше тестувати і перевіряти сумісність своїх застосунків. Це скорочує час розробки і, відповідно, знижує вартість професійних послуг. Розробка застосунків для Android має свої унікальні особливості через велику кількість різних пристроїв, що працюють на цій платформі. Розробникам необхідно адаптувати свої програми до різних варіантів екранів і специфікацій пристроїв, проводити тестування та оцінювати якість роботи своїх застосунків на пристроях різних виробників. Тому існує певний паритет з точки зору цінової політики при створенні мобільних застосунків для цих платформ. Однак ціни можуть дещо відрізнятися залежно від того, до якої студії чи компанії звертаються.[18]

Але, якщо говорити про більш прості речі, як-от розробка «для себе» або на безоплатній основі (open source application – застосунок з відкритим кодом), то в більшості випадків, розробники приходять до рішення писати застосунок на платформі Android, тому що на сьогодні існує багато інструментів та мов програмування, які дозволяють написання таких застосунків.

1.4 Вибір допоміжних інструментів. Постановка технічного завдання

Під час дослідження, найкращими для виконання проєкту, основними технологіями були обрані:

- Microsoft SQL Server для зберігання даних;
- REST API для обміну даними з сервером;
- XAML для покращення дизайну;
- C#/.NET для написання логіки back-end.

Якщо казати про вибрані технології, то цей вибір, безперечно, був обґрунтований. Першим питанням поставало зберігання даних і операції над ними. Серед найголовніших двух розгалужень при виборі системи управління базами даних (СУБД) є такі: реляційні та нереляційні бази даних. Якщо коротко, то

реляційна БД (або SQL-база, тому що підтримує запити SQL) – база, у котрій дані зберігаються у форматі таблиць, вони суворо структуровані та пов’язані одна з одною. У таблиці є рядки (rows) та стовпчики (columns), кожен є записом, а стовпчик – поле з призначеним йому типом даних. У кожній комірці інформація записана згідно шаблону. А нереляційна БД (NoSQL-база, що, відповідно, не підтримує запити SQL) – зберігає дані без чітких зв’язків між собою та без чіткої структури. Замість структурованих таблиць, всередині бази знаходиться безліч різнорідних документів, в тому числі і зображення, відео та навіть публікації у соціальних мережах. На відміну від реляційних БД, NoSQL бази не підтримують SQL запити[19]. Звернувшись до певних ресурсів [20], виходячи з типу нашого застосунку (мобільна розробка), рекомендується SQL БД. Виходячи з власних знань та досвіду, було обрано досить популярну платформу – MS SQL Server. Але постало головне питання – де розміщувати БД? Враховуючи, що дані, які повинні міститися в загальній БД застосунку (тобто дані користувачів, дані чеків тощо), зберігати дані окремо на кожному девайсі користувача – точно не є рішенням. Отже, треба використати хостинг, щоб застосунок міг звертатися до централізованого сховища даних, який б це девайс або користувач не був. Задля режиму тестування, було обрано власний персональний комп’ютер (ПК) в ролі хостингу для БД. За допомогою REST API відбувався обмін даними між мобільним застосунком та БД, що знаходиться на хостингу.

З приводу розробки мобільного застосунку, то тут навіть попри вибрані технології, все ще залишався вибір, що саме використовувати для дизайну та розробки функціоналу. Але, було вибрано варіант залишити вбудовані механіки (XAML та C#), хоча були варіанти міксувати це з використанням JavaScript-коду. Про інші інструменти та бібліотеки буде розписано детальніше в наступних розділах. Технічне завдання розписано наступним чином:

а) Мобільний додаток:
   1) сторінка входу в додаток:
      – додати поля: електронна пошта, пароль;

- додати кнопки: Увійти, Зареєструватися:
  - кнопка Увійти перевіряє користувача на вміст правильних даних, якщо правильно, то відбувається перехід на головну сторінку, якщо ні – з’являється повідомлення про помилку введених даних;
  - кнопка Зареєструватися переводить користувача на сторінку реєстрації;
- додати клікабельний текст: «Забули пароль?»:
  - відправляє пошту з інструкціями, якщо користувач ввів пошту в полі «Електронна пошта», якщо ні – з’являється повідомлення про помилку;
- зробити дизайн сторінки;

2) сторінка реєстрації:
   - додати поля: прізвище, ім’я, по батькові, електронна пошта, пароль, повторіть пароль;
   - перевіряти чи введені усі дані;
   - дані в «Повторіть пароль» повинні співпадати з «Пароль»;
   - після успішної реєстрації, користувач переходить на сторінку входу;
   - зробити дизайн сторінки;
3) головна сторінка (сторінка з переліком чеків):
   - відображає список чеків з наступною інформацією: логотип магазину, назва і адреса магазину, сума, дата створення чеку
   - при натисканні на чек, користувач потрапляє на сторінку перегляду чеку як документа;
   - зробити дизайн сторінки та чеків;
4) сторінка виклику камери для сканування QR-коду:
   - сканує QR-код, якщо він підходить для додатку, то з’являється повідомлення про успіх і далі можна побачити новий чек на головній сторінці, в разі, якщо QR-код не підходить для додатку,

з’являється повідомлення про помилку і користувач переходить на сторінку з переліком чеків;

5) сторінка профілю:
   – містить інформацію про користувача, а саме його ПІБ та електронну пошту;
   – змінити дизайн сторінки;

б) сервіс для обміну даних БД з мобільним додатком:
1) сервіс для створення QR-коду:
   – за допомогою програмної реалізації, створити метод, що буде приймати JSON тіло, в якому буде масив даних, що відповідає структурі моделі даних GoodsHistory;
   – у відповідь, з’являється QR-код;
2) сервіс для зв’язку мобільного додатку з БД:
   – сервіс виконує усі необхідні функції для роботи мобільного додатку, такі як вхід в аккаунт користувача, реєстрація, скидання паролю, показ чеків відповідно до користувача, створення та показ електронного документу та обробка QR-коду.

Висновки до розділу 1

Поставлені завдання, тобто аналіз теоретичного матеріалу, дослідження, порівняння аналогічних застосунків та вибір технологій, інструментів для розробки, були виконані успішно. Під час виконання поставленого завдання було детально вивчено технологію Xamarin.Forms та порівняно її з іншими технологіями, такими як React Native та Flutter.

Було проведено аналіз інших схожих проєктів, таких як застосунок "Dext", та "EVA" який також забезпечує збереження електронних чеків, та порівняно його з розробленим застосунком.

Отже, на основі результатів виконання завдань можна зробити висновок про те, що створення мобільного застосунку для зберігання електронних чеків є важливим завданням, особливо в контексті змін в законодавстві щодо електронної видачі чеків. Технологія Xamarin.Forms була ефективно досліджена для розробки застосунку, забезпечуючи швидку та ефективну розробку з використанням зрозумілих інструментів для програмістів. За результатами порівняння з іншими схожими проєктами, концепція мобільного застосунку виявилась конкурентоспроможною та має потенціал для подальшого вдосконалення та розвитку.

2 ПОБУДОВА ЗВ'ЯЗКІВ МІЖ ПРОЄКТОМ ТА ІНСТРУМЕНТАМИ. СТВОРЕННЯ МОДЕЛЕЙ. ФУНКЦІОНУВАННЯ ЗОВНІШНІХ ІНСТРУМЕНТІВ

2.1 Побудова зв'язків між мобільним застосунком та зовнішніми інструментами

Щоб зрозуміти, як функціонує застосунок, спочатку треба побачити і заглибитися у його зв'язок між усіма інструментами, які він використовує.

Якщо йти по порядку, перше завдання. яке постало на початку розробки застосунку – авторизація та реєстрація користувача. Посилаючись на навчальне відео[21], вдалося легко налаштувати вищевказаний функціонал. Для цього було обрано один із проєктів Google – Firebase Authenticator. Спочатку, треба було створити власний Developer аккаунт, використовуючи існуючий Google аккаунт. Після цього, потрапляючи на головний екран сайту Firebase, з'являється великий список функціоналу, який можна додати до свого застосунку. Це видно на наступному рисунку.

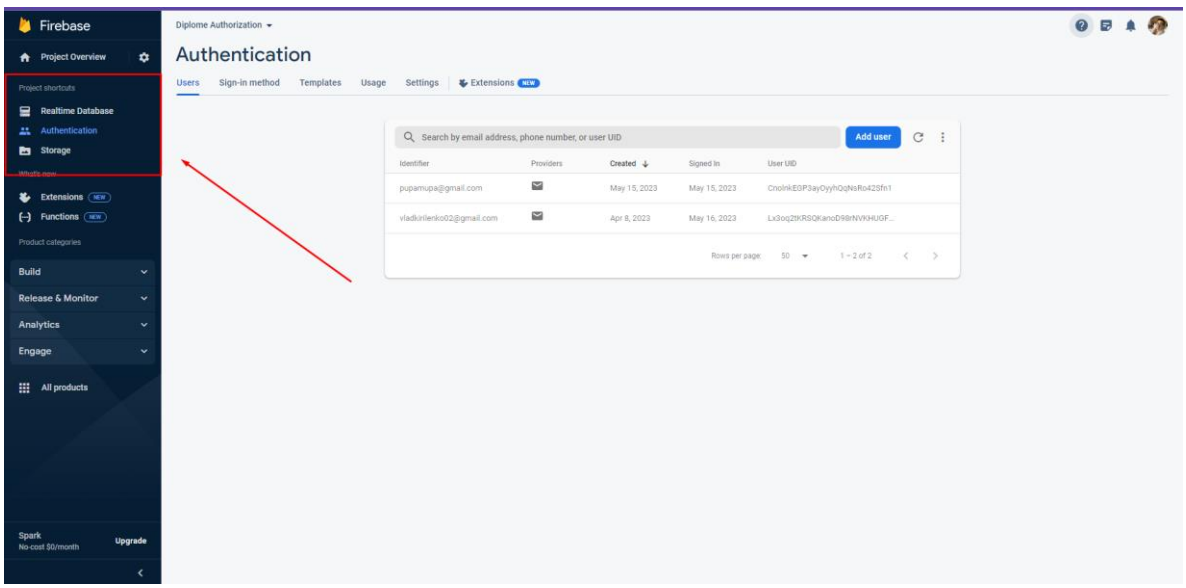

Рисунок 2.1 – Перегляд функціоналу інструменту Firebase

В нашому випадку, нам достатньо підключити до нашого застосунку лише функціонал аутентифікації користувачів, але спочатку, потрібно ознайомитись з інформацією, поданою на сторінці офіційної документації Firebase Console[22].

Для цілей проєкту, достатньо підключити лише авторизацію за допомогою пошти/паролю, хоча на наступному малюнку видно, що варіантів куди більше, наприклад підключення за допомогою соціальних мереж, номеру телефону, а також увімкнути можливість анонімної авторизації тощо.

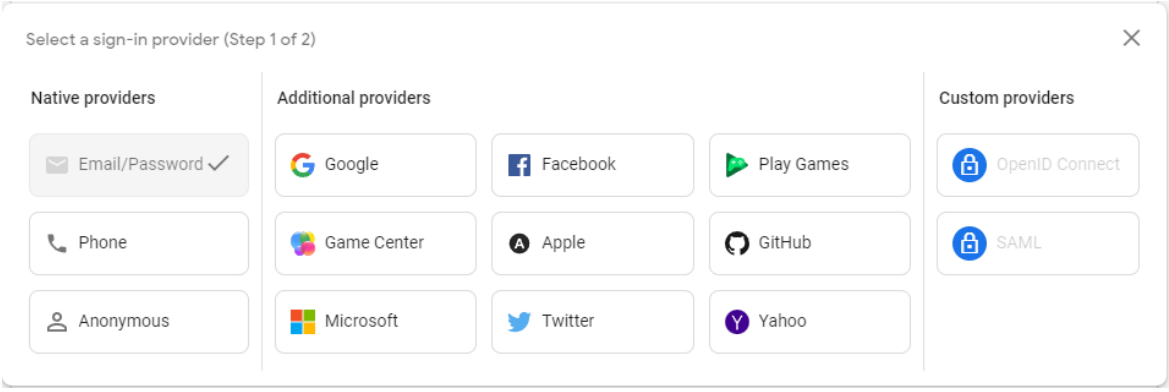

Рисунок 2.2 – Вибір варіантів авторизації

Інструмент виявився успішним для виконання завдання авторизації, тому що має необхідні влаштовані перевірки правильності введених даних (наприклад, перевіряє поле «Пошта» на введення дійсно пошти, в разі неправильних даних, викликає вікно з помилкою і не дає користувачу продовжити реєстрацію/авторизацію) і власні обробники помилок, що, в разі помилки, викликають вікна з детальним описом помилок.

Наступним зовнішнім інструментом став Google Drive API. В Xamarin.Forms є проблема з відображенням файлів, але, виходячи з офіційної документації, було декілька варіантів: або створити можливість створювати електронний чек на девайсі користувача і відтворювати його локально (що не є досить безпечним варіантом, адже просунуті користувачі зможуть змінювати файл), або завантажувати файл за межі девайсу користувача, отримувати посилання на файл і відображати в застосунку. Отже, було обрано варіант винести файл за межі

застосунку. Найкраще для цього підходив саме Google Drive API, адже він безкоштовний, має велику популярність, а також має інструментарій для вільного відображення певних документів, не виконуючи при цьому авторизацію (що й було дуже важливим знайти такий інструмент).

Отже, ознайомившись з офіційною документацією на сайті Google Drive API [23], було прийнято рішення з'єднати застосунок із цим інструментом, використовуючи власний Google аккаунт. Але це вже було набагато складніше, ніж з минулим інструментом. На допомогу приходить навчальне відео[24], яке спростило підключення інструменту до застосунку.

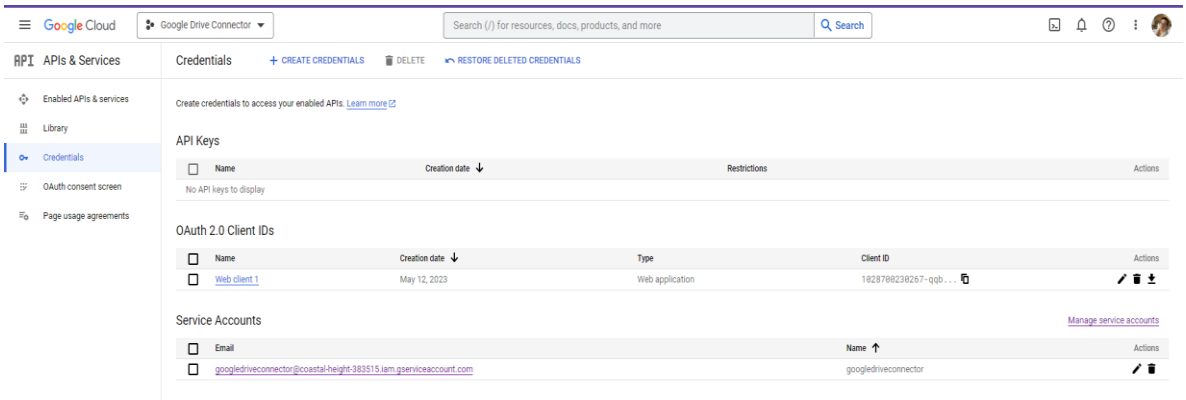

Рисунок 2.3 – Вікно налаштувань Google Drive API

Спочатку, треба створити проєкт і вказати його направлення, тобто в нашому випадку – мобільний застосунок. Після цього, треба пройти вказані етапи і отримати потрібні дані, а саме Client ID та Client secret, що використовуються для розуміння сервісу, з яким саме застосунком йому обмінюватися даними та командами.

