МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ОФОРМЛЕННЯ ЗАМОВЛЕНЬ НА МОВІ ПРОГРАМУВАННЯ KOTLIN»**

Виконав: студент 2 курсу, групи 8.1219

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми інженерія програмного забезпечення
(назва освітньої програми)

Є.К. Ходус
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.–м.н., доцент
Лісняк А.О.
(підпис)

« » 2020 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Ходусу Євгену Костянтиновичу
(прізвище, ім'я та по батькові)

1. Тема роботи Розробка мобільного додатку оформлення замовлень на мові програмування Kotlin

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » травня 2020 року № 756–с

2. Строк подання студентом роботи 30.11.2020

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково–пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Реалізація програмного забезпечення
4. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	04.06.2020	Виконано
2.	Збір вихідних даних.	15.06.2020	Виконано
3.	Обробка методичних та теоретичних	20.07.2020	Виконано
	джерел.		
4.	Розробка першого та другого розділу.	31.08.2020	Виконано
5.	Розробка третього розділу.	14.10.2020	Виконано
6.	Оформлення та нормоконтроль	30.11.2020	Виконано
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	16.12.2020	Виконано

Студент _____ Є.К. Ходус
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О. В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка мобільного додатку оформлення замовлень на мові програмування Kotlin»: 51 с., 39 рис., 11 джерел.

ANDROID, МОБІЛЬНИЙ ДОДАТОК, KOTLIN, ANDROID STUDIO, FIREBASE, GRADLE, ЕЛЕКТРОННА КОМЕРЦІЯ.

Об’єкт дослідження: можливості мови програмування Kotlin та хмарної бази даних Firebase.

Мета роботи: розробка мобільного додатку для оформлення замовлень на мові програмування Kotlin, демонстрація переваг використання цієї мови.

Методи дослідження: експериментально–теоретичний.

В кваліфікаційній роботі розглядаються основні переваги використання мови програмування Kotlin та хмарної СУБД Firebase, продемонстровані підходи та практики у створенні додатків для операційної системи Android.

Результати можуть бути використані для розробки комерційних мобільних додатків.

SUMMARY

Masters’s Qualifying Thesis «Development of the Kotlin mobile application for ordering»: 51 pages, 39 figures, 11 references.

ANDROID, MOBILE APP, KOTLIN, ANDROID STUDIO, FIREBASE, GRADLE, ELECTRONIC COMMERCE.

The objects of the study are Kotlin programming language and Firebase cloud database capabilities.

The aim of the study is development of a mobile application for ordering in the Kotlin programming language, demonstration of the benefits of using this language.

The methods of research are experimental, theoretical.

The qualification work considers the main advantages of using the programming language Kotlin and cloud database Firebase, demonstrates approaches and practices in creating applications for the Android operating system.

Results can be used to develop commercial mobile applications.

ЗМІСТ

ВСТУП

З розвитком технологій використання смартфонів у повсякденному житті стає дедалі поширенішим. В наш час майже у кожної дорослої людини є свій смартфон. Різні мобільні додатки дозволяють значно спростити наше життя та вирішувати певні проблеми. Вони мають застосування майже у всіх областях нашого життя.

У зв’язку з пандемією та карантинними обмеженнями, підприємцям стає все більш невигідно сплачувати оренду приміщень, заробітну плату найманим працівникам та інші витрати свого бізнесу. Для отримання додаткового потоку клієнтів і автоматизації роботи ідеальним порятунком для підприємців буде створення мобільного додатку для оформлення замовлень.

Сучасні інструменти розроблення мобільних додатків дозволяють розробникам реалізувати велику кількість ідей для різних цілей. В тому числі і використання смартфона в ролі засобу для здійснення закупівель.

Супермаркет, що не потребує інвестицій в торгівельну залу, великого штату касирів, вантажників, охоронців та мерчендайзерів – вигідна бізнес модель, що користується великим попитом як у великих мегаполісах, так і в невеликих провінційних містах.

На сьогоднішній день є певна кількість додатків в основі яких лежить створення покупок через Інтернет. Проте всі вони реалізовані на основі вже існуючих складних ERP–систем, що від самого початку не були пристосовані для роботи з мобільними додатками.

Отже виникла необхідність у створені сучасної, простої архітектури мобільного додатку інтернет комерції, що буде використовувати мінімум ресурсу сервера та обладнання клієнта, а також матиме можливість легкого масштабування у майбутньому. Оскільки більшість сучасних мобільних пристроїв працюють на базі операційної системи Android, то метою даної роботи є реалізація мобільного додатку для оформлення замовлень на базі цієї

операційної системи. Це забезпечує можливість поширити додаток серед великої кількості користувачів смартфонів, планшетів та персональних комп’ютерів.

Для досягнення даної мети були поставлені наступні завдання:

1. Проаналізувати предметну область і аналогічні проекти в мережі Інтернет.
2. Вивчити можливості мови програмування Kotlin та порівняти їх з мовою програмування Java.
3. Дослідити можливості Firebase Realtime Database.
4. Реалізувати мобільний додаток оформлення замовлень.
5. Провести тестування мобільного додатку.
6. Проаналізувати отриманий результат.

Практичне значення роботи полягає в отриманні мінімально життєздатного продукту, зручного та простого у використанні мобільного додатку з можливістю оформлення замовлень через мережу Інтернет, який може бути легко масштабованим у майбутньому.

1 АНАЛІЗ ВИМОГ ДЛЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Додатки для підприємств мобільних експрес–купівель

Розглянемо мобільні додатки на базі операційної системи Android з магазину Google Play, які призначені для підприємств мобільних експрес–купівель.

В ході пошуку не було знайдено аналогів в Україні. Збільшивши межі пошуку, було знайдено додатки для підприємств мобільних експрес– купівель, основними з яких є:

1) Furgoneta;
2) StreetEat;
3) Kater.

Далі був проведений огляд кожного з додатків для підготовки до порівняння.

Мобільний додаток «Furgoneta». Мобільний додаток «Furgoneta» належить однойменному бренду, підприємству мобільних експрес–купівель «Furgoneta» (Україна, Київ). Додаток є близьким за функціями в порівнянні із технічною характеристикою додатку, що розроблюється.

За допомогою даного мобільного додатку, можна знайти точки продажів, переглянути меню, зробити замовлення. Також передбачена можливість доставки на адресу.

При запуску програми користувач попадає на головний екран (див. рис. 1.1), на котрому вказані основні пункти меню, необхідні для ознайомлення з підприємством, продукцією підприємства, також для оформлення замовлення.

Рисунок 1.1 – Головний екран мобільного додатку «Furgoneta»

Для того, щоб знайти точки продажів слід зробити таку послідовність дій (див. рис. 1.2):

1) з головної сторінки перейти на бічну панель;

2) перейти по посиланню «Наші адреси»;

3) вибрати потрібну адресу, і натиснути на кнопку «На карті» для відкриття карти.

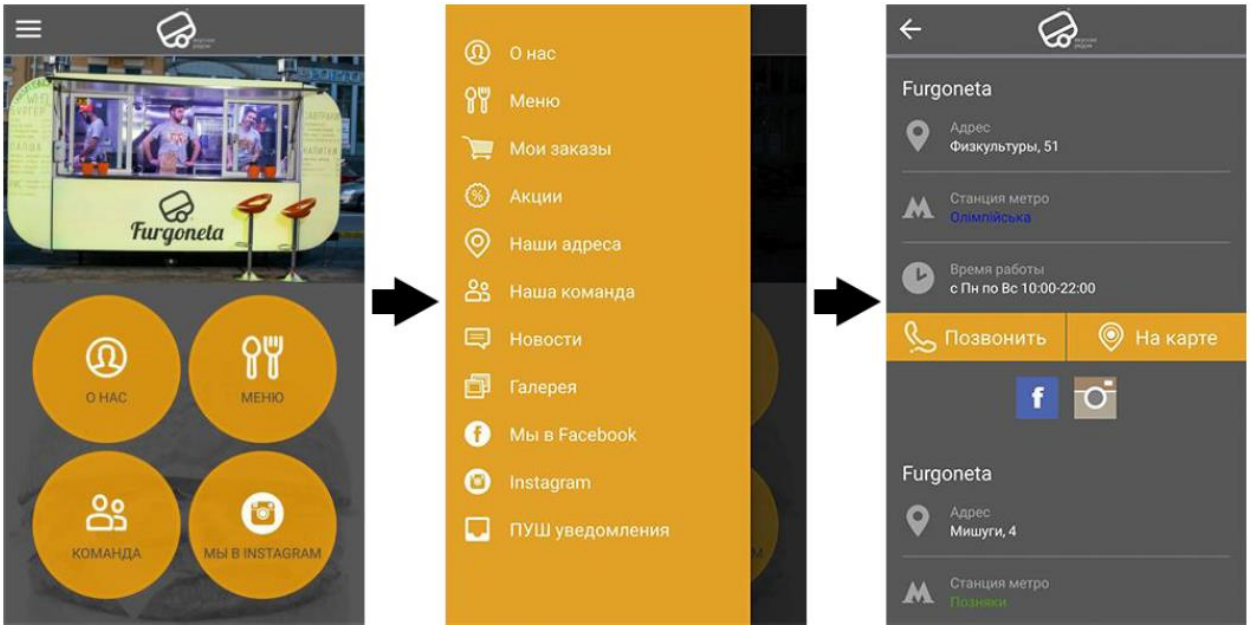

Рисунок 1.2 – Пошук точок продажу

Мобільний додаток «StreetEat». Мобільний додаток «StreetEat» є агрегатором підприємств мобільних експрес–купівель для таких країн, як Італія, Великобританія, Франція і Іспанія.

Цей додаток об'єднує інформацію про декількох підприємств мобільних експрес–купівель і має єдиний призначений для користувача інтерфейс. За допомогою цього додатку, можна знайти найближчі точки продажів різних брендів в залежності від наданої продукції. Так як додаток є агрегатором, то не можна зробити онлайн–замовлення.

На головному екрані мобільного додатка розташований перелік країн, що підтримуються у додатку (див. рис. 1.3).

Рисунок 1.3 – Головний екран мобільного додатка «StreetEat»

Для пошуку точки продажів слід виконати наступну послідовність дій (див. рис. 1.4):

1) на головному екрані вибрати країну;

2) обрати страву.

Після виконання даних дій, буде показана карта, з точками продажу різних підприємств мобільного експрес–харчування.

Рисунок 1.4 – Порядок дій для пошуку точок продажів

Мобільний додаток «Kater». Мобільний додаток «Kater» є єдиною системою замовлень для підприємств мобільного експрес–харчування в США.

Додаток об'єднує в собі різні підприємства мобільного експрес–харчування, в кожному з яких можна зробити онлайн–замовлення, використовуючи даний мобільний застосунок При створенні онлайн–замовлення, клієнт має можливість призначити готовність зробленого замовлення на зручний для нього час і забрати його самостійно.

При вході в мобільний додаток, користувач бачить список компаній, особистий кабінет, кошик замовлення (див. рис. 1.5). Для пошуку найближчих точок продажу, слід на головному екрані натиснути на кнопку «Map» (рис. 1.6). Буде показана карта з найближчими точками продажу всіх підприємств мобільного експрес харчування.

Рисунок 1.5 – Головний екран мобільного додатка «Kater»

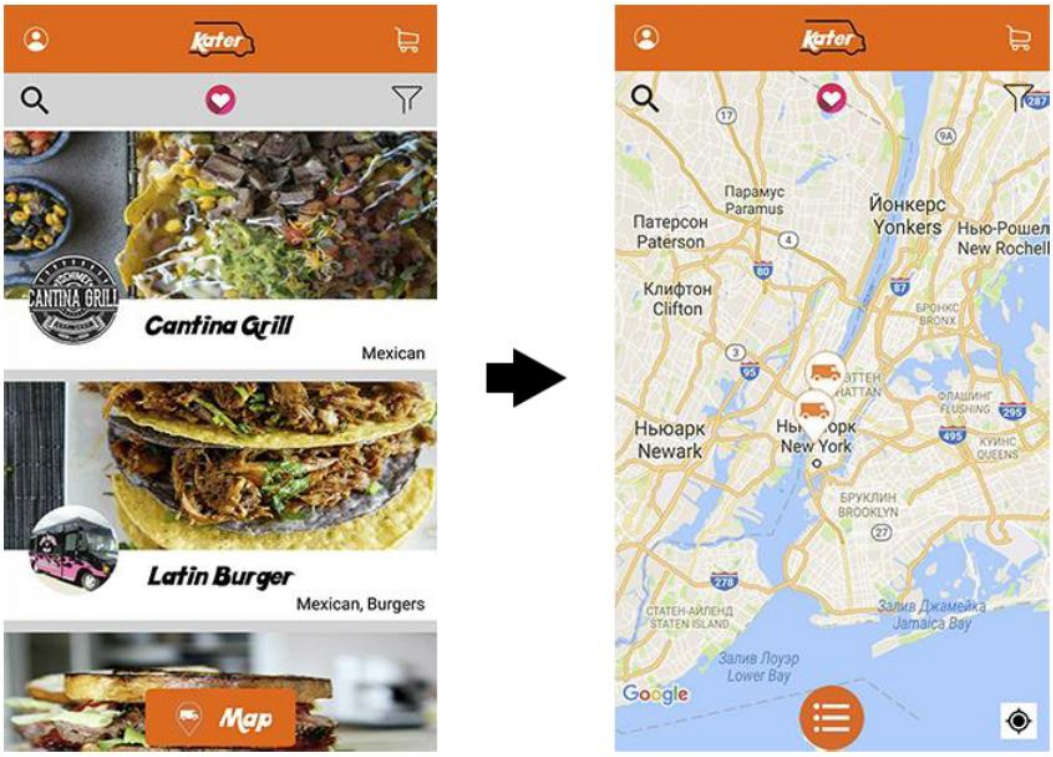

Рисунок 1.6 – Послідовність дій для пошуку точок продажів

1.2 Огляд хмарної бази даних Firebase Realtime Database

База даних (БД) – об'єктивна форма представлення сукупності даних, систематизованих так, щоб ці дані були знайдені і оброблені за допомогою ЕОМ.

Для управління, зміни та адміністрування бази даних використовується система управління базами даних (СУБД). Кожна СУБД підтримує різні моделі і структури БД [1, 2].

1.2.1 Реляційна модель БД

Однією з найпопулярніших моделей бази даних є реляційна модель. Реляційна модель надає способи чіткого структурування даних. У ній використовуються таблиці з колонками і рядами. У кожного ряду таблиці (запису) є унікальний номер (первинний ключ), а колони служать полями даних (див. рис. 1.7).

Рисунок 1.7 – Приклад таблиці реляційної БД

Реляційні моделі БД використовують схеми – строго описані структури даних. Схема бази даних включає в себе опис змісту, структури і обмеження цілісності [3].

1.2.2 Нереляційні бази даних

У NoSQL базах структура даних не регламентована – в окремому рядку або документі можна додати довільне поле без попереднього декларування зміни структури всієї таблиці. Таким чином, якщо з'являється необхідність змінити модель даних, то єдина достатня дія – відобразити зміну в коді програми [6].

NoSQL БД не ґрунтуються на одній моделі, а кожна з баз даних, в залежності від цілей, використовує різні моделі.

Залежно від моделі даних можна виділити чотири типи сховищ:

1) сховище «ключ–значення»;
2) сховище сімейств колонок;
3) документо–орієнтовані СУБД;
4) бази даних на основі графів.

Більш докладно розглянемо документо–орієнтовану модель. Дана модель ґрунтується на текстовому форматі обміну даними JSON. Бази даних такого формату складаються з ключів і даних. Ключ може містити в собі інші ключі, які в свою чергу будуть мати дані. Таким чином утворюються підвузли і вузли (див. рис. 1.8). В даному випадку вузлом є «customers», Підвузли є «customers_one», ключі «firstName», «birthday», «location» і їх дані становлять поля даних (див. рис. 1.9).

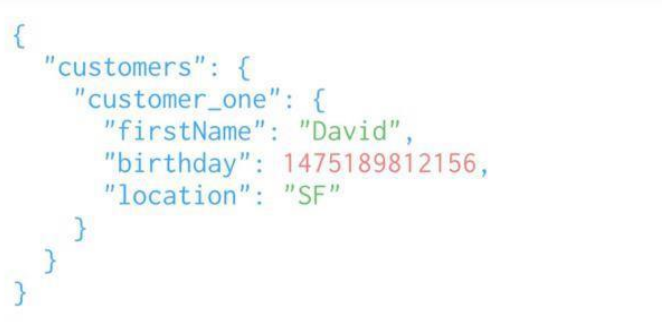

Рисунок 1.8 – Приклад вузла нереляційної бази даних

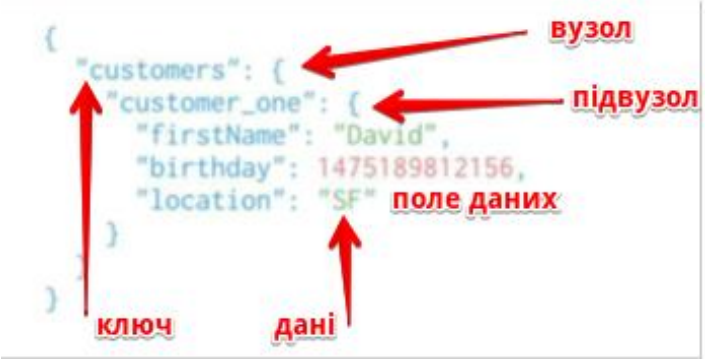

Рисунок 1.9 – Розбір вузла на підвузли і поля даних

Перевагами нереляційних БД перед реляційними моделями БД є:

1) велика швидкість зчитування в розподіленому середовищі;
2) можливість зберігати фізичні об'єкти в тому вигляді, в якому з ними працює додаток;
3) можливість розробки бази даних без завдання схеми;
4) більш прості способи горизонтального масштабування (створення кластерів з декількох машин);
5) рідна підтримка атомарності на рівні записів.