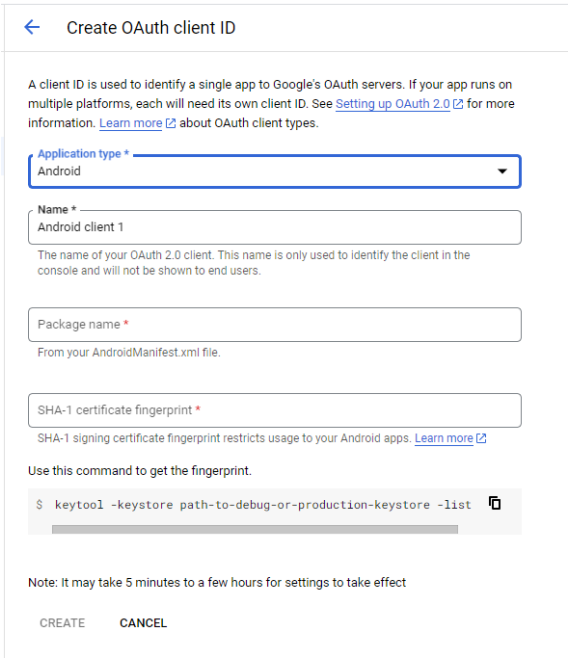

Рисунок 2.4 – Створення вхідних даних для Google Drive API

Після цього, секретні дані можна підключати до певного ресурсу, що приймає секретні дані і видає ключ доступу до сервісу розробнику. Це можна побачити на наступному рисунку

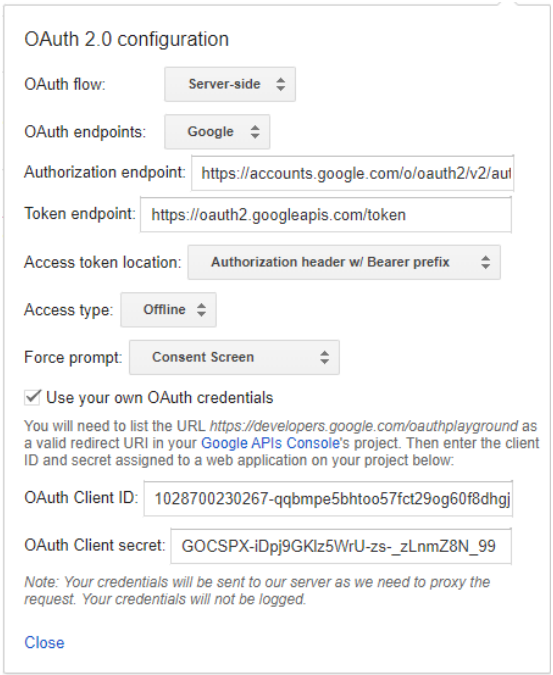

Рисунок 2.5 – Налаштування даних для виклику сервісу, що видає ключ доступу

Після того, як було налаштовано вхідні параметри, далі треба вибрати необхідні сервісі, які будуть використовуватися. В нашому випадку, це лише Drive API v3.

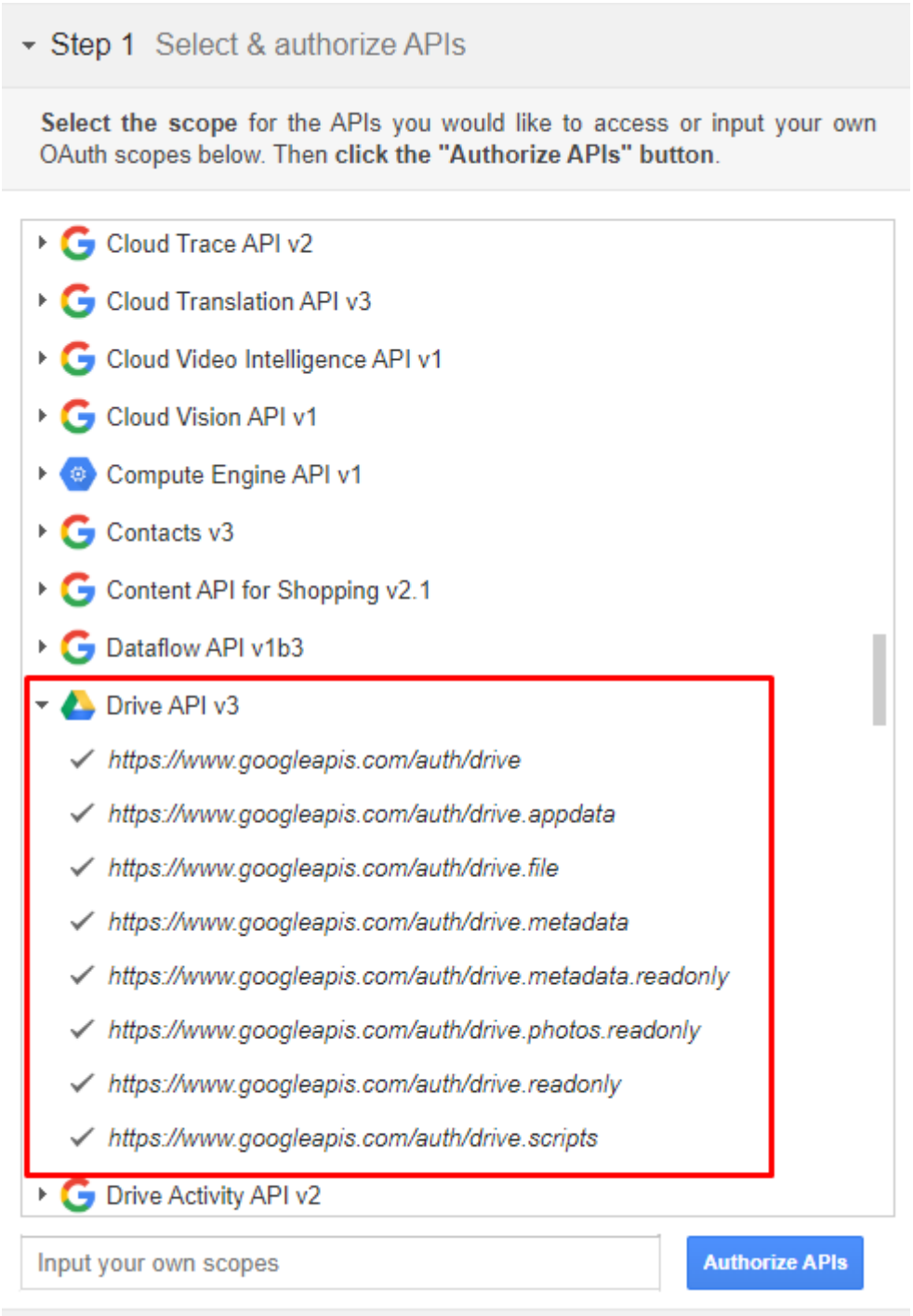

Рисунок 2.6 – Необхідні сервіси для роботи застосунку

Наступний крок налаштування сервісу для отримання ключа доступу є отримання авторизаційного ключа, за допомогою кнопки, що зображена на рис. 2.6 – «Authorize APIs». Далі ми отримуємо авторизаційний ключ, який в подальшому вставляємо в необхідне поле, що зображене на рис. 2.7 і отримуємо 2 необхідні ключа, які використовуватимуться в застосунку при виклику сервісу.

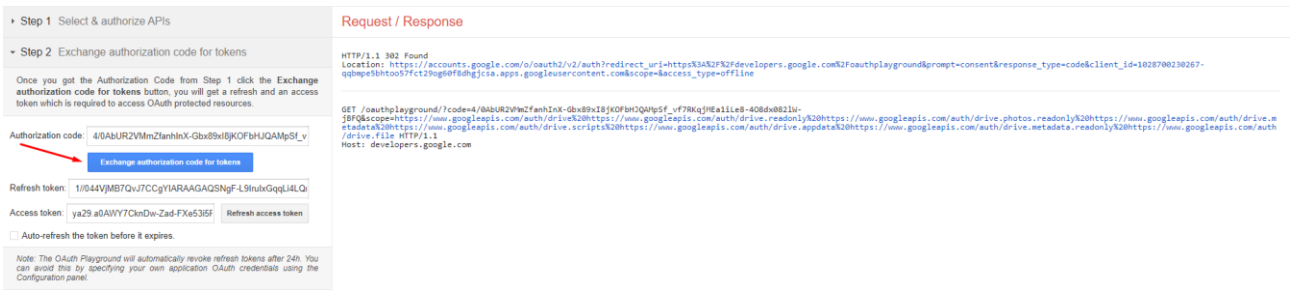

Рисунок 2.7 – Отримання ключів для виклику сервісу в застосунку

Якщо обмін ключами пройшов успішно, ми отримуємо HTTP-повідомлення 302 «Found», що означає, що сервіс знайшов наш введений ключ і передав нам створені ним ключі доступу.

Останнім зовнішній інструмент, який було використано в застосунку – перегляд документів, тобто в нашому випадку використання припало на перегляд електронних чеків прямо у додатку. Знайшовши інформацію на офіційному сайті Github розробників Xamarin.Forms, їхня платформа має певні проблеми із відображенням файлів і не має влаштованого рішення щодо цього. Отже, довелося застосувати трохи креативу. Так як було зазначено вище, зберігати створені чеки локально не є гарним рішенням, треба було вирішити проблему відображення чеків, які зберігаються за межами застосунку і навіть серверу, що контролює обмін даними між БД та застосунком, тобто, в нашому випадку – в Google Drive сховищі. Тому виникла потреба відображати файли, що зберігаються на приватному сховищі, при цьому застосунок не повинен змушувати користувача авторизовуватися в Google Accounts.

Але, Google має засіб перегляду файлу, що знаходиться в хмарному сховищі Google Drive. Це – звичайне посилання виду «https://docs.google.com/viewer?srcid={result}&pid=explorer&efh=false&a=v&chrome=false&embedded=true», де замість result підставляємо ID файлу електронного чеку. Але, щоб уникнути потреби авторизації для перегляду файлу, папку (або файл) треба робити відкритим по посиланню з правами лише переглядача.

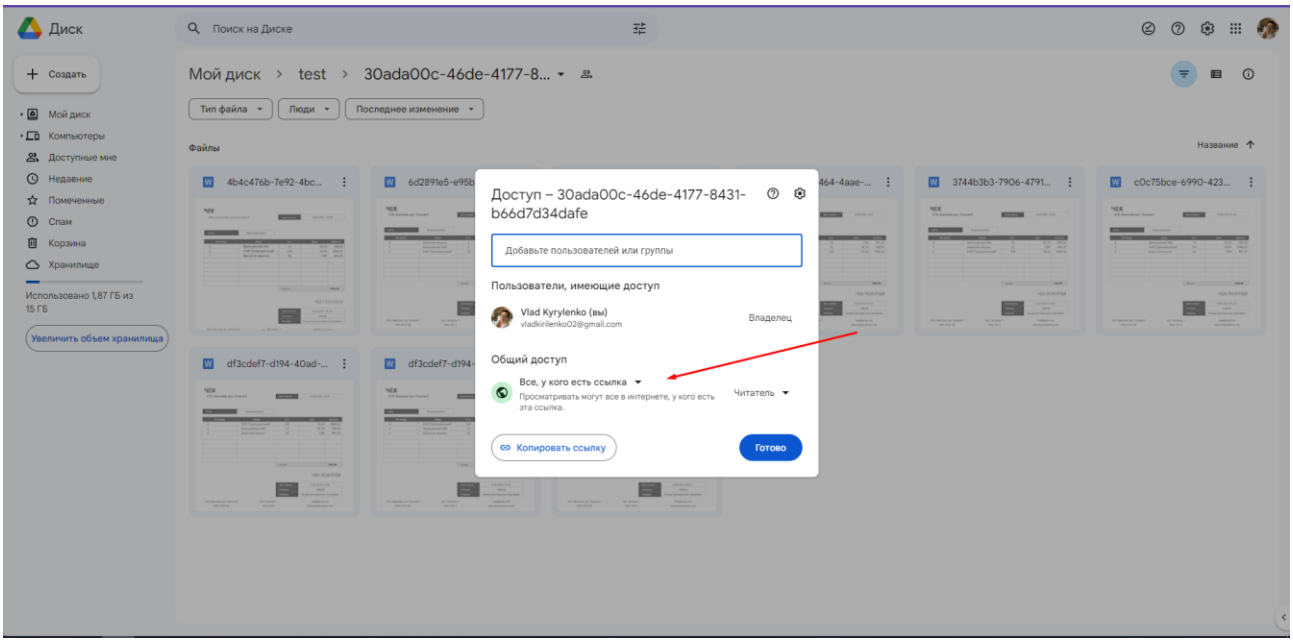

Рисунок 2.8 – Налаштування прав доступу до папки

Таким чином, налаштувавши права доступу для папки, усі файли, які будуть завантажені у цю папку, або створені в ній, будуть автоматично мати такі ж права, як і папка, що і вирішує проблему – файли, які завантажені застосунком, можуть бути переглянуті користувачем без авторизації за допомогою посилання вище, яке створюється автоматично програмою. Кінцевий вигляд зображений на наступному рисунку.

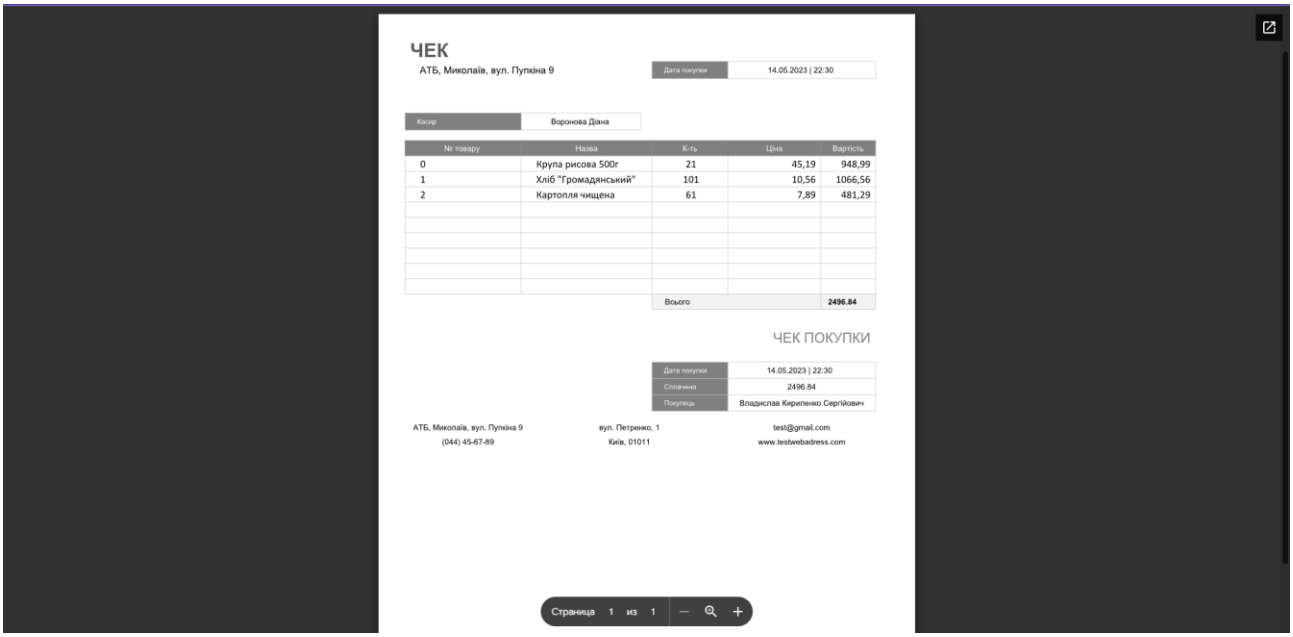

Рисунок 2.9 – Вигляд документу по спеціальному посиланню

Змінити файл при цьому неможливо, а перехід на документ, що знаходиться в хмарному сховищі через застосунок також не є можливим, що і відповідає вимогам кібербезпеки.

2.2 Формування моделей БД та її зв’язок із мобільним застосунком

Одним із непростих завдань було саме правильно реалізувати саму БД, тобто визначити її місце розташування (маючи на увазі або на сервері, або локально, на самому девайсі), вибрати певну СУБД, маючи певний спектр різних систем тощо. Ще у першому розділі було визначено, що для вирішення цієї задачі було обрано MS SQL Server, а місцем розташування БД було обрано віддалений сервер, в даному випадку – власний ПК.

Отже, наступним кроком було створення проєкту. Для цього знову було обрано MS Visual Studio, про яку згадувалось раніше. Так як, по суті, проєкт вважається типу Application Programming Interface (API), то необхідно в Visual Studio обрати саме цей шаблон, так як він одразу допомагає створити декілька необхідних шаблонних класів, папок та інших допоміжних файлів, що пришвидшить розробку на її ранніх етапах.