Для зберігання всіх даних користувача, буде використана база даних Firebase Realtime Database. Це хмарна база даних, дані зберігаються як JSON і синхронізуються в реальному часі з кожним підключеним клієнтом. При побудові крос–платформових додатків з цими SDK для iOS, Android і JavaScript, всі клієнти використовують один екземпляр бази даних Realtime і

автоматично отримують оновлення [3, 5].

База даних Firebase Realtime дозволяє створювати програми з безпечним доступом до бази даних безпосередньо з коду на стороні клієнта. Дані зберігаються локально, і навіть в автономному режимі події в реальному часі продовжують оновлюватись, надаючи кінцевому користувачеві найактуальніші дані. Коли пристрій відновлює з'єднання, база даних у реальному часі синхронізує зміни місцевих даних з віддаленими оновленнями, які відбувалися під час перебування клієнта в автономному режимі, автоматично усуваючи будь–які розбіжності.

База даних реального часу надає гнучку мову правил на основі виразів, яка називається Firebase Realtime Database Security Rules, щоб визначити, як дані повинні бути структуровані, коли і ким дані можуть бути прочитані або записані. При інтеграції з аутентифікацією Firebase розробники можуть визначити, хто має доступ до тих даних і як вони можуть його отримати.

База даних Realtime – це база даних NoSQL, має різні оптимізації і функціональність у порівнянні з реляційною базою даних. API баз даних реального часу розроблений, щоб дозволити операції, які можна виконувати швидко. Це дає змогу створити великий досвід роботи в реальному часі, який може обслуговувати мільйони користувачів без шкоди для реагування. У зв'язку з цим важливо подумати про те, як користувачам необхідно отримати доступ до даних, а потім відповідним чином структурувати їх.

Серед плюсів організації хмарної бази даних можна відзначити наступні:

- доступність. Послуга надається через Інтернет і є доступною цілодобово за умови функціонування сервера–провайдера;
- можливість розширення, масштабованість. Обробка і зберігання даних здійснюється на серверах віддалених дата–центрів і розподілена по фізичним машинам, надаючи рівно такі потужності, які необхідні користувачу;
- легкість у використанні. Користувачеві не потрібно піклуватися про

інфраструктуру і супровід процесів обробки і зберігання даних.

Таким чином, хоча Firebase Realtime Databse і не базується на технологіях SQL, вона представляє нові можливості, не вимагаючи при цьому особливих навичок або знань, відмінних від застосовних з технологіями традиційного SQL Server.

Для зберігання самих файлів є Firebase Strorage, яка і буде використовуватись. Cloud Storage побудована для розробників програм, які повинні зберігати та обслуговувати створений користувачем вміст, наприклад фотографії або відео.

Cloud Storage для Firebase – це потужна, проста і економічно ефективна служба зберігання об'єктів, побудована компанією Google. Firebase SDK для Cloud Storage додає безпеку Google для завантаження файлів і завантажень програм Firebase, незалежно від якості мережі. Можна використовувати SDK для зберігання зображень, аудіо, відео чи іншого вмісту, створеного користувачем. На сервері можна скористатися Google Cloud Storage, щоб отримати доступ до тих самих файлів [2].

Розробники використовують Firebase SDK для Cloud Storage для завантаження файлів безпосередньо з клієнтського пристрою. Cloud Storage зберігає файли в Google Cloud Storage, роблячи їх доступними як через Firebase, так і з Google Cloud. Це дає змогу завантажувати та завантажувати файли з мобільних клієнтів через Firebase SDK, а також здійснювати обробку на стороні сервера, наприклад, фільтрування зображень або перекодування відео за допомогою Google Cloud Platform.

Firebase SDK для Cloud Storage інтегрується з Firebase Authentication для ідентифікації користувачів, і надається декларативна мова безпеки, яка дозволяє встановлювати елементи керування доступом для окремих файлів або груп файлів, що забезпечує можливість створення файлів як загальним доступом так і приватним.

1.3 Вибір мови програмування

Kotlin – це відносно молода мова програмування від компанії JetBrains, що спеціалізується на створенні інструментів для розробки. Автори ставили за мету створити мову більш лаконічну і безпечну, ніж Java, і більш просту, ніж Scala. У 2017 Google оголосила про офіційну підтримку мови Kotlin для розробки під Android.

Java – це мова програмування та програмна платформа, яка широко використовується під час розробки програмного забезпечення для самих різних комп'ютерних платформ: від вбудованих пристроїв і мобільних телефонів, до корпоративних серверів і суперкомп'ютерів [3].

На даний момент, Android Studio надає розширену підтримку Kotlin і інструменти адаптації. Розробники можуть працювати одночасно на Kotlin і на Java. Незалежно від усіх відмінностей між двома мовами програмування, вони цілком сумісні. Як Java, так і Kotlin компілюються в байт–код. Це означає, що можна викликати Java–код з Kotlin і навпаки. Ця гнучкість має дві переваги: по–перше, це полегшує початок роботи з Kotlin–ом, поступово впроваджуючи код Kotlin в проект Java, а по–друге, обидві мови можна використовувати одночасно в будь–якому проекті розробки додатків Android.

Та все ж Kotlin стає більш популярним, тому що має такі переваги:

- покращена продуктивність. Послідовний і інтуїтивний синтаксис Kotlin забезпечує підвищення продуктивності команд розробників. Для написання програми потрібно менше часу і менше рядків коду.
- легкість підтримки. Android–розробникам легко підтримувати код на Kotlin, оскільки ця мова підтримується в багатьох IDE, включаючи Android studio, і в декількох інших SDK. Крім того, розробники можуть працювати з будь–яким звичним набором інструментів.
- надійність. Остання версія Kotlin має зворотну сумісність з усіма попередніми версіями. Це позбавляє Android–розробників від величезної головного болю – роботи з різними версіями.

- легкість вивчення
- підтримка Android Studio. Android Studio надає розширену підтримку Kotlin і інструменти адаптації. Розробники можуть працювати одночасно на Kotlin і на Java
- порівняння класу Java з еквівалентним класом Kotlin демонструє лаконічність коду Kotlin. Для однієї ж і тієї операції, що виконується в класі Java, клас Kotlin вимагає менше коду.
- розумне приведення типу. Перш ніж об'єкт може бути приведений в Java, обов'язково потрібно перевірити тип. Це також вірно в сценаріях, де очевидно потрібно приводити об'єкт. На відміну від Java, Kotlin має функцію розумного приведення, яка автоматично обробляє такі надлишкові приведення.

До того ж, Koltin вирішує цілий ряд проблем, від яких страждає Java, а саме:

- посилання на null контролюються системою типів;
- немає сирих (raw) типів;
- масиви в Kotlin інваріантні;
- Kotlin має правильні функціональні типи і підтримує їх використання замість SAM–типів з Java;
- варіативність на місці використання без підстановки;
- у Kotlin немає перевіряємих винятків.

Саме через низку переваг Kotlin перед Java, ця мова буде використовуватись у проекті.

2 ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Концептуальна модель бази даних

В результаті аналізу поставленого завдання і узагальнення вимог кінцевих користувачів складається концептуальна модель, яка представляє інформаційні об'єкти і їх зв'язку без вказівки способів фізичного зберігання.

Концептуальна модель – це діаграма, відображення предметної області, для якої розробляється база даних, з прийнятими позначеннями елементів. Так, всі об'єкти, що позначають речі, позначаються у вигляді прямокутника. Атрибути, що характеризують об'єкт – у вигляді овалу, а зв'язки між об'єктами – ромбами. Потужність зв'язку позначаються підписами біля стрілок.

При проектуванні концептуальної моделі, всі зусилля розробника повинні бути спрямовані в основному на структурування даних і виявлення взаємозв'язків між ними без розгляду особливостей реалізації і питань ефективності обробки. Проектування концептуальної моделі засновано на аналізі вирішуємих завдань по обробці даних. Концептуальна модель включає описи об'єктів і їх взаємозв'язків, що представляють інтерес в аналізованої предметної області і виявляються в результаті аналізу даних.

Концептуальна модель є, по суті, моделлю предметної області. Процес створення концептуальної моделі більшою мірою відноситься до проектування всієї інформаційної системи, однією з частин якої є база даних. На даному етапі відбувається аналіз завдань, що вирішуються в конкретній предметній області, описуються об'єкти предметної області та взаємозв'язку цих об'єктів. Концептуальна модель також може містити опис процесів, які відбуваються з об'єктами предметної області, що дозволяє найбільш повно врахувати всі нюанси функціонування розроблюваної інформаційної системи.

При проектуванні концептуальної моделі не враховуються особливості реалізації певних частин інформаційної системи і не розглядаються питання

підвищення ефективності обробки інформації. Кожен користувач може зареєструвати один або декілька акаунтів. Кожен профіль користувача має декілька полів, що містять основну інформацію. Окремою таблицями є магазин та товари. Кожний магазин має адресу та перелік товарів, що або є у наявності, або немає. Користувач може переглядати товари та додавати їх собі у кошик.

На першому етапі побудови концептуальної моделі маємо наступну схему:

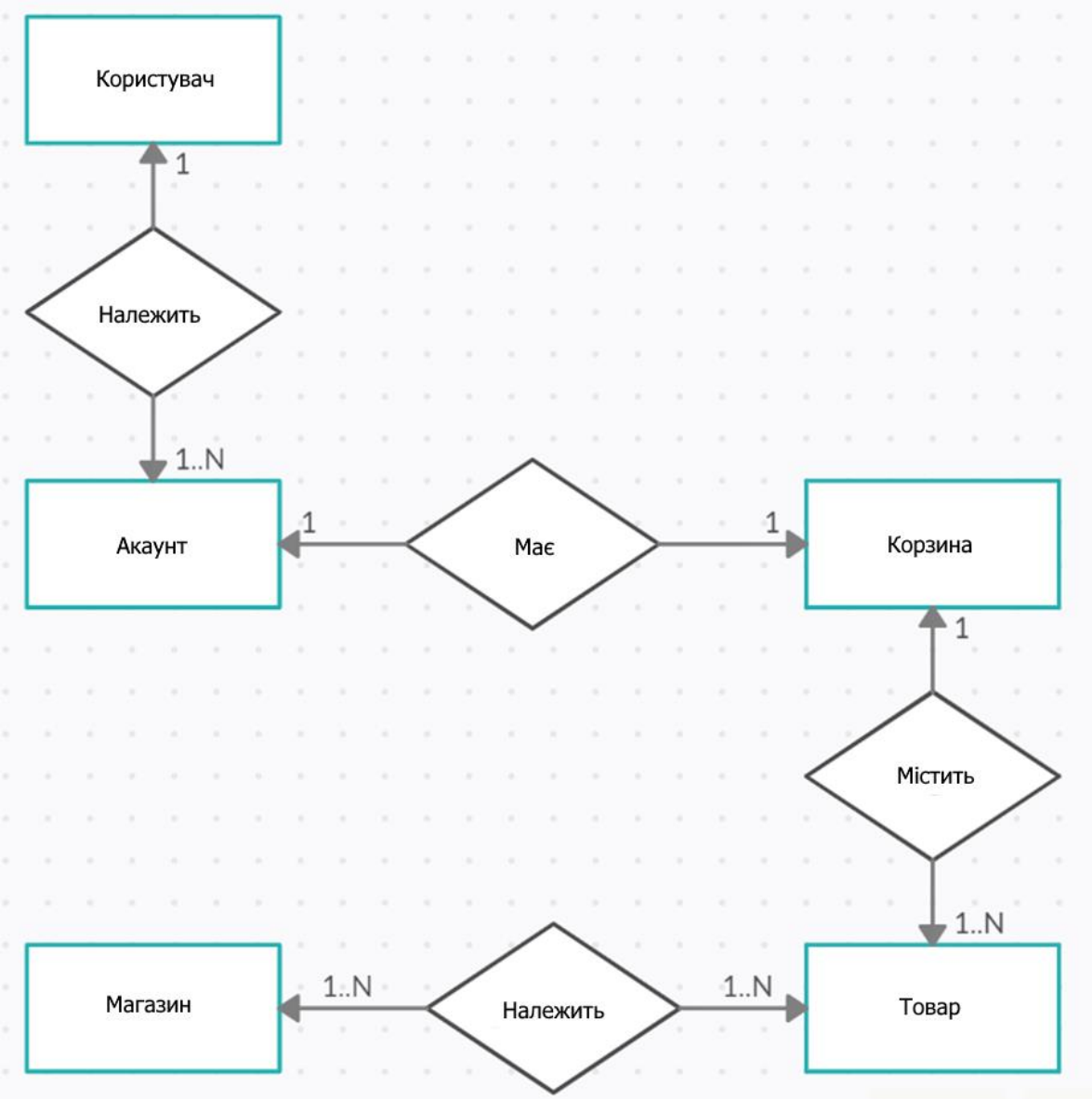

Рисунок 2.1 – Перший етап проектування концептуальної моделі бази даних

Між товарами та магазинами отримуємо зв’язок «багато до багатьох», що є недопустимим у реляційній моделі бази даних. Тому кожен такий зв’язок потрібно замінити на два «один до багатьох». Розширимо модель, додавши проміжні об’єкти «Перелік товарів в магазині».

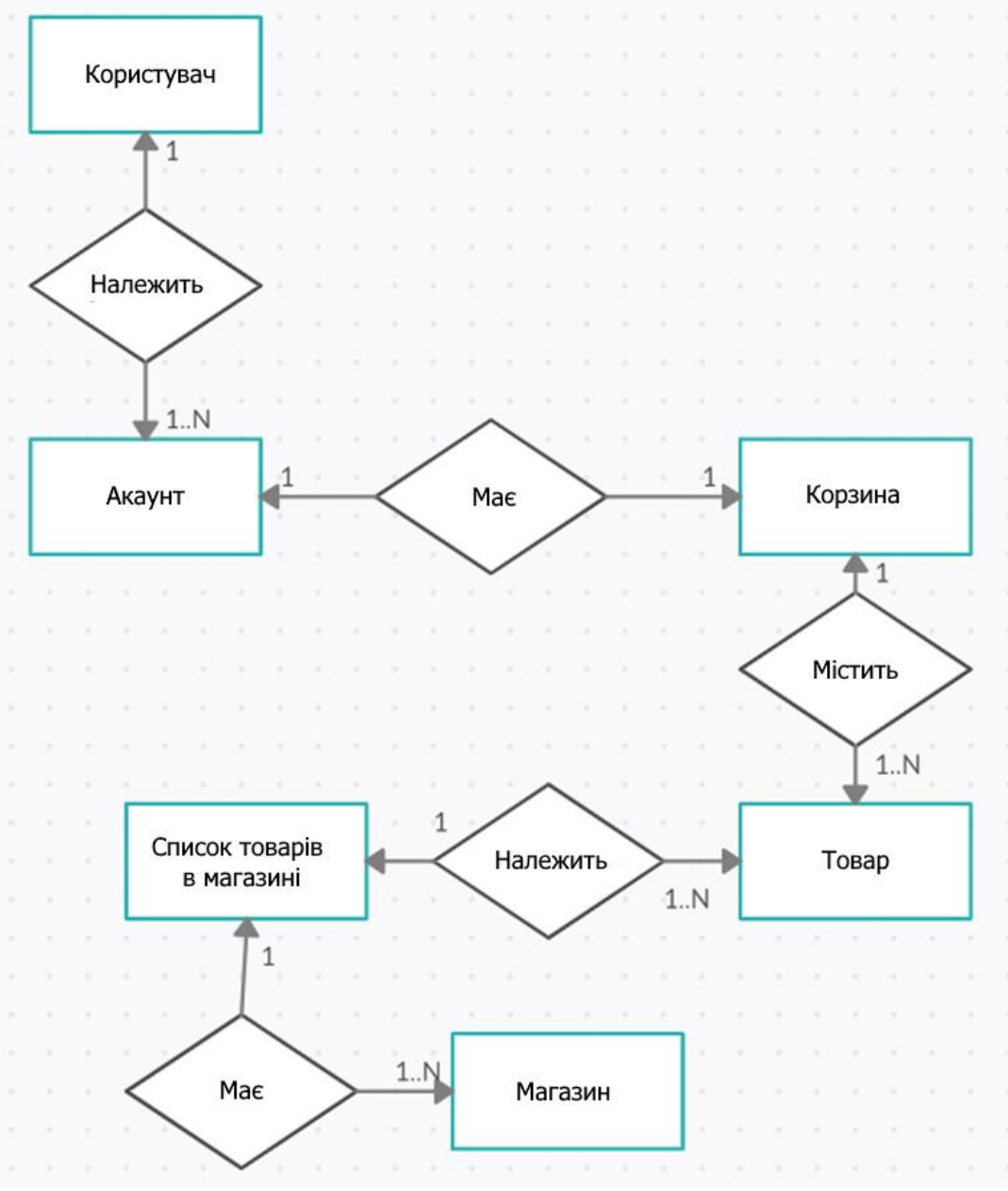

Рисунок 2.2 – Другий етап проектування концептуальної моделі бази даних

Наступним етапом буде додати атрибути для кожного об’єкту. Інформація про акаунт містить адресу електронної пошти, повне ім’я

користувача, номер телефону, фотографію, пароль від облікового запису, а також ID користувача – його унікальний номер у базі даних. Кошик має ID користувача. Товар має поля ID товару, назва, фотографія, ціна, та опис товару. Магазин складається з ID та адреси.