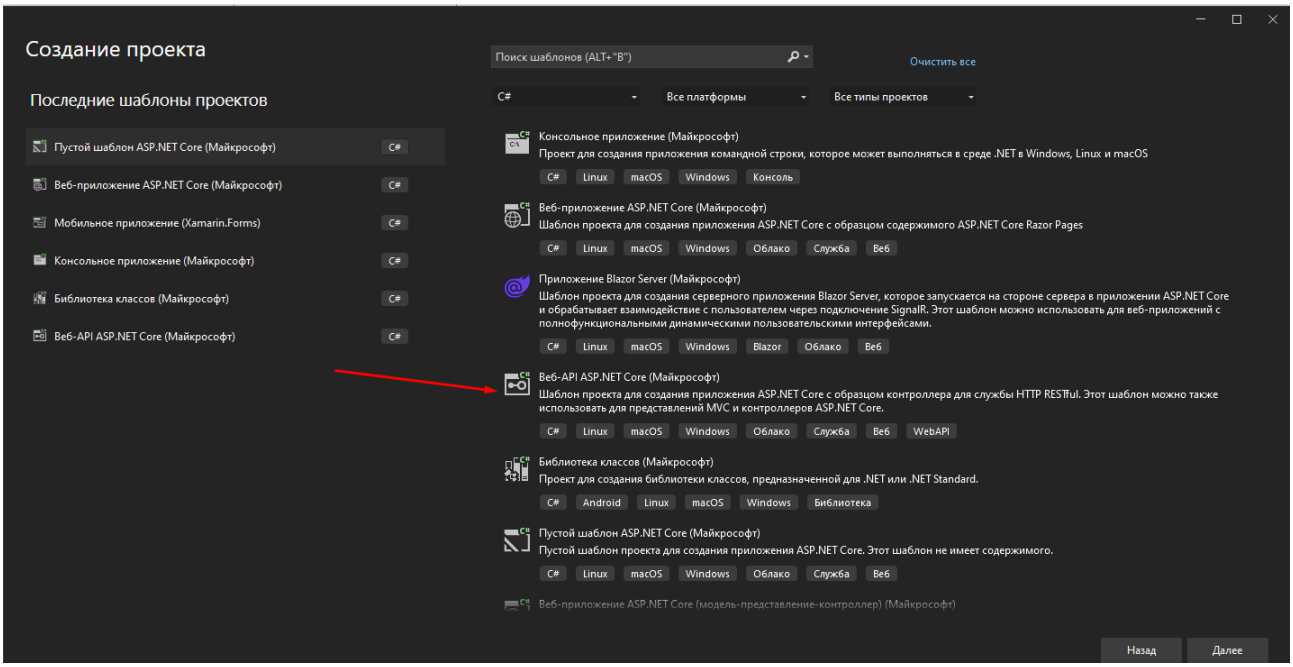

Рисунок 2.10 – Вибір шаблону для розробки

Після цього, підключаємо проєкт до БД за допомогою стандартної connection string[25] для підключення до локальної MS SQL Server БД.

Наступним кроком була розробка структури БД, адже постає питання, скільки треба створити таблиць, з якими колонками та як їх пов’язати між собою. Для вирішення цієї задачі був вибір між різними інструментами, але найбільш зручним виявився «dbForge Studio for MS SQL Server».

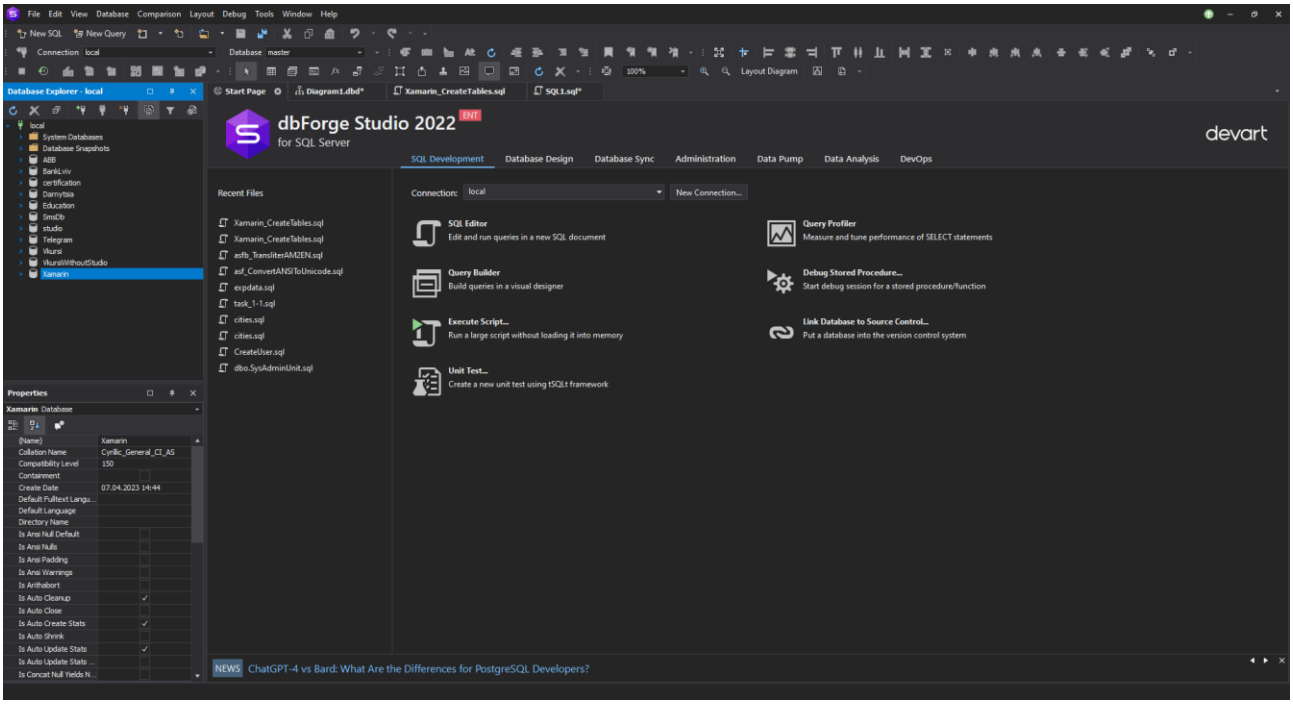

Рисунок 2.11 – Дизайн інструменту оперування даними з БД «dbForge Studio for MS SQL Server»

Цей інструмент відрізняється від звичайних тим, що має графічний інтерфейс, влаштовані команди, що допомагають спростити створення таблиці, вибірку даних, та інші дії, наприклад зміна даних. Також має можливість сортувати, групувати дані, не вносячи зміни в SQL-команду, а безпосередньо оперувати самими результатами. Особливо важливим є ще той факт, коли вже є наявна таблиця, або видозмінена і завданням є отримати код створення таблиці. Аби не проходити все знов по колу, виходячи з наявних даних, програма може написати код створення таблиці автоматично, а також код цілої БД та навіть процедур і функцій. І головне, має інструмент для зображення зв’язків таблиць між собою.

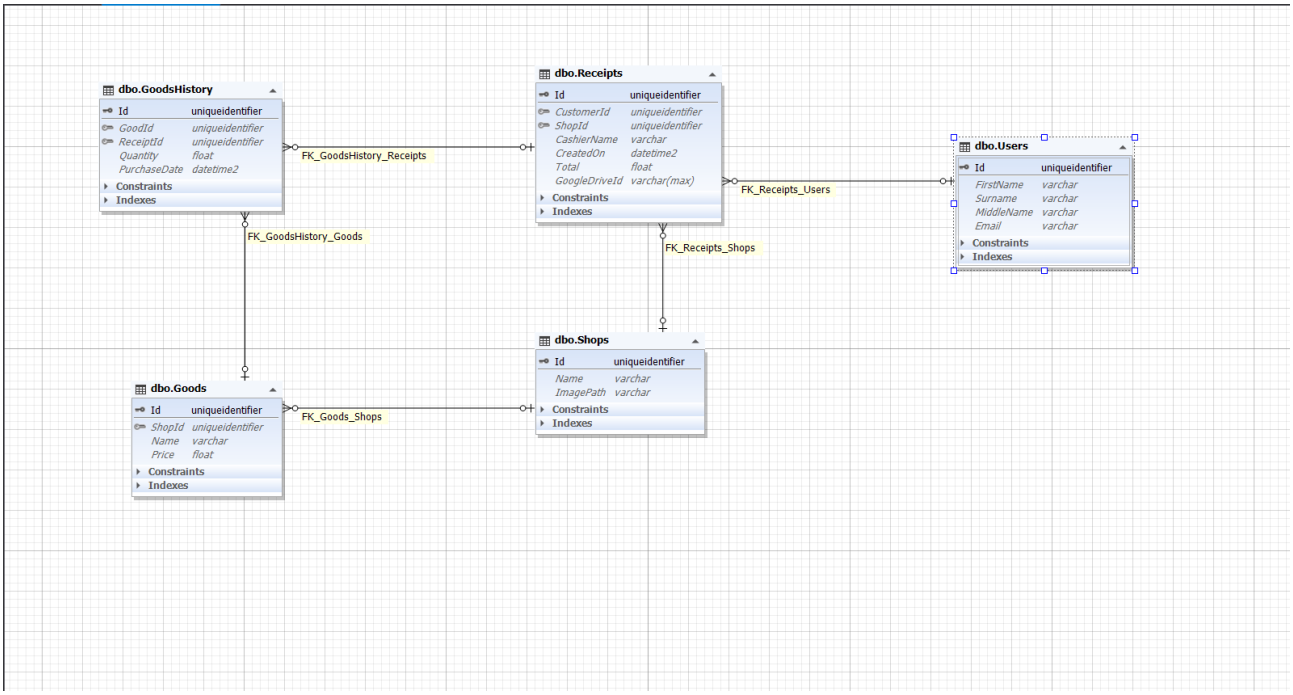

Рисунок 2.12 – Діаграма зв’язку таблиць між собою

Структура БД вийшла не складною. За авторизацію, реєстрацію та логін відповідає таблиця Users, що має такі поля:

– Id як головне унікальне поле,

– FirstName для зберігання ім’я користувача,

– Surname для зберігання прізвища користувача,

– MiddleName для зберігання по батькові людини,

– Email для зберігання електронної пошти користувача.

При цьому, пароль зберігається у раніше згаданому інструменті Firebase Console у вигляді захешованого рядка (такий, що має hash-сіль для безпеки зберігання).

Наступною таблицею є дані про магазини – Shops з простою і маленькою структурою, що має такі поля:

– Id як головне унікальне поле,

– Name для зберігання назви та адреси магазина,

– ImagePath є серверним полем, що зберігає шлях до малюнку логотипа магазину на сервері, яке передає по цьому шляху логотип до мобільного застосунку.

Далі, звісно, йде важлива таблиця з даними про товари, що вміщає в себе просто список товарів усіх магазинів. Але тут вже з'являється перший зв'язок із іншою таблицею Shops, аби розуміти, який товар відноситься до якого магазину, на випадок, якщо товари матимуть однакові назви. Сама таблиця із назвою Goods має такі ключі та поля:

– Id як головне унікальне поле,

– ShopId як ключ, що пов'язує дану таблицю із таблицею Shops,

– Name зберігає назву товару,

– Price зберігає ціну за одиницю товару.

Наступна таблиця є «невидимою» для ока користувача і використовується для підрахунків всередині мобільного застосунку. Ця таблиця має назву GoodsHistory та має наступні поля та ключі:

– Id як головне унікальне поле,

– GoodId як ключ, що пов'язує дану таблицю із таблицею Goods,

– ReceiptId як ключ, що пов'язує дану таблицю із таблицею Receipts (про яку буде йти мова далі),

– Quantity зберігає кількість одиниць товару,

– PurchaseDate зберігає дату появи товару в чеку.

Якщо коротко пояснити зв'язок цієї таблиці із таблицею Goods, то відбувається вибір товару з таблиці Goods покупцем, із цієї таблиці вибирається ціна та назва, а в таблицю історії записується вже кількість одиниць товару і поєднуючи ці дані, розраховується вартість товару в самій вже програмі за окремо кожний товар та загалом за увесь чек. Некоректно було б тримати усі ці розрахунки та дані лише в одній таблиці, саме тому було прийнято рішення створити окремо таблицю-список товарів, та окремо таблицю, що матиме дані про відповідні підрахунки, пов'язані із товарами та чеками.

Далі йде таблиця Receipts, що об'єднує в собі декілька зв'язків з іншими таблицями. Ось їх перелік:

– Id як головне унікальне поле,

– CustomerId як ключ, що пов’язує дану таблицю з таблицею Users,

– ShopId як ключ, що пов’язує дану таблицю з таблицею Shops,

– CashierName зберігає ПІБ касира, що продав товари користувачу додатка,

– CreatedOn зберігає дату створення чеку,

– Total зберігає загальну суму серед всіх товарів в чеку,

– GoogleDriveId зберігає унікальний ідентифікатор електронного чеку, який зберігається в хмарному сховищі Google Drive.

Висновки до розділу 2

В результаті виконання мобільного та API-застосунку було виконано поставлені завдання в даному розділі, а саме імплементація концепції застосунків, побудова та правильно налаштований зв’язок між моделями, налаштований зв’язок між СУБД та застосунками.

Під час виконання завдання було повністю і вичерпно описано інформацію щодо використання зовнішніх інструментів, такі як Google Drive API, dbForge Studio for MS SQL Server та Google Viewer посилання, а також їх безпосереднє підключення до застосунків, покрокові інструкції щодо підключення, певні приклади тощо. Були застосовані знання з джерел офіційних розробників цих інструментів.

Описана побудова моделей та їх зв’язки із застосунками виявили високу зручність використання при розробці застосунків, а також високу читабельність через невелику завантаженість кількістю полів та ключів, що пов’язують таблиці між собою, отже рішення є ефективним.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ API ТА МОБІЛЬНОГО ЗАСТОСУНКІВ. ПРОГРАМНИЙ ЗВ'ЯЗОК ЗАСТОСУНКІВ ТА ІНСТРУМЕНТІВ

3.1 Програмна реалізація таблиць БД за допомогою мови SQL

Програмна реалізація таблиць БД була виконана в інструменті dbForge Studio for MS SQL Server, який розглядався раніше. Спочатку, концепт БД розглядався інакше і мав іншу кількість таблиць та колонок в них. Протягом розробки застосунків, таблиці змінювались і тому початковий код їх створення не підходив для їх відтворення. Отже, на основі поточних таблиць, було застосовано вищезгаданий інструмент для відтворення коду створення цих таблиць. На наступному рисунку показаний алгоритм для відтворення цих дій:

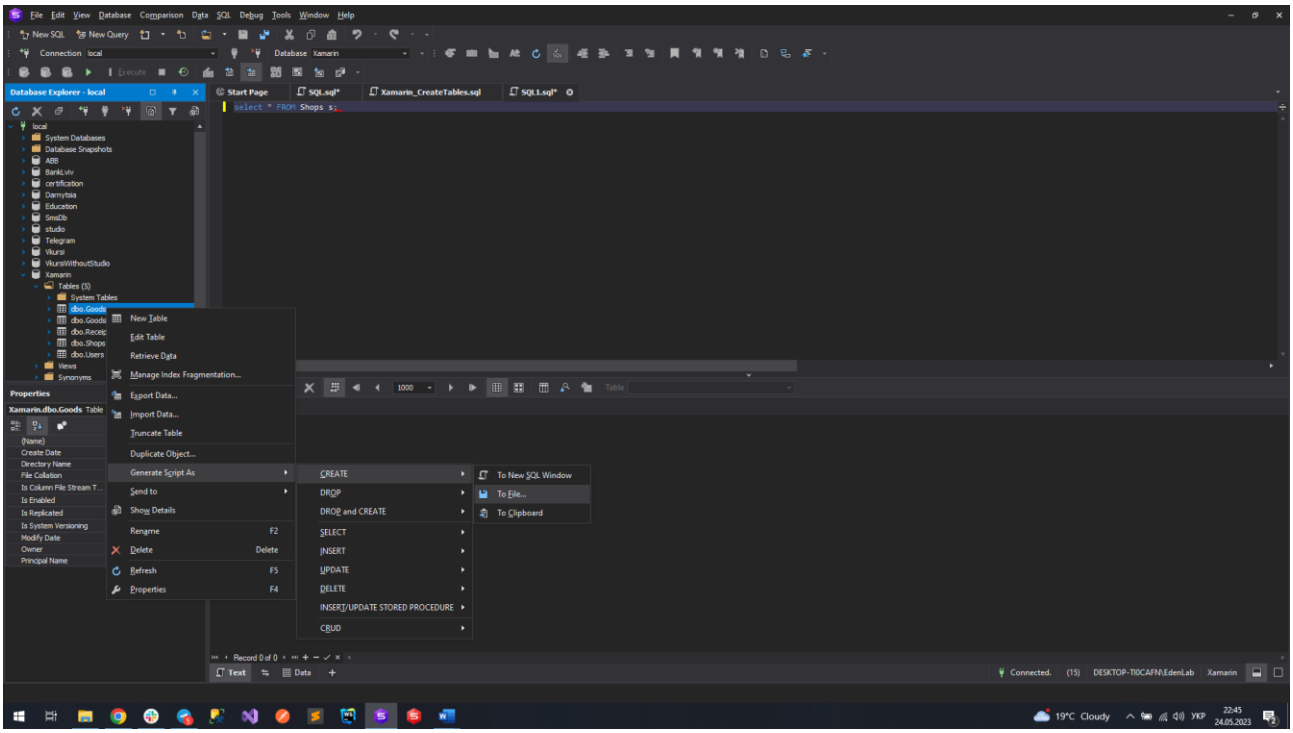

Рисунок 3.1 – Отримання коду для відтворення таблиці

Робимо так з кожною таблицею і отримуємо такий код:

```
CREATE TABLE dbo.Goods (
  Id uniqueidentifier NOT NULL DEFAULT (newid()),
  ShopId uniqueidentifier NULL,
```

```
  Name varchar(100) NULL,
  Price float NULL DEFAULT (0),
  PRIMARY KEY CLUSTERED (Id)
);
ALTER TABLE dbo.Goods
  ADD CONSTRAINT FK_Goods_Shops FOREIGN KEY (ShopId) REFERENCES dbo.Shops (Id);

CREATE TABLE dbo.GoodsHistory (
  Id uniqueidentifier NOT NULL DEFAULT (newid()),
  GoodId uniqueidentifier NULL,
  ReceiptId uniqueidentifier NULL,
  Quantity float NULL DEFAULT (0),
  PurchaseDate datetime2 NULL,
  PRIMARY KEY CLUSTERED (Id)
);
ALTER TABLE dbo.GoodsHistory
  ADD CONSTRAINT FK_GoodsHistory_Goods FOREIGN KEY (GoodId) REFERENCES dbo.Goods
(Id);
ALTER TABLE dbo.GoodsHistory
  ADD CONSTRAINT FK_GoodsHistory_Receipts FOREIGN KEY (ReceiptId) REFERENCES
dbo.Receipts (Id);

CREATE TABLE dbo.Receipts (
  Id uniqueidentifier NOT NULL DEFAULT (newid()),
  CustomerId uniqueidentifier NULL,
  ShopId uniqueidentifier NULL,
  CashierName varchar(100) NULL,
  CreatedOn datetime2 NULL,
  Total float NULL DEFAULT (0),
  GoogleDriveId varchar(max) NULL,
  PRIMARY KEY CLUSTERED (Id)
);
ALTER TABLE dbo.Receipts
  ADD CONSTRAINT FK_Receipts_Shops FOREIGN KEY (ShopId) REFERENCES dbo.Shops (Id);
ALTER TABLE dbo.Receipts
  ADD CONSTRAINT FK_Receipts_Users FOREIGN KEY (CustomerId) REFERENCES dbo.Users
(Id);

CREATE TABLE dbo.Shops (
  Id uniqueidentifier NOT NULL DEFAULT (newid()),
  Name varchar(100) NULL,
```

```
  ImagePath varchar(100) NULL,
  PRIMARY KEY CLUSTERED (Id)
);

CREATE TABLE dbo.Users (
  Id uniqueidentifier NOT NULL DEFAULT (newid()),
  FirstName varchar(50) NULL,
  Surname varchar(50) NULL,
  MiddleName varchar(50) NULL,
  Email varchar(100) NULL,
  PRIMARY KEY CLUSTERED (Id)
);
```

Кожна таблиця має свою унікальну колонку, або первинний ключ (в коді це позначається ключовими словами PRIMARY KEY CLUSTERED ([назва_колонки]) мовою програмування. В нашому випадку, кожна таблиця має колонку Id. Але ці колонки автоматично не заповнюються, тому, при створенні кожної таблиці потрібно додати такі ідентифікатори як NOT NULL, що означає, що колонка не може бути пустою при створенні чи зміні даних, а також додати ідентифікатор DEFAULT [value], де value – початкове значення. Це означає, що при створенні кожного запису в цій таблиці буде автоматично створено ID типу uniqueidentifier (тобто, звичайний GUID[26]). Значення створюється автоматично за допомогою існуючої системної функції newid().

Наступні колонки мали інші ситуативні ідентифікатори та початкові (стандартні) значення. Деякі з них мали ідентифікатор NULL, що є протиставним ідентифікатору NOT NULL, тобто такий, що допускає нульове значення. Вони мали такі типи даних:

– varchar([розмір]), що представляє рядок або текстове значення,
– float для чисел з плаваючою точкою (для збереження грошових даних),
– datetime2 – звичайний формат дати.

Далі, окремо від блоку створення таблиці додаються ключі для зв’язку з іншими таблицями. Для цього застосовується команда ALTER TABLE, що означає зміну табличних даних (не записів таблиці). За допомогою ключового слова

FOREIGN KEY даємо зрозуміти, що ця колонка є зв’язком з іншою колонкою таблиці, яка представлена в другій таблиці як PRIMARY KEY. А ключове слово REFERENCES дає зрозуміти, на яку таблицю та колонку посилається ця колонка.

3.2 Демонстрація мобільного застосунку. Програмна реалізація

Найголовнішим дослідженням цієї теми була саме розробка мобільного застосунку. Демонстрація застосунку починається з логічного початку – сторінки логіну:

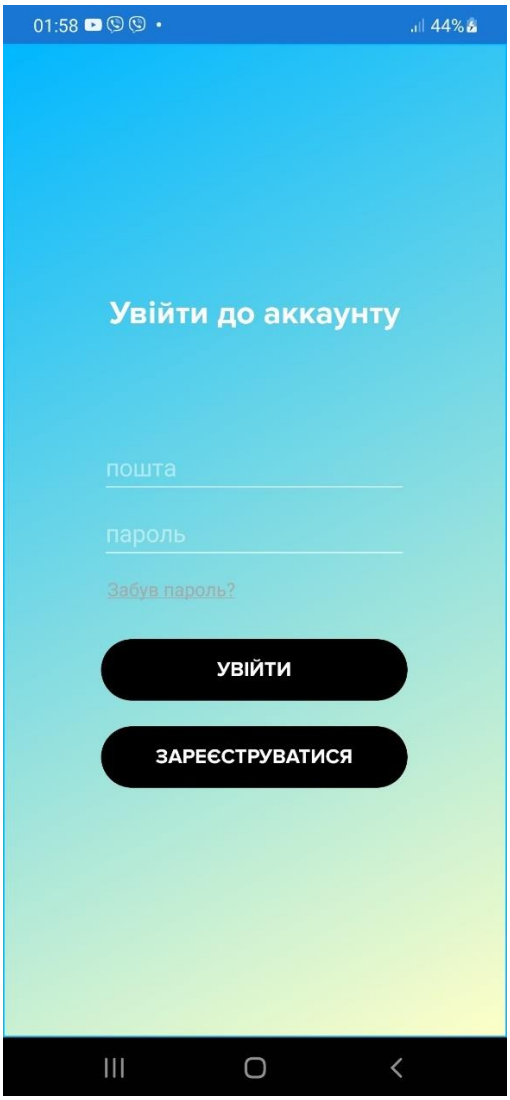

Рисунок 3.2 – Дизайн сторінки логіну

Виходячи з технічного завдання, описаного в першому розділі, сторінка містить усі необхідні елементи: поле для заповнення пошти та пароля, клікабельний надпис для зміни паролю і дві кнопки – для входу та реєстрації. Програмна реалізація дизайну відбувалася за допомогою розмітки XAML:

```
<?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="TestXamarin.LoginPage">
    <Frame BorderColor="#00B7FF"
           HasShadow="True"
           CornerRadius="0"
           HeightRequest="780"
           WidthRequest="400"
           HorizontalOptions="Center" VerticalOptions="Center">
        <Frame.Background>
            <LinearGradientBrush>
                <GradientStop Color="#00B7FF"
                              Offset="0.1" />
                <GradientStop Color="#FFFFC7"
                              Offset="1.0" />
            </LinearGradientBrush>
        </Frame.Background>
        <StackLayout HorizontalOptions="Center" VerticalOptions="Center"
WidthRequest="250">
            <Label Text="Увійти до аккаунту" TextColor="white" Margin="7, 0, 0, 80"
FontSize="Large" FontFamily="{StaticResource ProximaFont}"/>
            <Entry x:Name="EmailEntry" Placeholder="пошта" TextColor="black"/>
            <Entry x:Name="PasswordEntry" Placeholder="пароль" IsPassword="True"
TextColor="black"/>
            <Label Text="Забув пароль?" TextDecorations="Underline"
TextColor="DarkGray" Margin="5">
                <Label.GestureRecognizers>
                    <TapGestureRecognizer Tapped="ResetPasswordTapped"/>
                </Label.GestureRecognizers>
            </Label>
            <Button Text="Увійти" Clicked="LogInButtonClicked"
BackgroundColor="black" Pressed="Button_Pressed" CornerRadius="50"
FontFamily="{StaticResource ProximaFont}" HeightRequest="50" Margin="0, 20, 0, 15"/>
            <Button Text="Зареєструватися" Clicked="RegisterViaEmailButtonClicked"
Pressed="Button_Pressed" BackgroundColor="black" CornerRadius="50"
FontFamily="{StaticResource ProximaFont}" HeightRequest="50"/>
        </StackLayout>
    </Frame>
      </ContentPage>
```

Частина нефункціонального коду була прописана на C#, наприклад подія натискання на кнопки. Під час цієї події, кнопка зникає на певний час, та з’являється знову, створюючи таким чином анімацію натискання. Додатково для дизайну було завантажено шрифт. Код події:

```
private async void Button_Pressed(object sender, EventArgs e)
        {
            var button = sender as Button;
            await button.FadeTo(0, 50);
            await button.FadeTo(1, 50);
        }
```

Сторінка має необхідні перевірки на заповнення полів, а також відповідні повідомлення у вигляді діалогових вікон.

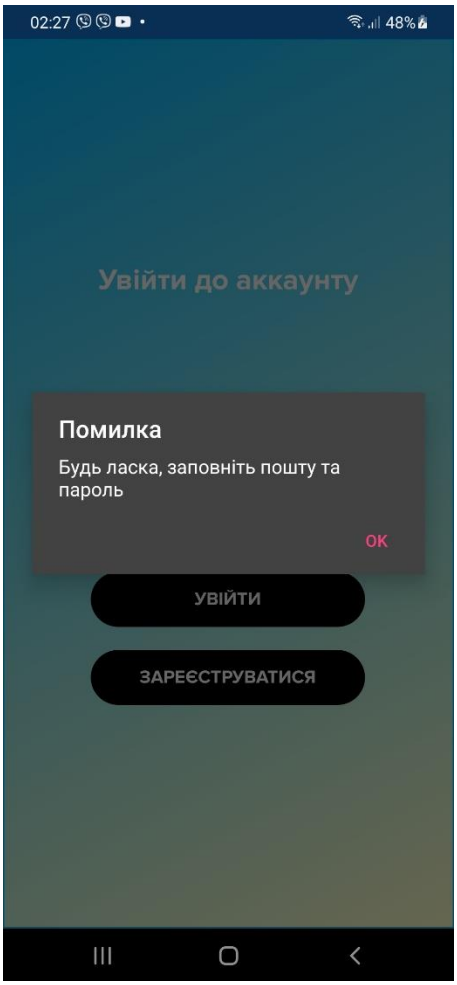

Рисунок 3.3 – Діалогове вікно з повідомленням

За перевірку введених даних та подальшу обробку відповідає наступний код:

```
private async void LogInButtonClicked(object sender, EventArgs e)
        {
            var email = EmailEntry.Text;
            App.UserEmail = email;
            var password = PasswordEntry.Text;
            if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
            {
                await DisplayAlert("Помилка", "Будь ласка, заповніть пошту та
пароль", "ОК");
                return;
            }

            var firebase = DependencyService.Get<IFirebaseAuthenticator>();
            try
            {
                await firebase.SignInWithEmailAndPasswordAsync(email, password);
                await Navigation.PushAsync(new MainPage());
            }
            catch (Exception ex)
            {
                await DisplayAlert("Помилка", ex.Message, "ОК");
            }
        }
```

Код спрацьовує під час натискання саме кнопки «Увійти». Спочатку перевіряє, чи введені пошта та пароль, а потім за допомогою звернення до сервісу FirebaseAuthenticator, відправляє запрос на сервіс Firebase і, за умови успіху, переходить на головну сторінку з чеками. Якщо умови не виконані, з'являється повідомлення як на рис. 3.3 (при цьому зміст повідомлення може змінюватися, в залежності від даних). Сам інтерфейс IFirebaseAuthenticator виглядає так:

```
public interface IFirebaseAuthenticator
    {
        Task<string> SignInWithEmailAndPasswordAsync(string email, string password);

        Task<string> CreateUserWithEmailAndPasswordAsync(string email, string password);

        Task SendPasswordResetEmailAsync(string email);
    }
```

Тобто, можна побачити, що інтерфейс містить лише 3 методи: перший – для логіну користувача, другий – для створення користувача за поштою та паролем, тобто користувач записується у сервіс Firebase, третій – для скидання паролю.

Останній функціонал, який реалізований на цій сторінці – скидання паролю. При натисканні на клікабельний надпи «Забув пароль?» з’являється діалогове вікно (рис. 3.4) з повідомлення про відправку на пошту інструкцій. Але головна умова, щоб це працювало – введена пошта в поле для пошти, а також її існування в базі Firebase. Так це виглядає в дії:

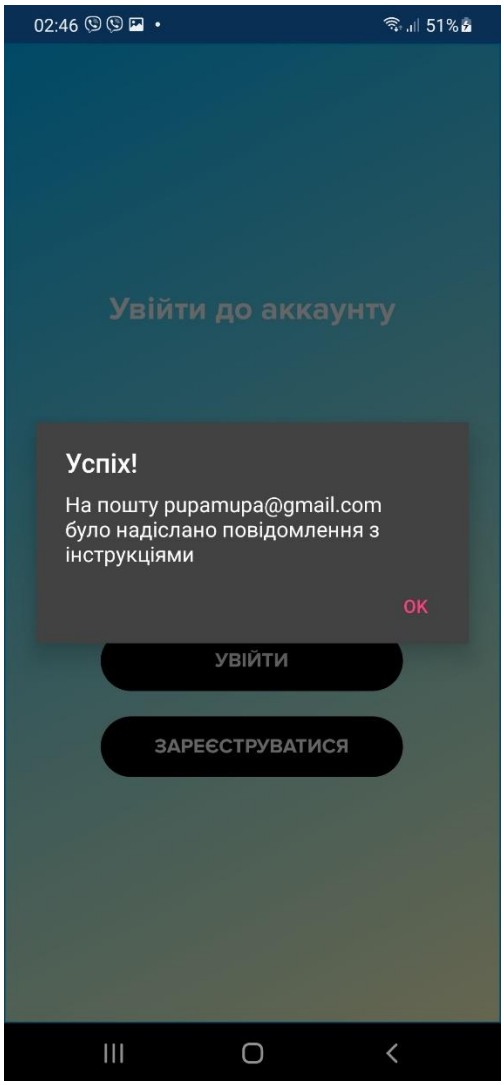

Рисунок 3.4 – Скидання паролю

В повідомленні зазначається посилання, на яке потрібно перейти для скидання паролю, а також містяться поля (рис. 3.5) для введення нового паролю, який потім збережеться в базі Firebase і буде в подальшому використовуватися для

входу з цією поштою. Також, Firebase Console має додаткові можливості для входу, наприклад вхід через соціальні мережі.

На пошті письмо користувачу для скидання паролю виглядає наступним чином:

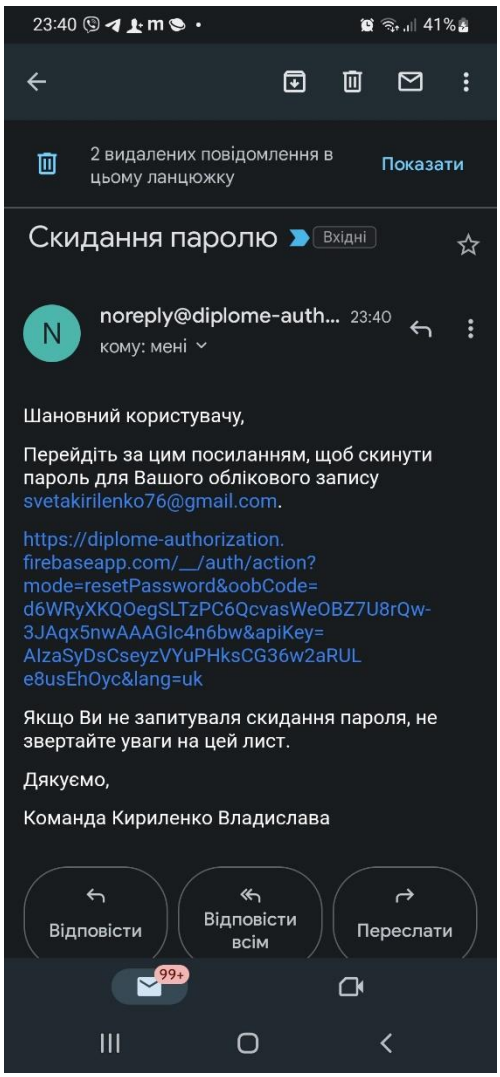

Рисунок 3.5 – Письмо для скидання паролю

Після переходу по посиланню, користувач має можливість вибрати собі новий пароль, який буде застосовуватися при наступному логіні до застосунку. Вебсторінка є влаштованною в сервіс Firebase.