Концептуальна модель бази даних готова виглядає наступним чином:

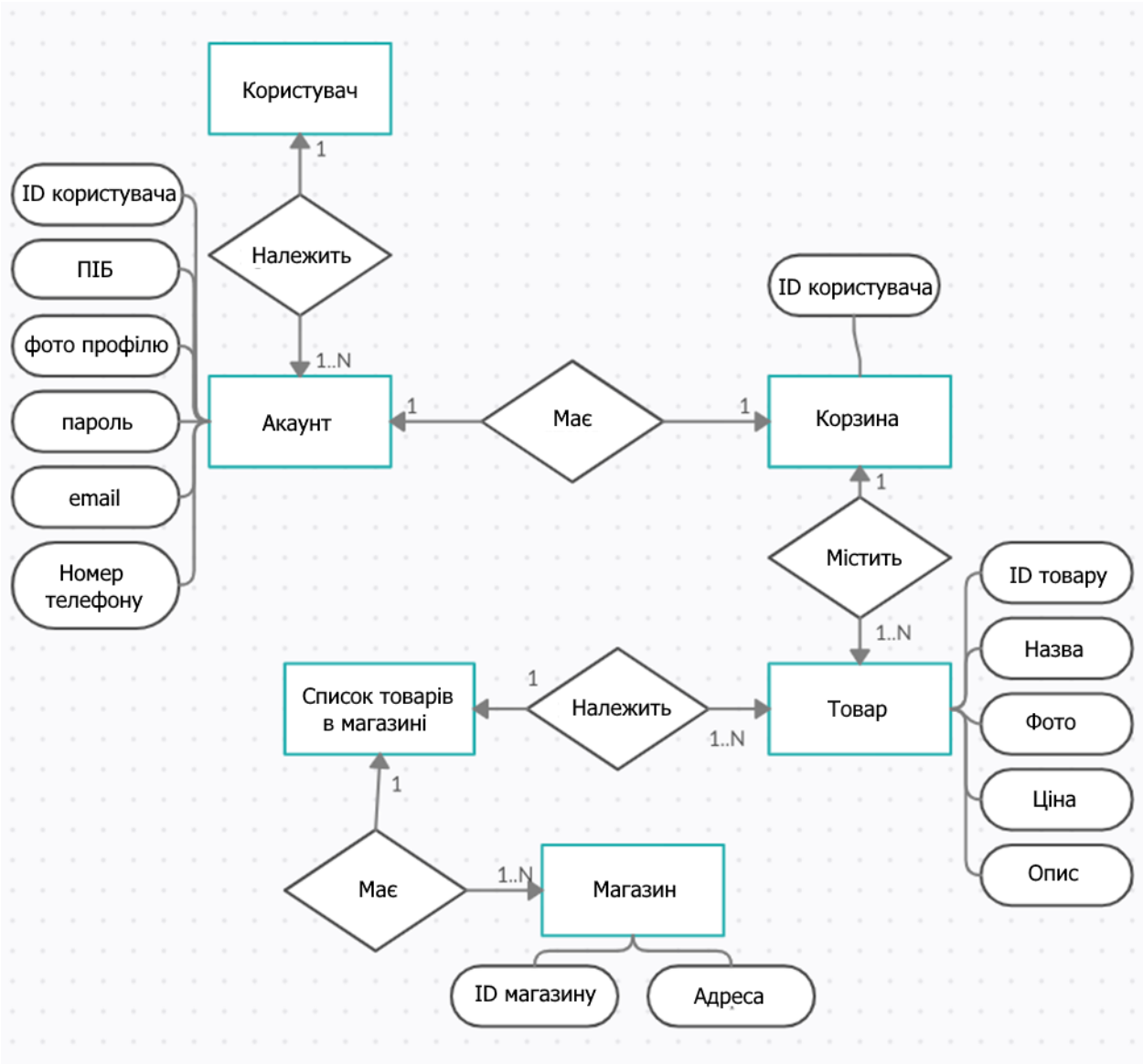

Рисунок 2.3 – Третій етап проектування концептуальної моделі бази даних

2.2 Логічний рівень проектування бази даних

Логічна модель БД являє собою результат перетворення концептуальної моделі, в якому основними об'єктами стають інформаційні об'єкти. В якості останніх виступають сутності – об'єкти або події, інформацію про яких необхідно зберігати в базі даних. Сутності характеризуються набором деяких властивостей, званих атрибутами. Логічна модель відображає логічні зв'язки між сутностями незалежно від того, яким чином будуть зберігатися дані. Логічна модель БД є універсальною, оскільки ніяк не пов'язана з конкретною реалізацією СУБД. Назви сутностей і атрибутів в логічної моделі можуть збігатися з назвами, що використовуються в реальному житті.

Основуючись на вимогах, що були визначені у першому розділі, було розроблено моделі поведінки проекту у нотації об’єктно–орієнтовної мови UML – це графічна мова для візуалізації, опису параметрів, конструювання та документування різних систем, визначено основний функціонал додатку та принципи його роботи.

Побудовано діаграму варіантів використання (Use Case diagram) (див. рис. 2.4) для моделювання функціональних вимог до системи (у вигляді сценаріїв взаємодії користувачів з системою). Варіанти використання це – опис послідовності дій, які може здійснювати система у відповідь на зовнішні дії користувачів або інших програмних систем. Варіанти використання відображають функціональність системи, описують функціональне призначення системи або те, що система повинна робити. Мета розробки діаграм наступна:

- визначити загальні межі і предметну область;
- сформулювати загальні вимоги до функціональної поведінки проектованої системи.

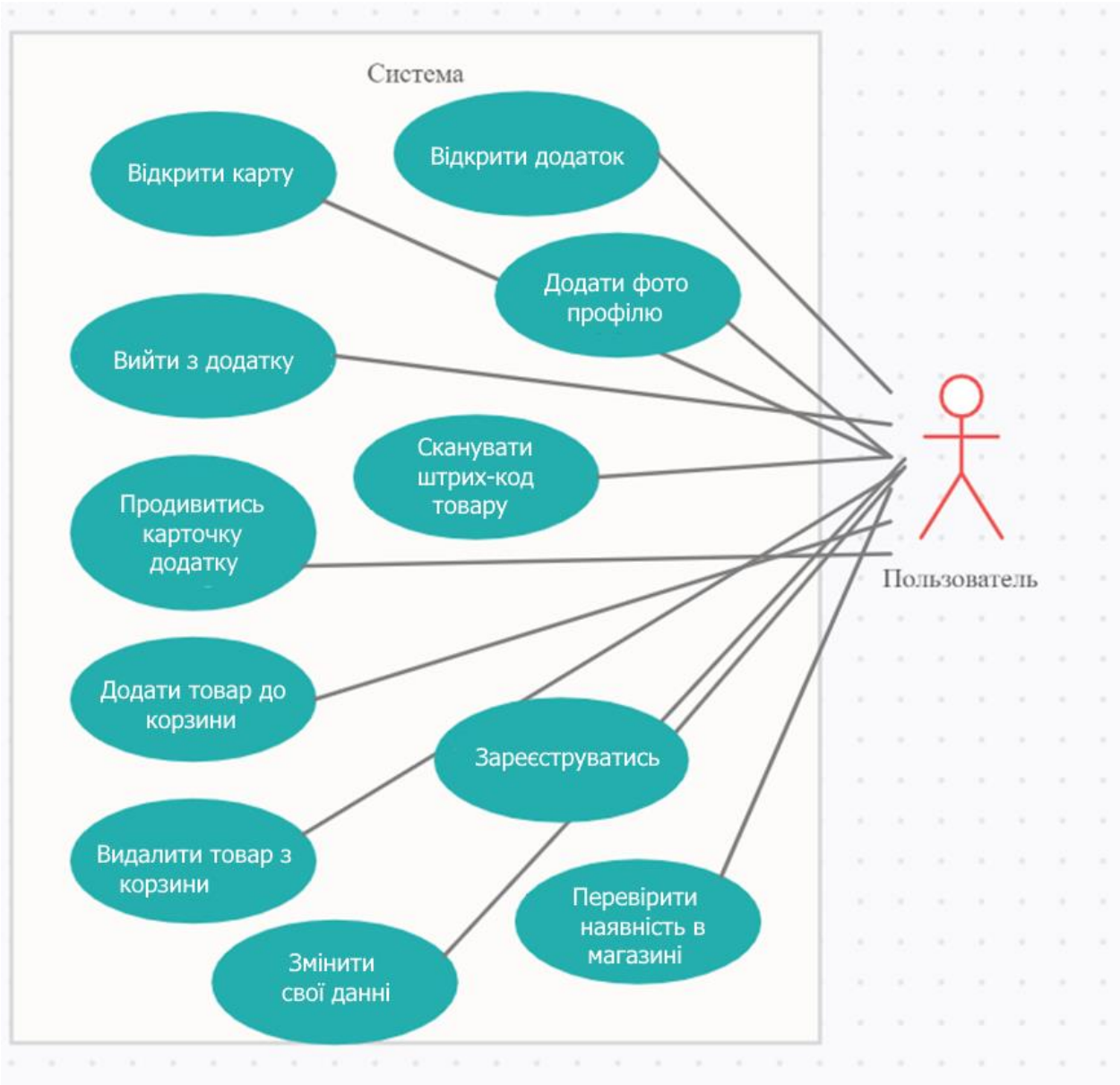

Рисунок 2.4 – Діаграма варіантові використання

Діаграми взаємодії (Interaction diagrams) – використовуються для моделювання процесу обміну повідомленнями між об'єктами, вони включають в себе наступні діаграми.