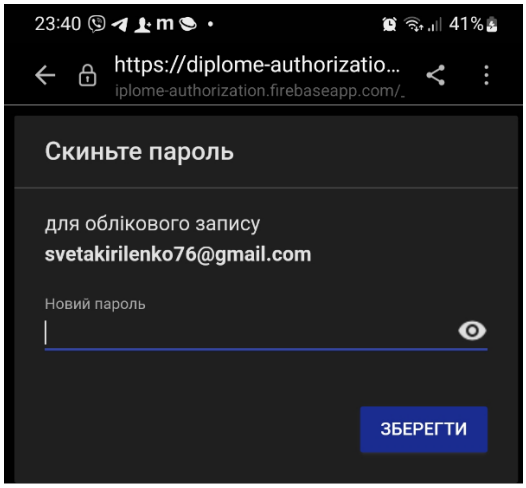

Рисунок 3.6 – Сторінка Firebase для скидання паролю

Наступним елементом для розгляду є сторінка реєстрації з полями для введення даних ПІБ, паролю та пошти. Сторінка має такий вигляд:

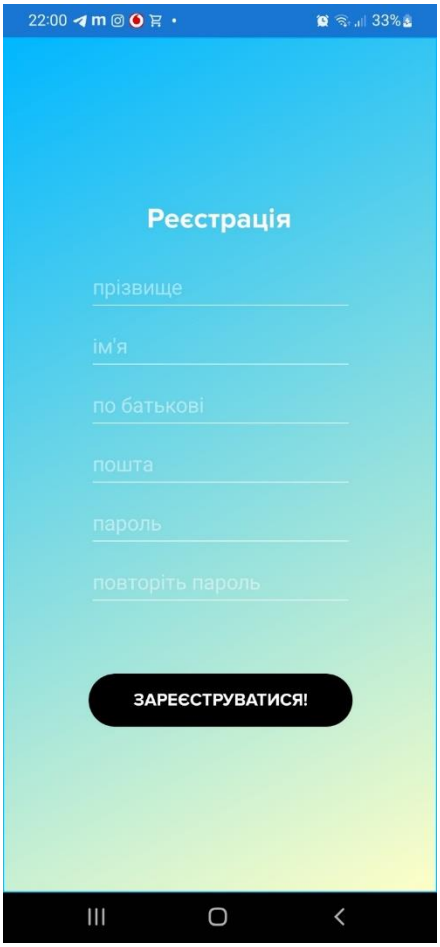

Рисунок 3.7 – Сторінка реєстрації

Інтерфейс сторінки було розроблено за допомогою XAML:

```
<?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="TestXamarin.RegisterPage">
    <Frame BorderColor="#00B7FF"
           HasShadow="True"
           CornerRadius="0"
           HeightRequest="780"
           WidthRequest="400"
           HorizontalOptions="Center" VerticalOptions="Center">
        <Frame.Background>
            <LinearGradientBrush>
                <GradientStop Color="#00B7FF"
                              Offset="0.1" />
                <GradientStop Color="#FFFFC7"
                              Offset="1.0" />
            </LinearGradientBrush>
        </Frame.Background>
        <StackLayout HorizontalOptions="Center" VerticalOptions="Center"
WidthRequest="250" Margin="-10, 0, 0, 0">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="0" />
                    <ColumnDefinition Width="250" />
                    <ColumnDefinition Width="50" />
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                    <RowDefinition Height="*" />
                </Grid.RowDefinitions>
                <Label Grid.Row="0" Grid.Column="1" Text="Реєстрація"
TextColor="white" Margin = "55, 0, 0, 0" FontSize="Large"
FontFamily="{StaticResource ProximaFont}" HeightRequest="30"/>
                <Entry Grid.Row="1" Grid.Column="1" x:Name="SurnameEntry"
Placeholder="прізвище" HeightRequest="50"/>
                <Entry Grid.Row="2" Grid.Column="1" x:Name="NameEntry"
Placeholder="ім'я" HeightRequest="50"/>
                <Entry Grid.Row="3" Grid.Column="1" x:Name="MiddleNameEntry"
Placeholder="по батькові" HeightRequest="50"/>
                <Entry Grid.Row="4" Grid.Column="1" x:Name="EmailEntry"
Placeholder="пошта" HeightRequest="50"/>
                <Entry Grid.Row="5" Grid.Column="1" x:Name="PasswordEntry"
Placeholder="пароль" IsPassword="True" HeightRequest="50"/>
```

```
                <Entry Grid.Row="6" Grid.Column="1" x:Name="PasswordRepeatEntry"
Placeholder="повторіть пароль" IsPassword="True" HeightRequest="50"/>
                <Button Grid.Row="8" Grid.Column="1" Text="Зареєструватися!"
Clicked="ButtonRegisterClicked" Pressed="Button_Pressed" BackgroundColor="black"
CornerRadius="50" FontFamily="{StaticResource ProximaFont}" HeightRequest="50"/>
            </Grid>
        </StackLayout>
    </Frame>
</ContentPage>
```

Структура даної розмітки відрізняється від минулої тим, що тут застосована Grid-розмітка[27], яку можна коротко описати як площину, яка поділена на частини за допомогою горизонтальних та вертикальних прямих ліній. Площини можуть бути різні за розмірами, що й допомагає робити розмітку зручно і найголовніше – адаптивно.

Серверний функціонал сторінки перевіряє наявність усіх введених полів, а в разі порушення цих умов, видає помилку, що зображена на рис. 3.6.

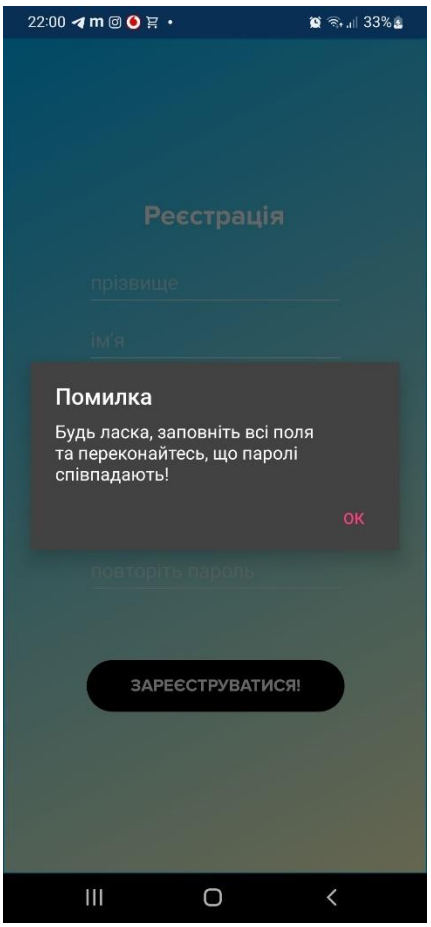

Рисунок 3.8 – Перевірка заповнення полів під час реєстрації користувача

Код перевірки виглядає наступним чином:

```
            var client = App.GetClient();
            var email = EmailEntry.Text;
            var password = PasswordEntry.Text;
            var passwordRepeat = PasswordRepeatEntry.Text;
            var surname = SurnameEntry.Text;
            var firstName = NameEntry.Text;
            var middleName = MiddleNameEntry.Text;
            if (string.IsNullOrEmpty(email) ||
                string.IsNullOrEmpty(password) ||
                string.IsNullOrEmpty(passwordRepeat) ||
                string.IsNullOrEmpty(surname) ||
                string.IsNullOrEmpty(firstName) ||
                string.IsNullOrEmpty(middleName) ||
                password != passwordRepeat)
            {
                await DisplayAlert("Помилка", "Будь ласка, заповніть всі поля та
переконайтесь, що паролі співпадають!", "ОК");
                return;
            }
```

Після виконаних умов, користувача повертає на сторінку логіна, аби він міг увійти у застосунок, а також уся необхідна інформація за допомогою API застосунку записується новим записом у таблиці Users, а також відбувається виклик сервісу в Firebase:

```
await client.PostAsync(Consts.ApiStrings.Url + Consts.ApiStrings.CreateUser, new
StringContent(JsonConvert.SerializeObject(new CreateUserModel
            {
                Email = EmailEntry.Text,
                FirstName = NameEntry.Text,
                MiddleName = MiddleNameEntry.Text,
                Surname = SurnameEntry.Text
            }), Encoding.UTF8, "application/json"));

            var firebase = DependencyService.Get<IFirebaseAuthenticator>();

            try
            {
                await firebase.CreateUserWithEmailAndPasswordAsync(email, password);

                await DisplayAlert("Успіх!", $"Користувача з поштою {email} успішно
зареєстровано", "ОК");

                await Navigation.PopAsync();
            }
```

Після успішного логіну в застосунок, користувач потрапляє на головну сторінку, а саме на сторінку зі списком чеків.

Рисунок 3.9 – Головна сторінка зі списком чеків

Найголовніше – це чеки. На цій сторінці вони представлені логотипом магазину, назва магазину та його адреса, сума товарів у чеку та дата створення чеку.

Дизайн сторінки відрізняється від минулих представлених сторінок так само як і її розмітка, адже головною відмінністю є динамічний контент. Кожен користувач має різну кількість чеків із різних магазинів, з різною сумою, датою тощо. Саме тому, в розмітці відбувається прив’язка елементів до даних, що, в свою чергу, наповнюються за допомогою серверного функціоналу. Окрім динамічного контенту, було також застосовано новий спосіб розмітки – FlexLayout, або

«резинова розмітка», що дозволяє програмі розставляти елементи, виходячи з розміру девайсу або контейнера. Такі елементи завжди змінюють своє положення/розмір після зміни розмірів їх контейнерів.

```
<ListView x:Name="ReceiptsList"
                      HasUnevenRows="True"
                      ItemsSource="{Binding Receipts}"
                      ItemTapped="OnItemTapped"
                      SeparatorColor="white">
                <ListView.ItemTemplate>
                    <DataTemplate x:DataType="data:ReceiptsShopsModel">
                        <ViewCell>
                            <StackLayout Orientation="Horizontal" Margin="0, 0, 0,
30">
                                <Image Source="{Binding ShopImagePath, Mode=TwoWay}"
WidthRequest="150" HeightRequest="110"/>
                                <StackLayout>
                                    <FlexLayout Direction="Column">
                                        <Label Text="{Binding ShopName}"
FontSize="16" TextColor="black" FontFamily="{StaticResource ProximaFont}"
HeightRequest="40" WidthRequest="250"/>
                                        <Label Text="{Binding Total,
StringFormat='Сума: {0:C} ₴', Mode=TwoWay, Converter={StaticResource
FloatConverter}}" TextColor="Gray" FontFamily="{StaticResource
PoppinsMediumItalic}"/>
                                        <Label Text="{Binding CreatedOn,
StringFormat='{0:dd MMMM yyyy | HH:mm}', Mode=TwoWay}" TextColor="black" Margin="0,
20, 0, 0"/>
                                    </FlexLayout>
                                </StackLayout>
                            </StackLayout>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
```

Прив’язка відбувається за допомогою властивості Binding та назви властивості у серверному коді.

Щодо серверного коду, при переході на сторінку відбувається завантаження чеків з БД відповідно до користувача. Найголовнішим з цього є те, що звичайні колекції в Xamarin не є автоматично оновлюваними, саме тому доводиться використовувати таку колекцію, що реалізовує інтерфейс INotifyPropertyChanged або вже влаштовану в систему колекцію – ObservableCollection. Виходячи з назви, інтерфейс вимагає від користувача реалізацію метода, який сповіщуватиме програму про оновлення даних властивостей моделей, що дозволить передати ці

дані розмітці і змінити дизайн відповідно повідомленням, які були створені за допомогою вищезазначеної реалізації інтерфейсу. Тому, було прийнято рішення використати клас ObservableCollection для виконання цієї задачі. Оновлення колекції виглядає наступним чином:

```
public ObservableCollection<ReceiptsShopsModel> Receipts { get; set; }

        public ReceiptsListPage()
        {
            InitializeComponent();

            Receipts = new ObservableCollection<ReceiptsShopsModel>();
            BindingContext = this;
        }

        private ObservableCollection<ReceiptsShopsModel> InitReceipts()
        {
            var result = string.Empty;
            try
            {
                var client = App.GetClient();
                var response = client.GetAsync(Consts.ApiStrings.Url +
Consts.ApiStrings.GetReceiptsAndShops + App.UserEmail).Result;
                result = response.Content.ReadAsStringAsync().Result;
            }
            catch (Exception ex)
            {
                DisplayAlert("error", ex.Message, "ok").RunSynchronously();
            }

            var receipts =
JsonConvert.DeserializeObject<ObservableCollection<ReceiptsShopsModel>>(result);

            return receipts;
        }
```

Можна побачити, що в цьому методі відбувається виклик API сервісу, що повертає колекцію чеків, а саме оновлення відбувається ще в конструкторі сторінки, тобто після її ініціалізації.

Важливою деталлю є те, що для успішної прив’язки даних треба прив’язати контекст даних. Це робиться або в розмітці, або в серверному функціоналі. Було вирішено зробити це серверним кодом («BindingContext = this»). Також, для уникнення помилок, рекомендується не викликати жодні асинхронні функції у конструкторі.

Оновлення змісту списку чеків відбувається не лише при вході в застосунок, але й при переході між сторінками (реалізовано для випадку, коли користувач отримує чек після успішного сканування QR-коду):

```
private void Receipts_Appearing(object sender, EventArgs e)
        {
            Receipts.Clear();
            var receipts = InitReceipts();
            foreach (var receipt in receipts)
            {
                Receipts.Add(receipt);
            }
        }
```

Список повинен не просто змінюватися або прирівнюватися іншій змінній, а саме оновлювати існуючу колекцію. Тобто, в цьому випадку, спочатку робиться очищення списку і поетапне його завантаження (відбувається додавання послідовне додавання кожного елементу в колекцію).

Також на головній сторінці можна побачити внизу екрану переходи по іншим сторінкам, що містять іконки та написи, що дають зрозуміти користувачу куди ведуть ці сторінки та пояснюють їх роботу.

Одним із найбільш важливих функціоналів цього застосунку – відображення цих чеків. Вони створюються за допомогою існуючого Word-шаблону і за допомогою бібліотеки Xceed, наповнює шаблон потрібними значеннями. Сам чек зберігається на хмарному сховищі Google Drive, а користувач може переглянути свій чек за допомогою натискання на потрібний йому чек, адже в БД для цього відведене окреме поле, що зберігає ID файлу в Google Drive, за допомогою якого потім і формується посилання на перегляд файлу у застосунку. При цьому, користувач не може редагувати даний файл або переглядати його за лаштунками застосунку.

Рисунок 3.10 – Електронний чек

Чек містить усі необхідні поля, що показують усю детальну інформацію про покупки: час покупки, назви товарів, їх ціна, кількість та окремо порахована вартість кожної позиції, загальна сума покупки, адреса магазину, ім’я та прізвище касира тощо.