Діаграму послідовності (Sequence diagrams) (див. рис. 2.5) – відображає тимчасову послідовність подій, що відбуваються в рамках варіанту використання. Для її побудови кожному з об’єктів сценарію ставиться у відповідність його лінія життя, яка відображає перебіг подій між його створенням та руйнуванням. На діаграмі вона позначається вертикальною

лінією, на верхівці якої в прямокутнику зображується назва об’єкта. На діаграмі зазначені всі об’єкти, які беруть участь у взаємодії.

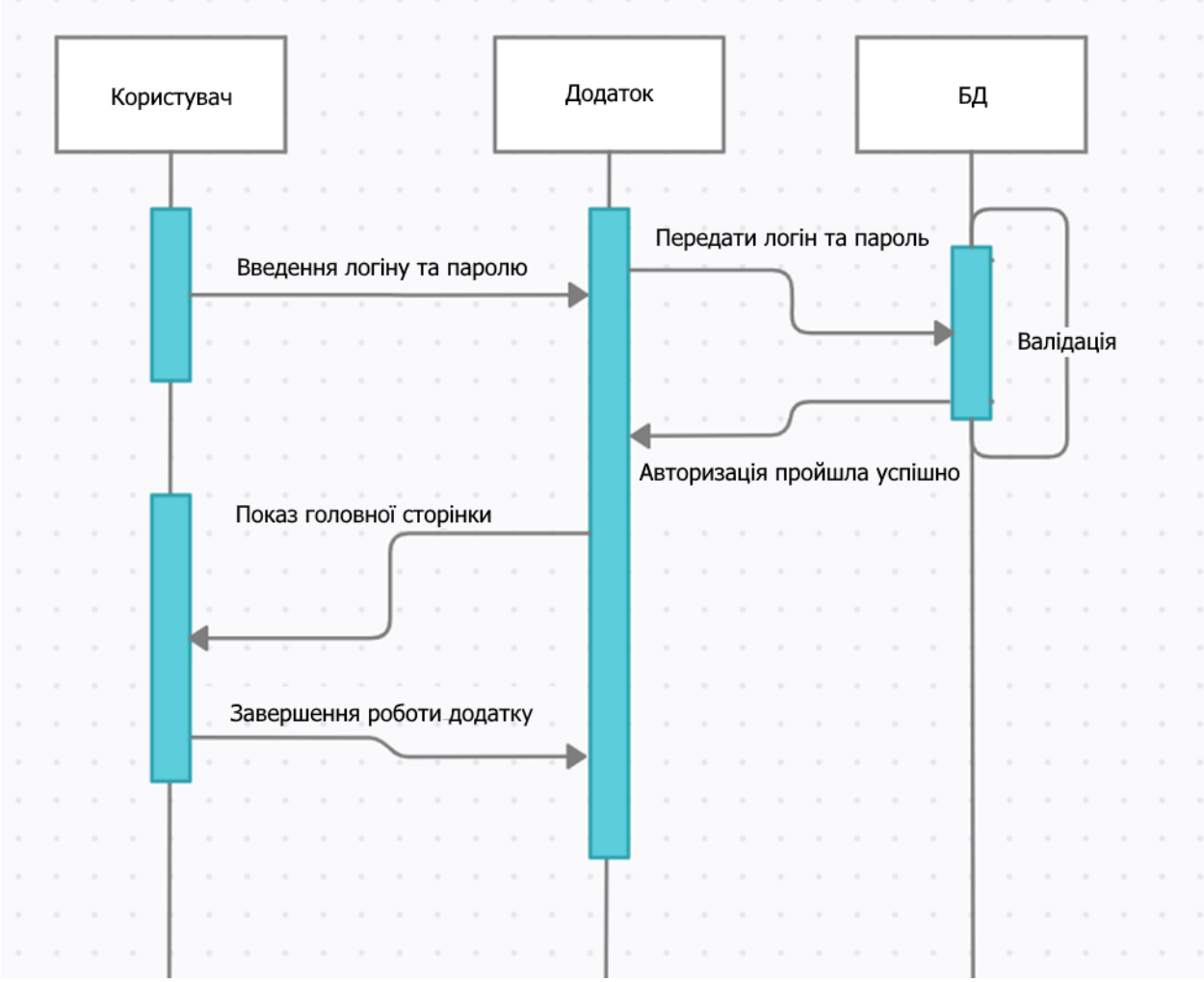

Рисунок 2.5 – Діаграма послідовності

Діаграму кооперації (Collaboration diagrams) (див. рис. 2.6) – концентрує увагу на зв'язках між об'єктами. Діаграми кооперації являють собою сукупність об’єктів, поведінка яких значуща для досягнення складових мети системи, та взаємовідношення тих ролей, які об’єкти відіграють у співробітництві. На даному вигляді діаграм моделюється статична взаємодія об’єктів, при цьому фактор часу не враховується і не відображається на діаграмі співробітництва.

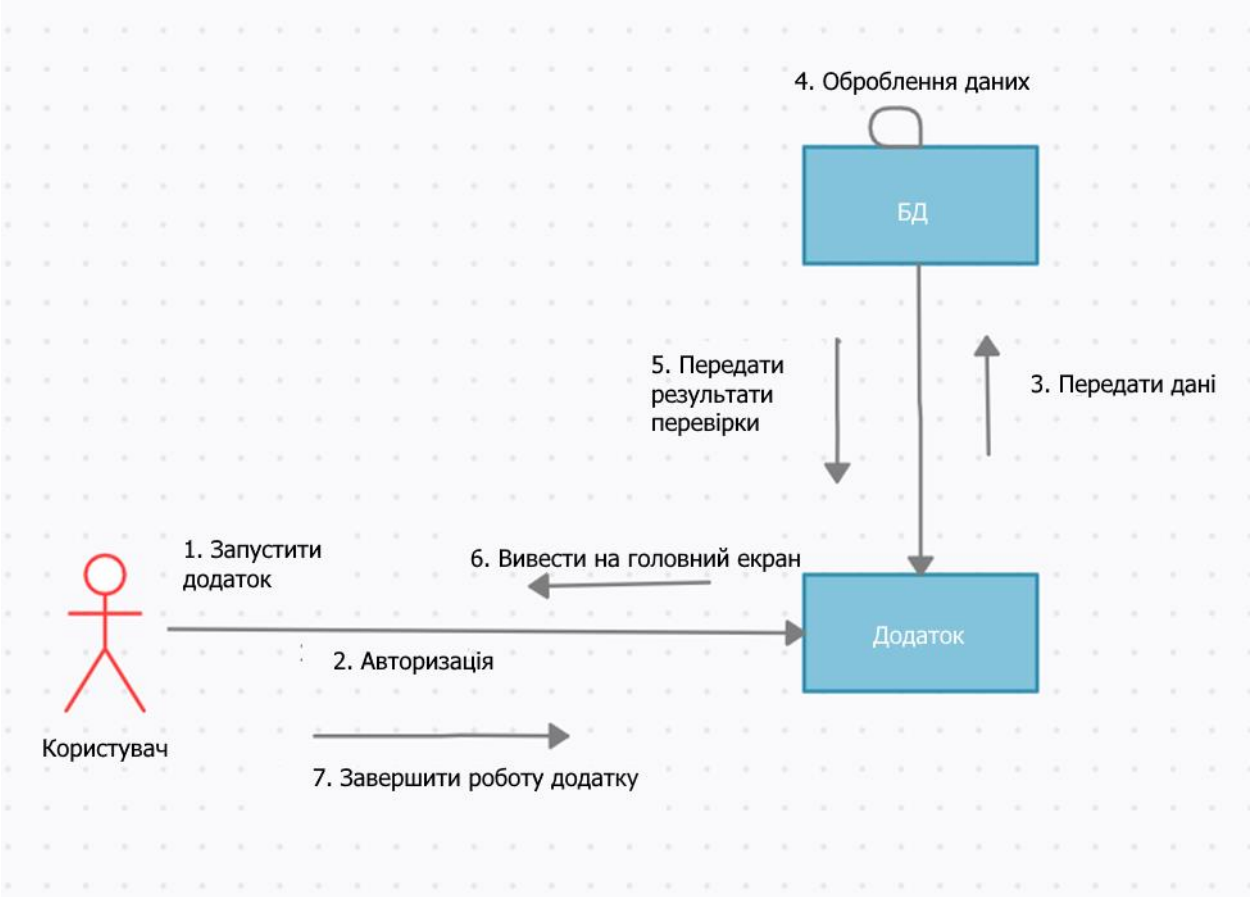

Рисунок 2.6 – Діаграма кооперації

Діаграму діяльності (Activity diagrams) (див. рис. 2.7) – для моделювання поведінки системи в рамках різних варіантів використання, або потоків управління. Окрема діяльність (робота) зображується на діаграмі прямокутником із закругленими кутами. Потоки керування між роботами показуються стрілками. Якщо мова йде про прийняття рішення, то з відповідного прямокутника виходять дві стрілки, на кожній може позначатися текст умови, якій вона відповідає. Діаграма діяльності нагадує відомі блок–схеми алгоритмів та програм, зокрема передбачено відображення можливості виконувати паралельно кілька діяльностей.