Наступним розділом для розгляду є сторінка сканування QR-коду. Сама по собі вона не представляє собою ніякого наповнення дизайном і не містить складну XAML-розмітку:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="TestXamarin.CameraPage"
             xmlns:zxing="clr-
namespace:ZXing.Net.Mobile.Forms;assembly=ZXing.Net.Mobile.Forms"
             Title="Сканування" IconImageSource="qr_code.png">
    <StackLayout>
        <zxing:ZXingScannerView IsScanning="True" OnScanResult="OnScanResult"/>
    </StackLayout>
</ContentPage>
```

Після того, як бібліотека ZXing була завантажена за допомогою Nuget[28], з’являється можливість використовувати їхні елементи у розмітці. В даному випадку використовується сканер, який вмикає камеру девайса та при знаходженні QR-коду реагує на це і оброблює результати. Аби при скануванні виконувались певні дії, додано властивість OnScanResult і передано посилання на метод, що оброблює відсканований QR-код:

```
Device.BeginInvokeOnMainThread(async () =>
            {
                var alerts = new string[] { };
                var guid = HandleScanResult(result);
                if (guid is null)
                {
                    alerts = new[] { "Неправильний QR-код", "Цей QR-код не є
валідним для цього додатку", "ОК" };
                    Navigation.InsertPageBefore(new MainPage(alerts),
Navigation.NavigationStack[0]);
                    await Navigation.PopToRootAsync();
                    return;
                }

                if (App.CachedResults.Contains(result.Text))
                {
                    return;
                }
                App.CachedResults.Add(result.Text);

                var client = App.GetClient();
                var response = await client.PostAsync(
                    Consts.ApiStrings.Url + Consts.ApiStrings.BindReceiptToCustomer,
                    new StringContent(
                        JsonConvert.SerializeObject(new BindReceiptModel
                        { ReceiptId = guid.Value, UserEmail = App.UserEmail }),
Encoding.UTF8,
                        "application/json"));
                if (response.StatusCode != System.Net.HttpStatusCode.OK)
                {
                    alerts = new[] { "Помилка", response.StatusCode.ToString(), "ОК"
};
```

```
                    Navigation.InsertPageBefore(new MainPage(alerts),
Navigation.NavigationStack[0]);
                    await Navigation.PopToRootAsync();
                    return;
                }

                alerts = new[] { "Успіх!", "Чек додано!", "ОК" };
                Navigation.InsertPageBefore(new MainPage(alerts),
Navigation.NavigationStack[0]);
                await Navigation.PopToRootAsync();
            });
```

При роботі з камерою обов’язково треба мати на увазі, що в цей час програма повинна знаходитися на головному потоці (адже як відомо, наразі смартфони є багатоядерними і мають багато потоків, що дозволяє програмі одночасно використовувати декілька потоків для підвищення продуктивності), інакше це може призвести до помилок та «падіння» застосунку. Це робиться за допомогою делегату Device.BeginInvokeOnMainThread, що приймає функцію і виконує її в головному потоці. Також викликається API сервіс для прив’язки новоствореного чеку до користувача в БД. Для правильного показу діалогових вікон користувачу було обрано опцію додавання параметрів до конструкторів сторінок, що прийматимуть рядок з повідомленням, і в разі отримання такого повідомлення, сторінка завантажується разом з діалоговим вікном. Тобто, після використання скану, користувач потрапляє на головну сторінку з певним повідомленням, що сигналізує про успіх або навпаки – помилку, що має певний її опис.

З приводу помилки, їх є 2 типи: перший, що сигналізує про сканування QR-коду, але такого, що не відповідає потребам програми (має не такий URL), а другий сигналізує про певні серверні помилки, тобто розрив з інтернетом (коли відбувається виклик сервісу), або проблеми з камерою тощо. Наступний код робить обробку URL, що представляє собою QR-код:

```
private Guid? HandleScanResult(ZXing.Result result)
        {
            var match = Regex.Match(result.Text,
                @"(^([0-9A-Fa-f]{8}[-]?[0-9A-Fa-f]{4}[-]?[0-9A-Fa-f]{4}[-]?[0-9A-Fa-
f]{4}[-]?[0-9A-Fa-f]{12})$)");
            if (string.IsNullOrEmpty(match.Value))
```

```
        {
            return null;
        }
        return Guid.Parse(match.Value);
    }
```

Цей довгий рядок є Regex-рядком – спеціальні рядки, що за певним шаблоном можуть шукати певні значення в тексті. В даному випадку, ми перевіряємо, чи містить QR-код необхідний GUID чеку для подальшої прив’язки до користувача за допомогою API сервісу.

Переходячи до самої сторінки, першим необхідним рішенням було отримання дозволу девайсу на використання застосунком камери та інших інструментів, такі як Wi-Fi, мобільний Інтернет та інші:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" android:versionName="1.0"
package="com.companyname.testxamarin">
    <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="33" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.CAMERA" />
    <application android:label="TestXamarin.Android"
android:theme="@style/MainTheme" android:usesCleartextTraffic="true"></application>
</manifest>
```

Після отримання необхідних дозволів, при першому їхньому використанні, застосунок спитає користувача, чи надає він дозвіл на використання, але в подальшому це не буде турбувати користувача (якщо не вибрати варіант «Дозволити лише цього разу»). Дозвіл на використання безпровідних мереж девайсу та на статус цих мереж необхідний для зв’язку між мобільним застосунком та API сервісом. Так само і відбувається з дозволом на використання камери – при першому відкритті користувачем сторінки сканування QR-коду, користувач повинен надати дозвіл на використання його камери, в іншому випадку, сторінка не завантажиться і робота застосунку буде обмеженою та безрезультативною, адже це є однією з його головних функцій.

Сама сторінка виглядає наступним чином:

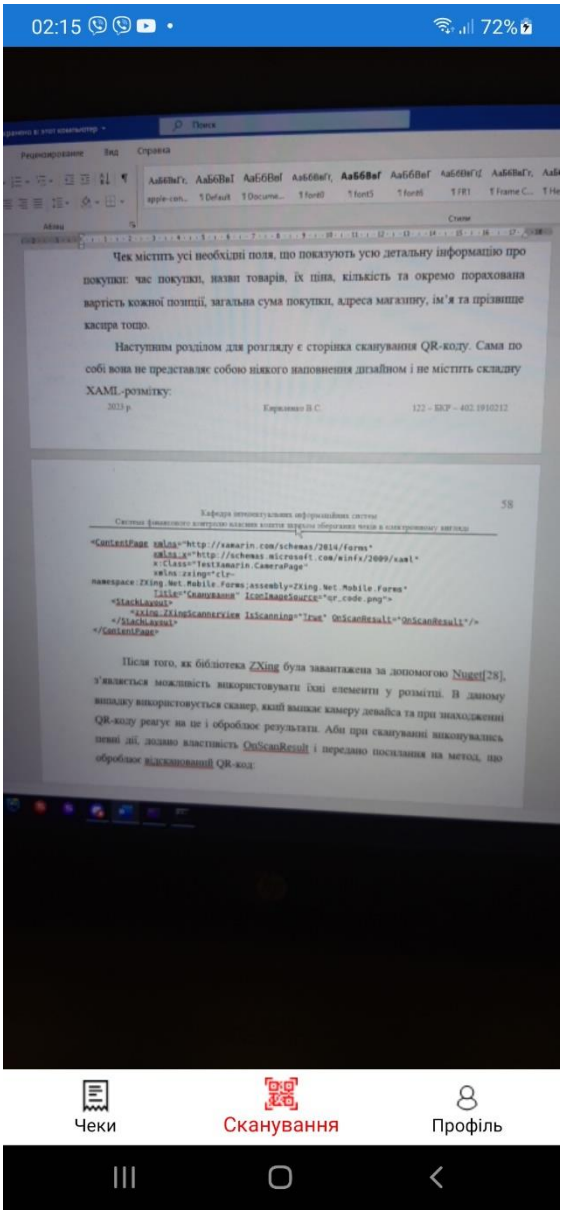

Рисунок 3.11 – Сторінка сканування QR-коду

Після успішного сканування, застосунок повертає користувача на головну сторінку із чеками (де, власне, з’являється вже новий чек) із повідомленням про успіх:

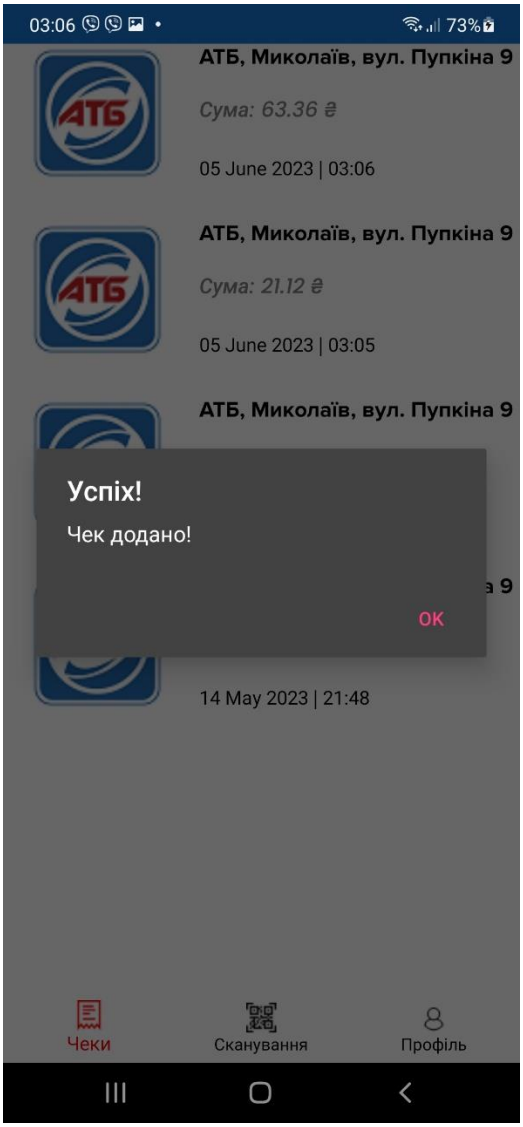

Рисунок 3.12 – Повідомлення про успішне сканування QR-коду

В разі, якщо було відсканировано QR-код, що не підтримується застосунком (тобто такий, що не містить GUID або містить щось ще окрім нього), то застосунок повідомляє користувача про неуспішне сканування з детальним поясненням причини:

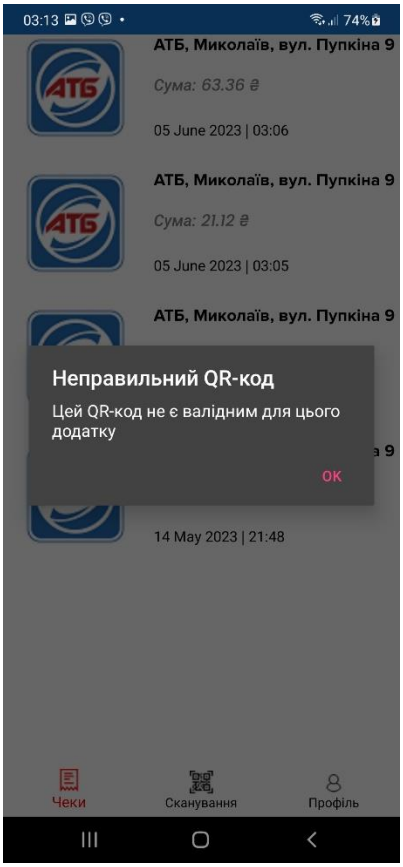

Рисунок 3.13 – Приклад неуспішного сканування QR-коду

В такому випадку, застосунок в будь-якому випадку може застерегти користувача про неправильне його використання, а також, це може допомогти запобігти майбутнім помилкам, в разі, якщо користувач спробує відсканувати QR-код, що не підтримується системою, що могло б призвести до непередбачуваних наслідків.

Рисунок 3.14 – Приклад QR-коду, що не підлягає обробці

Останнім розділом є сторінка профілю. Величезної різниці у підході до розмітки або використання серверного функціоналу немає, проте сам дизайн має певні відмінності, хоча й зберігає схожі риси сторінок реєстрації та логіну, а також одночасно поєднує дизайн сторінки списку чеків. На рис. 3.15 це добре спостерігається.

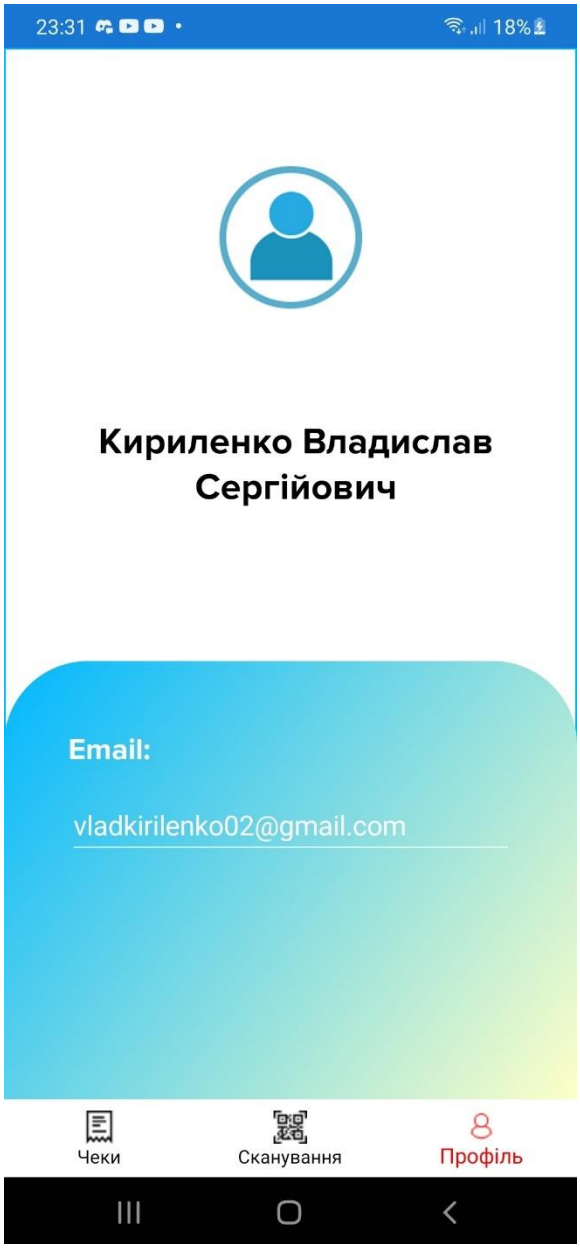

Рисунок 3.15 – Сторінка профілю користувача

На сторінці зображений ПІБ користувача та його email – тобто невелика інформаційна сторінка. Серед раніше згаданих нових рис дизайну – прямокутник, що має заокруглені кути лише зверху. Така фігура не підтримується стандартним дизайном Xamarin, але підтримується кастомний рендеринг, тобто розробник може на базі існуючих елементів дизайну створювати свої, що матимуть власні нові властивості. Так як прямокутники в стандартному дизайні Xamarin не мають

можливості вибирати кути для округлення (а лише округлюють усі), то довелось впроваджувати власний елемент:

```
protected override void OnElementChanged(ElementChangedEventArgs<Frame> e)
        {
            base.OnElementChanged(e);

            if (e.NewElement != null && Control != null)
            {
                UpdateCornerRadius();
            }
        }

        protected override void OnElementPropertyChanged(object sender,
PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName == nameof(CustomFrame.CornerRadius) ||
                e.PropertyName == nameof(CustomFrame))
            {
                UpdateCornerRadius();
            }
        }

        private void UpdateCornerRadius()
        {
            if (Control.Background is GradientDrawable backgroundGradient)
            {
                var cornerRadius = (Element as CustomFrame)?.CornerRadius;
                if (!cornerRadius.HasValue)
                {
                    return;
                }

                var topLeftCorner = Context.ToPixels(cornerRadius.Value.TopLeft);
                var topRightCorner = Context.ToPixels(cornerRadius.Value.TopRight);
                var bottomLeftCorner =
Context.ToPixels(cornerRadius.Value.BottomLeft);
                var bottomRightCorner =
Context.ToPixels(cornerRadius.Value.BottomRight);

                var cornerRadii = new[]
                {
                    topLeftCorner,
                    topLeftCorner,

                    topRightCorner,
                    topRightCorner,

                    bottomRightCorner,
                    bottomRightCorner,

                    bottomLeftCorner,
                    bottomLeftCorner,
                };
```

```
            backgroundGradient.SetCornerRadii(cornerRadii);
        }
    }
```

В цьому коді представлено заміщення стандартної рамки Frame на CustomFrame. Для цього необхідно перевизначити дві функції: OnElementChanged та OnElementPropertyChanged. Це робиться для того, аби елемент або його властивості могли реагувати на зміни.

3.3 Програмна реалізація API сервісу

Дуже важливу роль у роботі застосунку відіграв саме API сервіс, який дає можливість працювати БД. Саме за допомогою цього сервісу, мобільний застосунок обмінюються із БД різними даними та виконує суттєву кількість операцій. Сам сервіс представлений у вигляді одного контроллера, що має певну кількість кінцевих точок (англ. – endpoints[29]) – по одному на кожне завдання. Також, API сервіс виконує важливу роль додаткового слою кіберзахисту, адже маючи сервісний функціонал, що винесено за межи основного застосунку покращує (хоч і зменшує продуктивність та швидкодію застосунку внаслідок формування запитів та відповідей сервісу, що забирає певний час, а не пряме виконання коду без запитів) кіберзахист застосунку, адже отримати код такого сервісу важко, лише можна намагатися направляти власні запроси до сервісу, з метою отримати певні результати, що не передбачені штатною роботою даного сервісу.