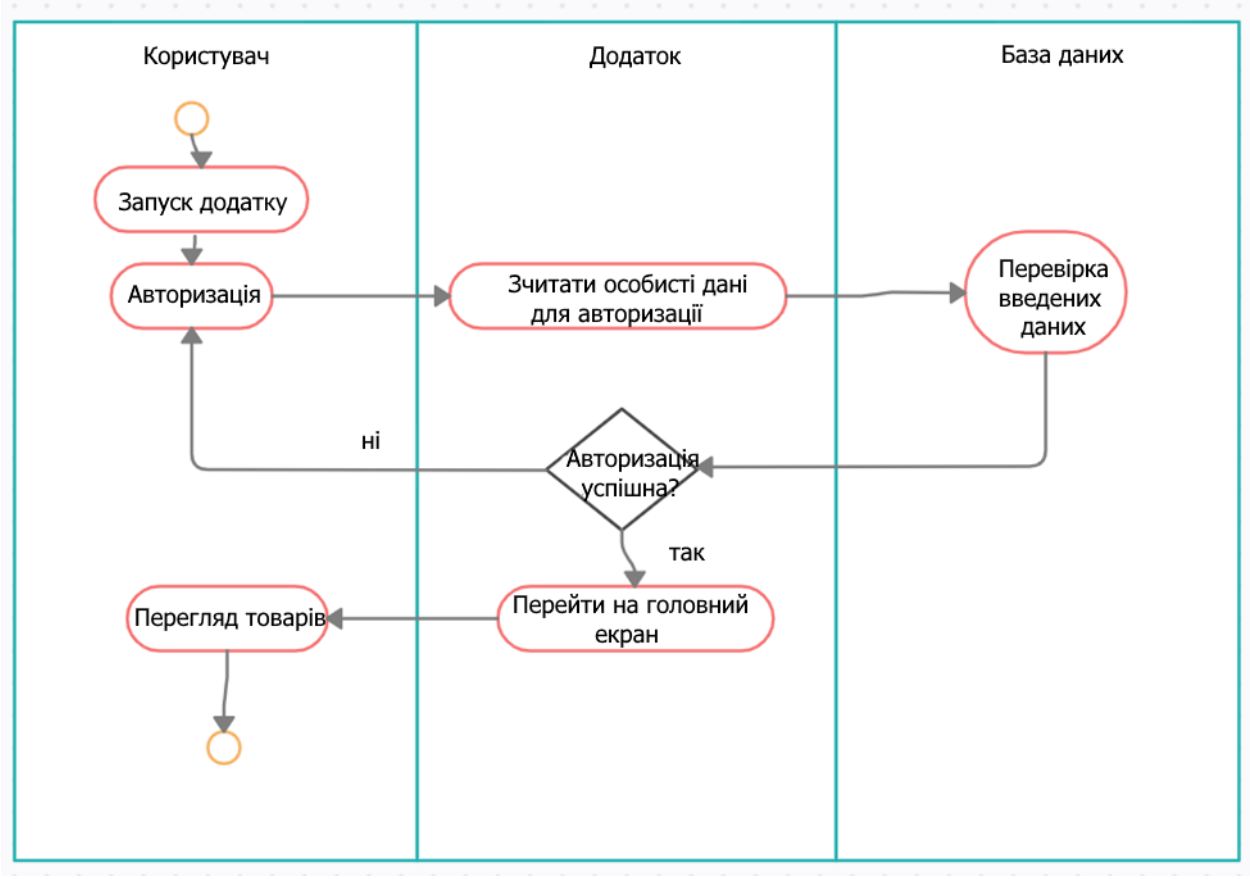

Рисунок 2.7– Діаграма діяльності

Діаграму станів (Statechart diagrams) (див. рис. 2.8) – для моделювання поведінки об'єктів системи при переході з одного стану в інший. Діаграми станів застосовуються для того, щоб пояснити, яким чином працюють складні об'єкти. Діаграма станів показує, як об'єкт переходить з одного стану в інший. Очевидно, що діаграми станів служать для моделювання динамічних аспектів системи. Така діаграма корисна при моделюванні життєвого циклу об'єкта. Від інших діаграм вона відрізняється тим, що описує процес зміни станів тільки одного примірника певного класу – одного об'єкта, причому об'єкта реактивного, тобто об'єкта, поведінка якого характеризується його реакцією на зовнішні події.

Рисунок 2.8– Діаграма діяльності

Діаграма компонентів (Component diagrams) (див. рис. 2.9) використовується для моделювання ієрархії компонентів / підсистем системи. Вона дозволяє визначити архітектуру системи, що розробляється, встановивши залежності між програмними компонентами, в ролі яких може виступати початковий і виконуваний код. Основними графічними елементами діаграми компонентів є компоненти, інтерфейси і залежності між ними. Діаграма компонентів розробляється для наступних цілей:

– візуалізувати загальну структури початкового коду програмної системи;

– специфікації виконуваного варіанту програмної системи;

– забезпечення багатократного використання окремих фрагментів програмного коду;

– представлення концептуальної і фізичної схем баз даних.

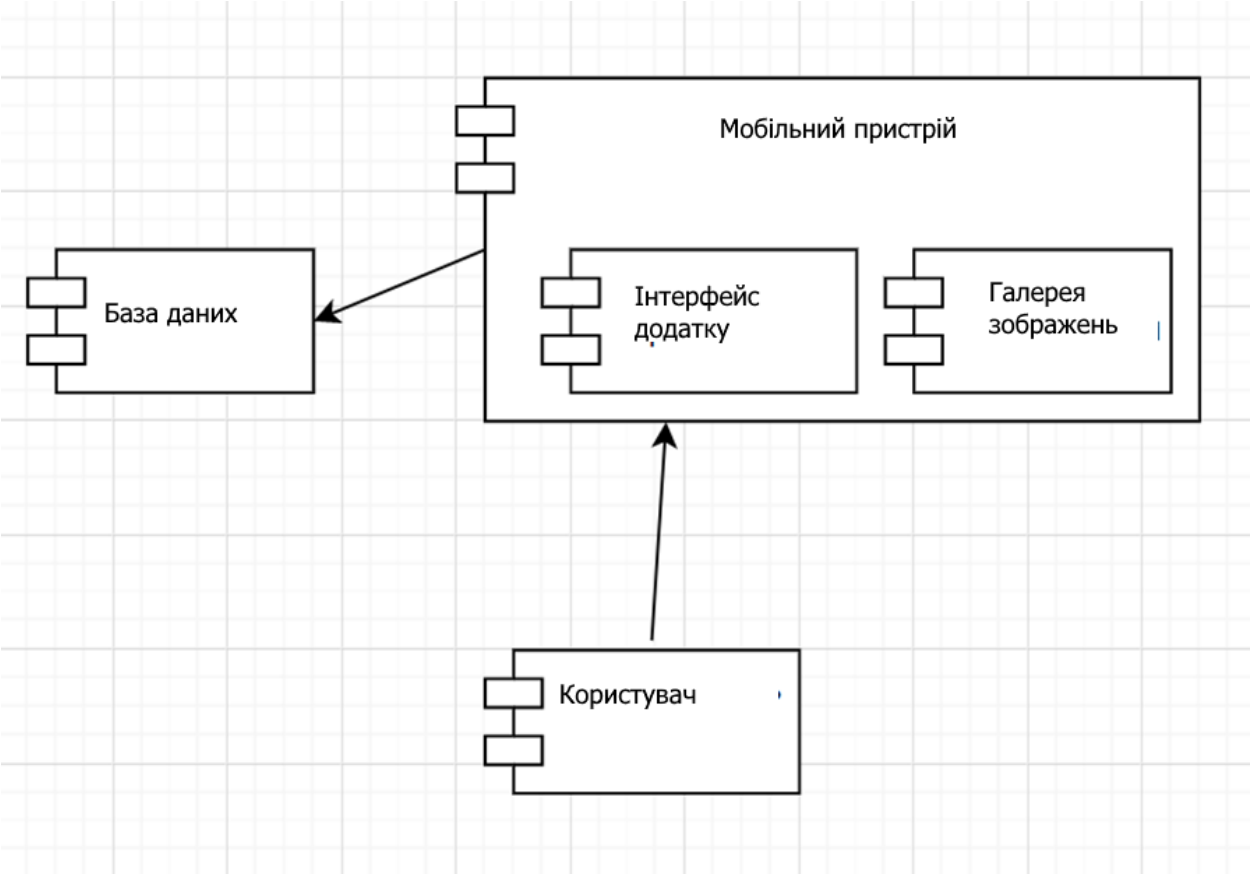

Рисунок 2.9 – Діаграма компонентів

Діаграма розгортання або розміщення (Deployment diagrams) (див. рис. 2.10) використовується для моделювання фізичної архітектури системи. Діаграма розгортання містить графічні зображення процесорів, пристроїв, процесів і зв'язків між ними. На відміну від діаграм логічного уявлення, діаграма розгортання є єдиною для системи в цілому, оскільки повинна цілком відображати особливості її реалізації. Розробка діаграми розгортання, як правило, є останнім етапом специфікації моделі програмної системи.