Переходячи до конкретної програмної реалізації API сервісу, варто зазначити, що сам контроллер не має реалізацію кожної кінцевої точки, проте в кожній викликає метод мікросервісу, що й виконує необхідні інструкції. Для розуміння, як працює контроллер, достатньо лише представлення однієї кінцевої точки, адже усі вони мають однакову структуру, лише посилаються на різні методи мікросервісу. Також вони мають однакову обробку статус-коду, що сигналізує про успіх або помилки (в даному випадку, реалізація включає статус-коди 200 OK та 500 Internal Server Error). Нижче приклад:

```
[HttpPost("create-user")]
        public async Task<IActionResult> CreateUser(CreateUserModel model)
        {
            try
            {
                await _localDbServices.CreateUserAsync(model);

                return Ok();
            }
            catch (Exception)
            {
                return StatusCode(500, "An unexpected error occurred");
            }
        }
```

В даному прикладі є використання атрибутів – інструкції в квадратних дужках над назвою методу. Цей атрибут сигналізує про тип методу (POST-метод зазвичай відправляє певні параметри у тілі або URL і не повертає ніяких результатів, окрім статус-коду) та власне, його назву. Ця назва враховується при виклику сервісу через URL, наприклад: http:/localhost:3000/api/localdb/create-user. Кожна така кінцева точка обгортається в try-catch блок задля того, аби програма продовжувала свою дію після перехоплення помилок. Також, задля того, аби контроллер міг використовувати мікросервіс, треба його додати до конструктора, як на прикладі:

```
private readonly ILocalDbServices _localDbServices;

        public LocalDbController(ILocalDbServices localDbServices)
        {
            _localDbServices = localDbServices;
        }
```

Саме значення мікросервісу передається до конструктора контроллеру за допомогою Dependency Injection (DI)[30]. Для цього необхідно в стартапі програми вказати посилання на інтерфейс та клас реалізації цього мікросервісу. Без цього контроллер не зможе отримати доступ до реалізації мікросервісу, що викличе певні помилки. Підсумовуючи, задля нормальної роботи API сервісу, необхідно провести певні підготовчі роботи, що включають в себе створення

контроллеру, створення мікросервісу, додавання його до DI тощо. Наступний код показує необхідні роботи в головному класі кожного вебзастосунку Program:

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllers();
builder.Services.AddScoped<ILocalDbServices, LocalDbServices>();
var app = builder.Build();

app.MapGet("/", () => "Hello World!");
app.UseDeveloperExceptionPage();
app.UseRouting();

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllers();
});
app.Run();
```

Переходячи до розбору усіх методів мікросервіса, головним їхнім призначенням є саме операції з БД. Це завдання може вирішуватися декількома шляхами, наприклад Entity Framework або Dapper, тобто, за допомогою ORM[31]. В даному випадку, на проєкті було використано саме Dapper. Але різниця між двома вищевказаними системами не є великою, якщо говорити про продуктивність, хоча й принцип дії істотно відрізняється. Entity Framework може працювати за двома принципами: code-first та database-first[32]. Перший працює таким чином, що спочатку у серверному функціоналі пишуться моделі, які потім за допомогою міграції будують таблиці, зв’язки, колонки в БД. Другий принцип працює навпаки – спочатку створюються таблиці в БД, які потім мігрують до серверного функціоналу в ролі класів моделей. Якщо говорити про Dapper, то він працює досить простим чином – це бібліотека, яка допомагає відтворювати SQL-запроси на серверному функціоналі в БД і отримувати необхідні результати. Цей варіант є дуже зручним для тих, кому більше подобається (або більш обізнаний у SQL, ніж у back-end програмуванні). Тобто, замість того, щоб створювати моделі та міграції, достатньо посилати запроси до БД через серверну частину програми.

Якщо порівнювати практичність цих принципів, то перший спосіб краще підійде для тих програм, які не є вибагливими щодо операцій із БД, тобто не

потребують складних SQL запитів. В свою чергу, Dapper є більш зручним для тих, хто не любить користуватися міграціями.

Перший метод, що реєструє користувача, не є складним, тому що просто створює запис із наданих даних під час реєстрації:

```
public async Task CreateUserAsync(CreateUserModel model)
        {
            _dbConnection.Open();

            await _dbConnection.ExecuteAsync(@"INSERT Users (Id, FirstName, Surname,
MiddleName, Email)
                                                VALUES (DEFAULT, @firstName,
@surname, @middleName, @email)", new
            {
                firstName = model.FirstName,
                surname = model.Surname,
                middleName = model.MiddleName,
                email = model.Email
            });

            _dbConnection.Close();
        }
```

При цьому, в усіх цих методах зберігається певна логіка – існує публічна властивість в мікросервісі як dbConnection, що на початку метода запускає з’єднання, а в кінці – роз’єднує програму і БД. З мобільного застосунку передається модель, що передається в SQL запит в якості параметрів.

Аналізуючи вже згадану раніше сторінку профіля, у якої небагато інформації, але все ж, вона підтягується саме з БД, а не зберігається в застосунку.

```
var result = await _dbConnection.QueryFirstOrDefaultAsync<ProfileInfoModel>(@"SELECT
CONCAT(u.Surname, ' ', u.FirstName, ' ', u.MiddleName) FullName, u.Email

FROM Users u

WHERE u.Id = @userId", new
            {
                userId
            });
```

Серед незвичайного тут лише використання стандартної SQL-функції CONCAT(), що поєднує varchar (текстові значення в SQL) значення між собою в одне єдине значення. Так як ПІБ користувача не є єдиною колонкою, то задля

нормального відображення на сторінці профіля користувача необхідно поєднувати їх в одне ціле. Простіше це зробити одразу SQL-запитом, аніж збирати по частинах декількома запитами SQL, тим паче, що це б негативно вплинуло на продуктивність сервісу, так як більша кількість запитів пропорціонально зменшує швидкодію.

Для отримання даних про чеки та магазини (для відображення коректної інформації на головній сторінці чеків), використовується наступний метод:

```
var result = await _dbConnection.QueryAsync<ReceiptsShopsModel>(
                @"SELECT r.Id, r.CreatedOn, r.Total, s.Name ShopName, s.ImagePath
ShopImagePath FROM Receipts r
                    JOIN Shops s ON r.ShopId = s.Id
                    WHERE r.CustomerId = @userId
                    ORDER BY r.CreatedOn DESC", new
                {
                    userId
                }) as IList<ReceiptsShopsModel>;
```

Можна побачити, що в даному коді використовується з’єднання з іншою таблицею. Це також скорочує кількість запитів, адже дозволяє в одній вибірці отримати дані одразу з кількох таблиць одночасно. Також, для більшої зручності, запит також робить сортування по даті створення чеку, що допомагає користувачеві зорієнтуватися швидше, так як нові чеки знаходяться вище, старі, відповідно, нижче.

Створення QR-коду в даній роботі є лише допоміжною частиною, отже, функціонал, який написаний для відтворення цього QR-коду лише імітує його наявність і запускається за допомогою програми Postman – програми, яка допомагає робити запити до різних API-сервісів.

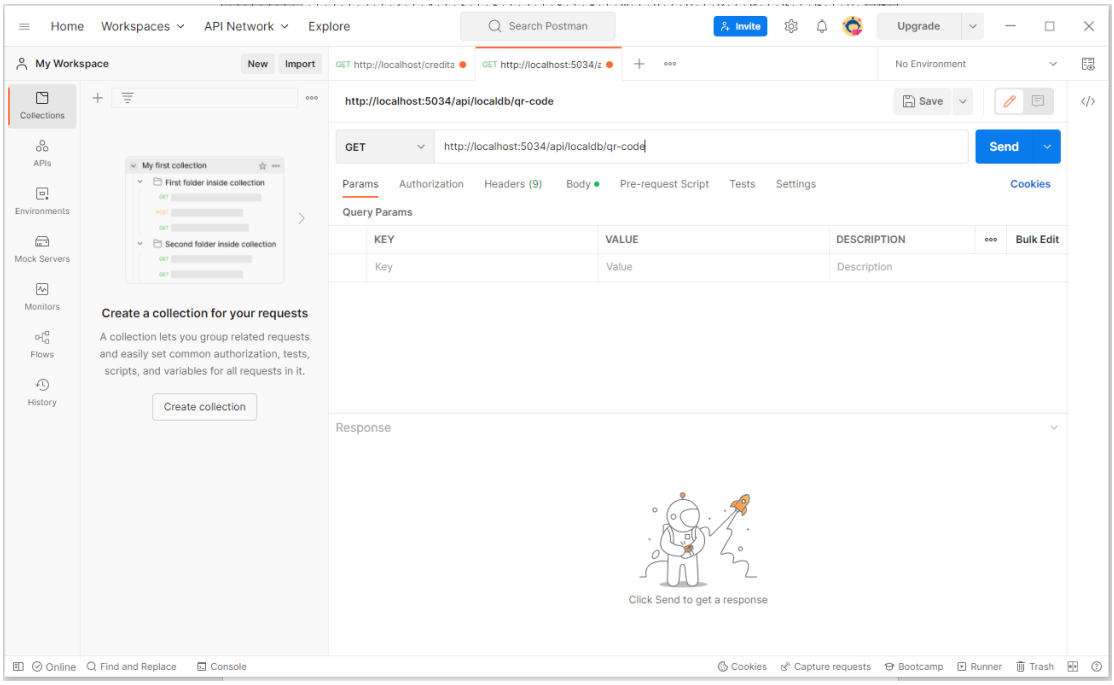

Рисунок 3.16 – Інтерфейс програми Postman

Для успішної імітації створення QR-коду, необхідно ввести коректну URL адресу, виставити типу методу GET та через вкладку «Body» додати тіло методу, що передасть необхідні параметри.

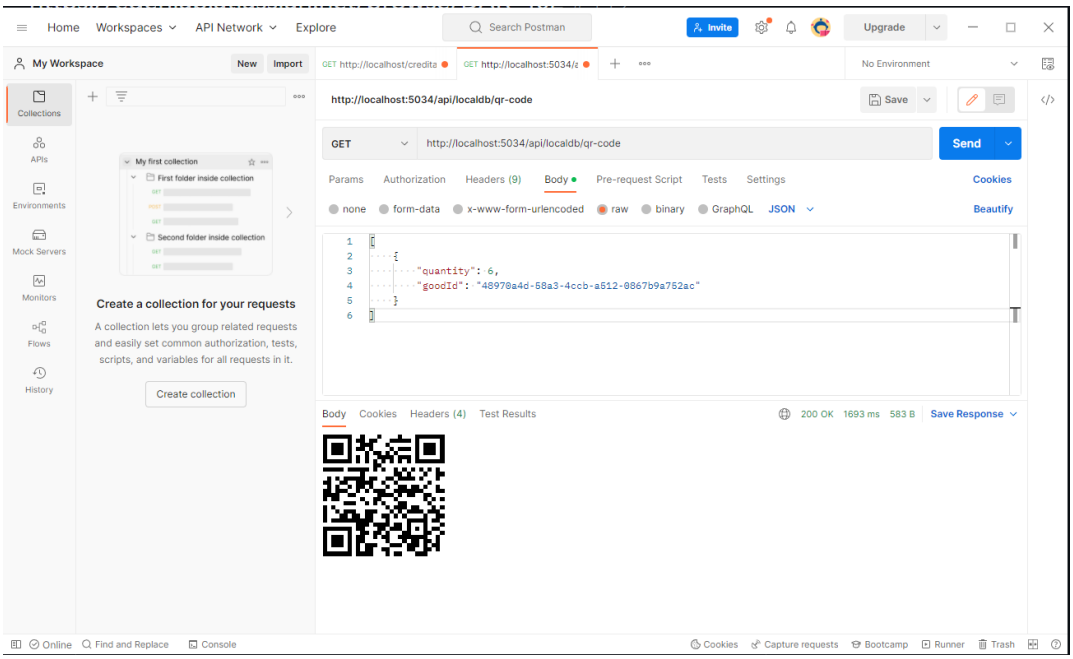

Рисунок 3.17 – Створення QR-коду за допомогою програми Postman

Передавши необхідні значення у тіло метода, необхідно натиснути на кнопку «Send», що відправить запит до сервісу та поверне необхідний результат, разом із HTTP-кодом. При цьому, в коді обробка виглядає так:

```
public async Task<byte[]> CreateQRCodeLinkAsync(IList<GoodsBuyingModel> list)
        {
            var client = new HttpClient();
            _dbConnection.Open();

            var receiptId = Guid.NewGuid();
            var shopId = Guid.Empty;
            var total = 0.0f;
            await _dbConnection.ExecuteAsync(@"INSERT Receipts (Id, CustomerId,
ShopId, CashierName, CreatedOn, Total, GoogleDriveId)
                                                    VALUES (@id, NULL, NULL,
'Воронова Діана', SYSDATETIME(), DEFAULT, NULL)", new
            {
                id = receiptId
            });
            foreach (var item in list)
            {
                var good = await
_dbConnection.QueryFirstOrDefaultAsync<GoodsModel>(@"SELECT * FROM Goods WHERE Id =
@goodId", new { goodId = item.GoodId });
                shopId = good.ShopId;
                await _dbConnection.ExecuteAsync(@"INSERT GoodsHistory (Id, GoodId,
ReceiptId, Quantity, PurchaseDate)
                                                    VALUES (DEFAULT, @goodId,
@receiptId, @quantity, SYSDATETIME())", new
                {
                    goodId = item.GoodId,
                    receiptId,
                    quantity = item.Quantity
                });
                var goodTotal = good.Price * item.Quantity;
                total += goodTotal;
            }

            await _dbConnection.ExecuteAsync(@"UPDATE Receipts
                                                SET ShopId = @shopId,
                                                   Total = @total
                                                WHERE Id = @receiptId", new
            {
                shopId,
                total,
                receiptId
            });
            _dbConnection.Close();
            return await
client.GetByteArrayAsync($"https://api.qrserver.com/v1/create-qr-
code/?size=150x150&data={receiptId}");
        }
```

На вхід до методу передається список куплених користувачем товарів. Але спочатку створюється сам чек у БД, з майже пустими значеннями. Далі, поступово вартість кожного товару обраховується і сумується загальна вартість до тих пір, доки не підрахуються усі товари зі списку, що були передані до методу в якості параметру. Далі, інформація завантажується у таблицю GoodsHistory, за допомогою якої надалі формуватиметься чек. Далі прив'язується магазин, в якому відбулася покупка через інформацію у товарі. GUID створеного чеку передається до іншого сервісу – QRServer (сервіс, що створює QR-коди), який приймає будь-яку текстову інформацію.

Наступний метод реалізовує прив'язку документа до користувача – спочатку в БД за допомогою підстановки GUID користувача, потім – ланцюжок методів для створення, відображення та завантаження електронного чеку.

Після прив'язки чеку до користувача через GUID, починається робота над електронним чеком. Спочатку отримується необхідна інформація про покупки, магазин, товари, користувача. Потім, за допомогою існуючого шаблону чека у вигляді документу Microsoft Word та заміні макросів створюється чек відповідно до отриманих раніше даних:

```
var goodsModelList = await _dbConnection.QueryAsync<GoodsDocumentModel>(@"SELECT
g.Name, gh.Quantity, g.Price, g.Price * gh.Quantity Total FROM GoodsHistory gh

JOIN Goods g ON gh.GoodId = g.Id

WHERE gh.ReceiptId = @receiptId", new
            {
                receiptId
            }) as IList<GoodsDocumentModel>;
            var template = DocX.Load(@"D:/receipt_template.docx");
            var table = template.Tables.FirstOrDefault();
            if (table != null)
            {
                for (var i = 0; i < goodsModelList?.Count; i++)
                {
                    table.Rows[i + 6].Cells[0].Paragraphs.First().Append($"{i}");
                    table.Rows[i +
6].Cells[1].Paragraphs.First().Append($"{goodsModelList[i].Name}");
                    table.Rows[i +
6].Cells[2].Paragraphs.First().Append($"{goodsModelList[i].Quantity}");
                    table.Rows[i +
6].Cells[3].Paragraphs.First().Append($"{goodsModelList[i].Price}");
```

```
                    table.Rows[i +
6].Cells[4].Paragraphs.First().Append($"{goodsModelList[i].Total}");
                }
            }
            var fields = new Dictionary<string, string>
            {
                { "%ShopName%", documentModel.ShopName },
                { "%CreatedOn%", documentModel.CreatedOn.ToString("dd.MM.yyyy |
HH:mm") },
                { "%CashierName%", documentModel.CashierName },
                { "%Total%",
documentModel.Total.ToString(CultureInfo.InvariantCulture) },
                { "%CustomerName%", $"{documentModel.Surname}
{documentModel.FirstName} {documentModel.MiddleName}" }
            };

            foreach (var field in fields)
            {
                template.ReplaceText(new StringReplaceTextOptions
                {
                    SearchValue = field.Key,
                    NewValue = field.Value
                });
            }

            template.SaveAs(@$"D:/{receiptId}.docx");
```

Змінений документ зберігається на сервері, у його файловому сховищі. Документ має назву відповідно до GUID чеку в БД. Наступним кроком в обробці документа стає завантаження файлу до хмарного сховища і проходження необхідних перевірок:

```
await using var stream = File.OpenRead(@$"D:/{receiptId}.docx");
            var service = GetService();
            var request = service.Files.List();
            var results = await request.ExecuteAsync();
            var fileId = string.Empty;
            foreach (var file in results.Files)
            {
                if (file.Name == $"{documentModel.UserId}" && file.MimeType ==
"application/vnd.google-apps.folder")
                {
                    fileId = await UploadFile(file.Id, stream, $"{receiptId}");
                }
            }

            if (string.IsNullOrEmpty(fileId))
            {
                var parentId = CreateFolder($"{documentModel.UserId}");
                fileId = await UploadFile(parentId, stream, $"{receiptId}");
            }

            await _dbConnection.ExecuteAsync(@"UPDATE Receipts
                                               SET GoogleDriveId = @fileId
```

```
                                        WHERE Id = @receiptId", new
{
    fileId,
    receiptId
});
```

Новостворений файл відкривається асинхронно, у вигляді потоку. За допомогою методу, що отримує екземпляр сервісу Google Drive (за допомогою спеціальних даних як на рис. 2.4), можна звертатися до API. Відбувається перевірка на наявність папки (яка має назву відповідно до GUID користувача), відповідно, якщо є, то чек завантажується в існуючу папку, якщо немає – створюється, після чого відбувається завантаження в цю папку.

Хоча й API підтримує можливість надання та зняття прав для певних користувачів, папок або файлів, але для досягнення мети роботи, достатньо було батьківській папці надати права на читання усім, але не надавати при цьому права на редагування, коментування або видалення:

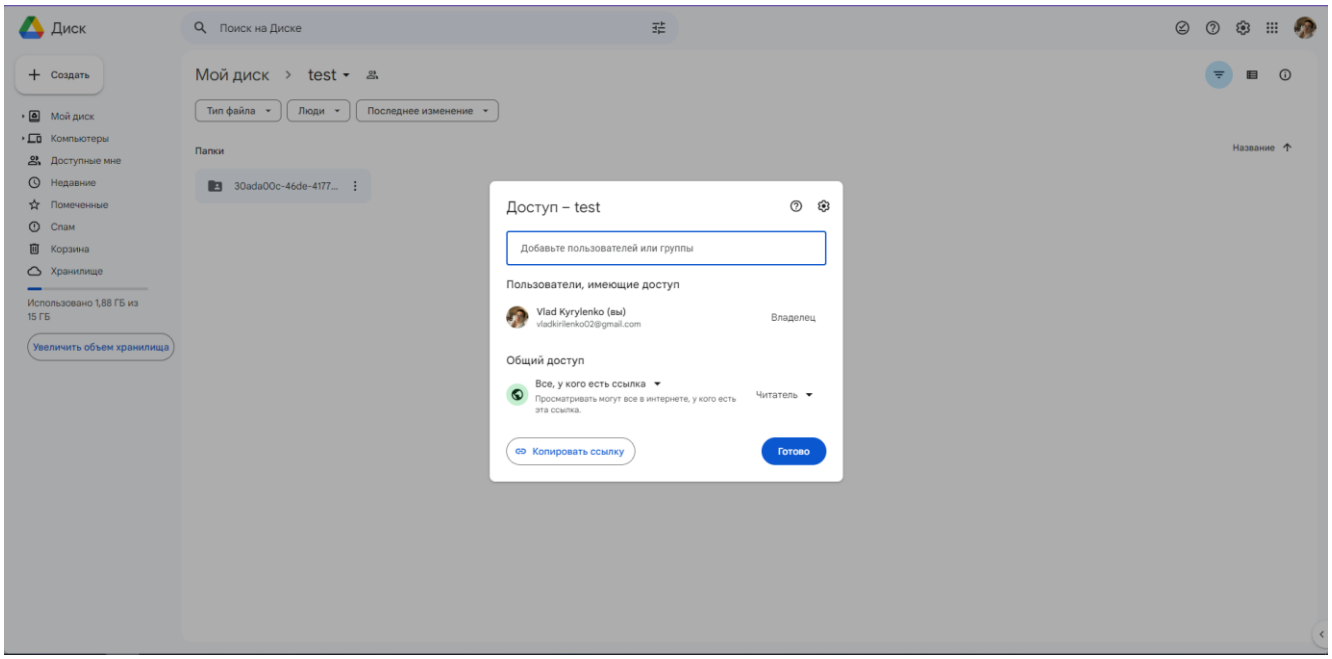

Рисунок 3.18 – Права доступу батьківської папки

В Google Drive присутнє делегування прав, що означає, що якщо батьківська папка має певні права, такі ж самі передаються до створених внутрішніх папок і файлів по замовчуванню, але ці права можна змінювати.

Також на рис. 3.18 можна побачити, як працює ієрархія в хмарному сховищі – головна папка містить папки для користувачів, а всередині тих папок знаходяться самі електронні чеки, як на наступному рисунку.

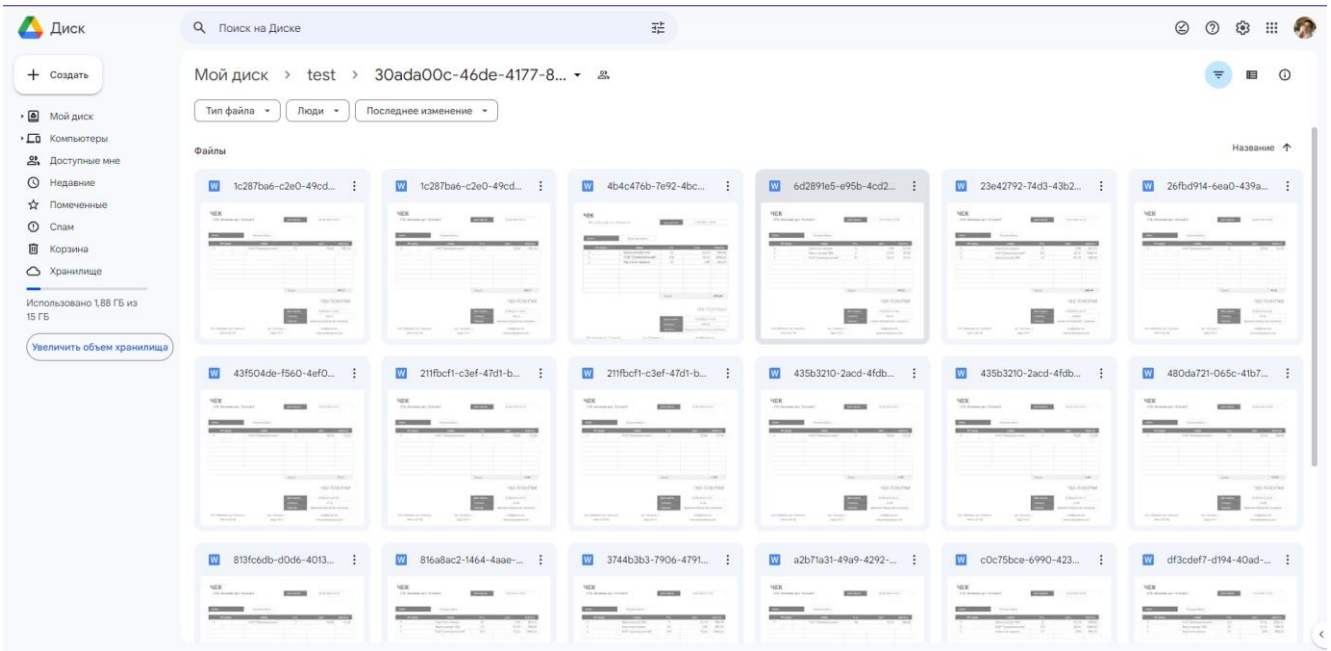

Рисунок 3.19 – Зберігання електронних чеків у хмарному сховищі

Таким чином, створений файл має свій певний Google Drive ID, який використовується для перегляду або редагування. Цей ID зберігається в окремій колонці таблиці Receipts для кожного запису. За допомогою цього ID формується посилання на файл і відкривається чек в мобільному застосунку.

Висновки до розділу 3

Програмна реалізація є найважливішою частиною для досягнення мети роботи та дослідження. В цьому розділі було повністю розкрито програмну реалізацію мобільного застосунку та API сервісу, який застосовувався для обміну даними між БД та мобільним застосунком. Також, метою написання даного розділу було створення путівника по користуванню мобільним застосунком, що й було досягнено.

Опис програмної реалізації застосунку є важливою частиною для розуміння користувачами, експертами, аналітиками та іншими, як саме працює застосунок, які завдання може виконувати та за яких умов і вхідних даних. Демонстрація реалізації допомагає створити широку картину і враження про застосунок.

Застосунок є основним інструментом для внеску у дослідження проблем і завдань даної роботи. За результатами розробки можна з’ясувати, що поставлене завдання виконано, а сама реалізація може бути повністю або частково використана для модернізації даного рішення або його використання для вирішення цієї проблеми на державному та законодавчому рівні.

ВИСНОВКИ

Розроблена система фінансового контролю, яка використовує збереження чеків в електронному вигляді, успішно відповідає поставленим цілям і завданням. Вона надає зручний та ефективний спосіб контролювати особисті фінанси, дозволяючи користувачам зберігати, організовувати та швидко отримувати доступ до своїх чеків.

Використання електронного збереження чеків дозволяє зменшити ризик втрати чеків, покращити точність обліку та забезпечити зручний доступ до фінансової інформації.

Використання мобільного застосунку та сервісу, які зв'язуються з базою даних, дозволяє користувачам зручно сканувати та зберігати чеки. Результати експериментального дослідження підтверджують ефективність і корисність розробленої системи у порівнянні з традиційними методами контролю власних коштів. Поточні результати можуть бути використані для створення загального єдиного державного рішення поставленого завдання.

В процесі роботи було виявлено певні обмеження, такі як обмежена доступність інформації про чеки в електронному вигляді та потреба в розвитку відповідних правових норм і стандартів для електронного збереження чеків. Для подальшого розвитку системи можуть бути розглянуті такі напрямки досліджень, як розширення функціональності мобільного застосунку, впровадження технологій штучного інтелекту для автоматичного аналізу фінансових даних та дослідження впливу системи на фінансову поведінку користувачів, а також впровадження необхідних правових норм та законів для використання подібних рішень і загального користування електронними чеками.

Мета даної роботи полягала у вирішенні проблеми централізованого зберігання електронних чеків в одному застосунку, при цьому ще й відповідності застосунку законодавству України, а також показати прототип застосунку для вирішення поставленої задачі. Мета роботи досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Lb.ua. Набув чинності закон про електронні чеки та гарантії на придбану техніку: вебсайт. URL: https://lb.ua/economics/2021/07/25/490144_nabuv_chinnosti_zakon_pro_elektroni.html (дата звернення 15.05.2023)
2. Національний банк України. Обсяги безготівкових розрахунків в Україні зростають, попри війну: вебсайт. URL: https://bank.gov.ua/ua/news/all/obsyagi-bezgotivkovih-rozrahunkiv-v-ukrayini-zrostayut-popri-viynu (дата звернення 15.05.2023)
3. Кількість користувачів смартфонів в Україні збільшилася до 85% — дослідження: вебсайт. URL: https://ms.detector.media/mediadoslidzhennya/post/21573/2018-08-03-kilkist-korystuvachiv-smartfoniv-v-ukraini-zbilshylasya-do-85-doslidzhennya/#:~:text=%D0%97%D0%B0%20%D1%97%D1%85%20%D0%B4%D0%B0%D0%BD%D0%B8%D0%BC%D0%B8%2C%2085%25%20%D1%83%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D1%86%D1%96%D0%B2,%D0%BC%D1%96%D1%81%D1%82%20%2D%2054%2D55%25 (дата звернення 16.05.2023)
4. Dext App. Documentation: вебсайт. URL: https://app.dext.com/ (дата звернення 10.05.2023)
5. Developer Guides: вебсайт. URL: https://developer.android.com/docs (дата звернення 09.05.2023)
6. Android Developers Blog: вебсайт. URL: https://android-developers.googleblog.com/ (дата звернення 09.05.2023)
7. Android developer portal with tools, libraries, and apps: вебсайт. URL: https://android-arsenal.com/ (дата звернення 09.05.2023)
8. Android Weekly – Free weekly Android & Kotlin development newsletter: вебсайт. URL: https://androidweekly.net/ (дата звернення 10.05.2023)

9. Developers Forum: вебсайт. URL: https://stackoverflow.com/questions/tagged/android (дата звернення: 12.05.2023)
10. Online Courses, Learning Paths, and Certifications for Android: вебсайт. URL: https://www.pluralsight.com/courses/android-layout-fundamentals (дата звернення 12.05.2023)
11. Android Programming: The Big Nerd Ranch Guide (5th Edition): вебсайт. URL: https://bignerdranch.com/books/android-programming-the-big-nerd-ranch-guide-5th-edition/ (дата звернення 15.05.2023)
12. Android Developers Channel: вебсайт. URL: https://www.youtube.com/user/androiddevelopers (дата звернення 14.05.2023)
13. Xamarin.Forms Documentation: вебсайт. URL: https://learn.microsoft.com/uk-ua/xamarin/xamarin-forms/ (дата звернення 05.05.2023)
14. Xamarin.Forms Github Official Repository: вебсайт. URL: https://github.com/xamarin/Xamarin.Forms (дата звернення: 05.05.2023)
15. Microsoft Q&A. Xamarin: вебсайт. URL: https://learn.microsoft.com/en-us/answers/tags/18/xamarin (дата звернення: 04.05.2023)
16. Xamarin Blog: вебсайт. URL: https://devblogs.microsoft.com/xamarin/ (дата звернення: 07.05.2023)
17. Online Courses, Learning Paths, and Certifications for Xamarin.Forms: вебсайт. URL: https://www.pluralsight.com/paths/building-cross-platform-apps-with-xamarin-forms (дата звернення: 02.05.2023)
18. iOS чи Android: яку платформу обрати для розробки. вебсайт. URL: https://kitapp.pro/uk/ios-chi-android-yaku-platformu-obrati-dlya-rozrobki/ (дата звернення: 15.04.2023)
19. Anthony Molinaro SQL Cookbook: Query Solutions and Techniques for Database Developers. Covers SQL Server, PostgreSQL, Oracle, MySQL, and DB2. O'Reilly and Associates; 1st edition, Jan 1, 2006. 633 p.

20. Michael J. Hernandez Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design. Addison-Wesley Professional; 2nd edition May 15, 2003. 611 p.
21. Урок 2. Firebase. Створення, налаштування та підключення в Xamarin.Forms застосунку: вебсайт. URL: https://www.youtube.com/watch?v=iqC8sDXCWXk (дата звернення 01.05.2023)
22. Firebase Documentation: вебсайт. URL: https://firebase.google.com/docs/?hl=en&authuser=0 (дата звернення 02.05.2023)
23. Введение в API Google Диска: вебсайт. URL: https://developers.google.com/drive/api/guides/about-sdk?hl=ru (дата звернення: 09.05.2023)
24. How to create Google Oauth2 web application credentials in 2021: вебсайт. URL: https://www.youtube.com/watch?v=pBVAyU4pZOU&list=PLws6LF6K-IU-zzbgwzyHh9IQ1zCpBwwnf&index=3 (дата звернення: 10.05.2023)
25. SQL Server connection strings: вебсайт. URL: https://www.connectionstrings.com/sql-server/ (дата звернення: 29.04.2023)
26. GUID: вебсайт. URL: https://uk.wikipedia.org/wiki/GUID (дата звернення 01.05.2023)
27. Keith J. Grant CSS in Depth. Manning; First Edition April 7, 2018. 472 p.
28. An introduction to NuGet: вебсайт. URL: https://learn.microsoft.com/en-us/nuget/what-is-nuget (дата звернення 03.05.2023)
29. Jim Webber, Savas Parastatidis, Ian Robinson REST in Practice: Hypermedia and Systems Architecture. O'Reilly Media; 1st edition October 12, 2010. 448 p.
30. What is dependency injection?: вебсайт. URL: https://www.techtarget.com/searchapparchitecture/definition/dependency-injection (дата звернення: 05.06.2023)
31. ORM (Object Relation Mapping): автоматизація запису даних: вебсайт. URL: https://www.hwlibre.com/uk/%D1%80%D0%B5%D0%BB%D1%8F%D1%86%D1%96%D0%B9%D0%BD%D0%B5-

%D0%B2%D1%96%D0%B4%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BE%D0%B1%E2%80%99%D1%94%D0%BA%D1%82%D0%B0-orm/ (дата звернення 07.06.2023)

32. Julia Lerman, Rowan Miller Programming Entity Framework: Code First: Creating and Configuring Data Models from Your Classes. O'Reilly Media; 1st edition December 20, 2011. 194 p.