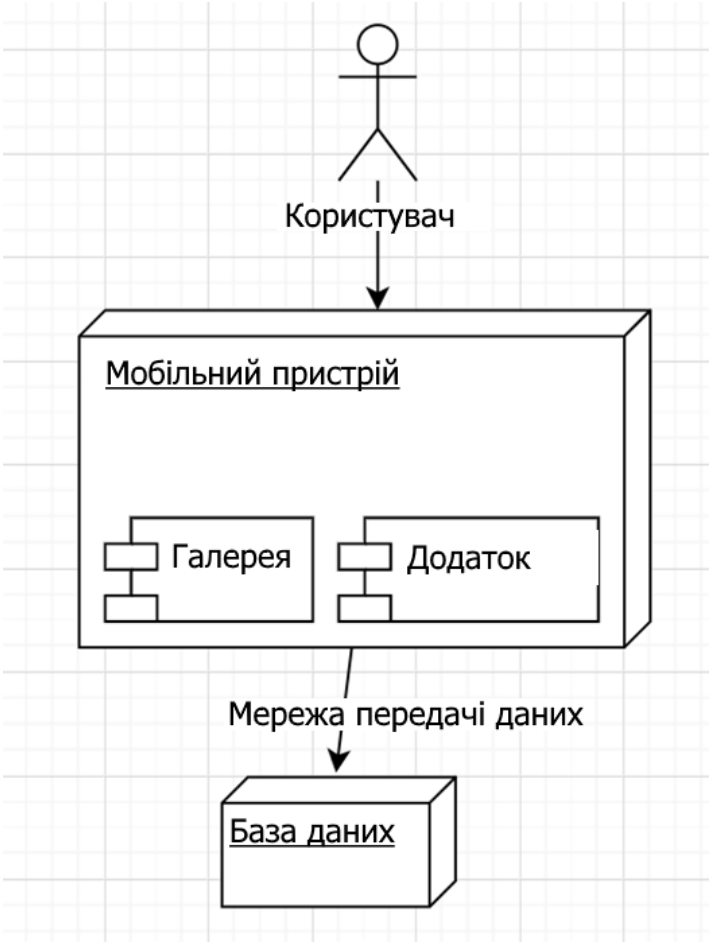

Рисунок 2.10 – Діаграма розгортання

3 РОЗРОБКА ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Фізичний рівень проектування бази даних

Фізична модель є відображенням логічної моделі стосовно конкретної СУБД. Однієї і тієї ж логічної моделі бази даних може відповідати кілька різних фізичних моделей, що відображають особливості реалізації конкретних СУБД. У фізичної моделі важливо описати всю інформацію про фізичні об'єкти бази даних – таблицях, колонках, індексах, процедурах і т.д. Сучасні засоби проектування фізичної моделі БД дозволяють на основі створеної моделі сформувати необхідні приписи (команди, запити) для обраної системи управління базою даних. СУБД на підставі отриманих приписів формує фізичну структуру бази даних, призначену для зберігання реальної інформації.

Для реалізації бази даних проекту будемо використовувати Firebase Realtime Database. Взявши за основу отриману модель даних, було створено необхідні таблиці (див. рис 3.1).

Рисунок 3.1 – Таблиці у Firebase Realtime Database

Таблиця «user» має наступні поля :

Рисунок 3.2 – Таблиця «user»

Таблиця «shop» складається з наступних підвузлів:

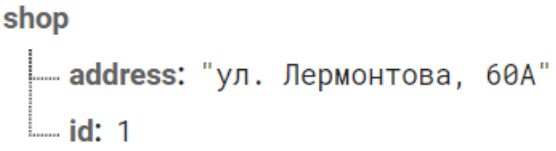

Рисунок 3.3 – Таблиця «user»

Таблиця «product» представляє собою наступне:

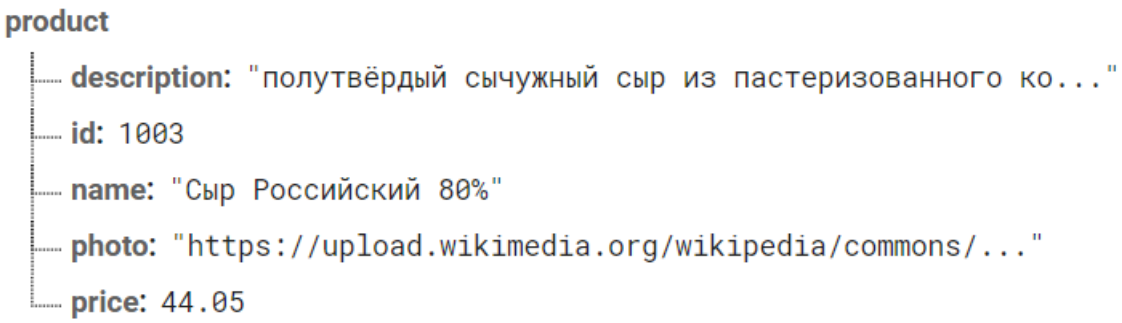

Рисунок 3.4 – Таблиця «product»

3.2 Реєстрація та авторизація в системі

Для авторизації в системі використовується Firebase Authentication. Визначені наступні способи входу:

- за допомогою електронної пошти;
- аккаунту соціальної мережі Facebook.

Система автоматично призначає користувачам унікальний ідентифікатор, який потім використовується в базі даних як ключ.

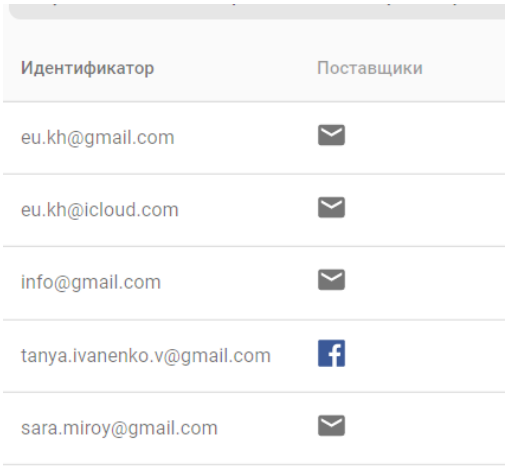

Рисунок 3.4 – Перелік акаунтів Firebase Authentication

Після запуску додатку, користувач бачить перший екран з логотипом додатку на 2 секунди (див. рис. 3.5) .

Рисунок 3.5 – Вікно запуску додатку

Далі, користувач потрапляє на екран авторизації. Тут він може зробити вхід якщо він вже має зареєстрований акаунт за допомогою логіну та паролю або соціальної мережі Facebook (див. рис. 3.6).

Рисунок 3.6 – Вікно авторизації

Код програми для авторизації за допомогою електронної пошти та паролю:

```
val email = email_input.text.toString()
val password = password_input.text.toString()
if(validate(email, password)) {
        mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener {
                if(it.isSuccessful) {
                        startActivity(Intent(this, HomeActivity))
                        finish() }}}
        else {
                showToast("Please enter email and password")
}
```

Для авторизації через Facebook акаунт використовується бібліотека «com.google.firebase.auth.FacebookAuthProvider» (див. рис 3.7).

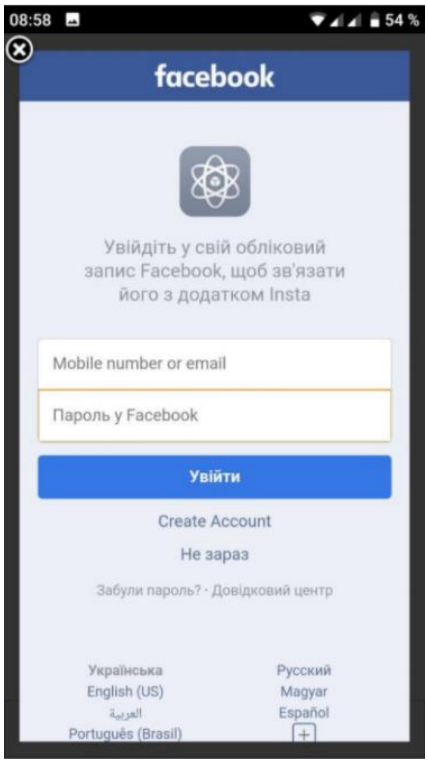

Рисунок 3.7 – Вхід через акаунт у Facebook

Щоб зареєструвати новий акаунт, на потрібно натиснути «Нет аккаунта? Зарегистрироваться», що розташовується внизу вікна, ввести свою електронну пошту, ім’я та придумати пароль (див. рис. 3.8).

Рисунок 3.8 – Реєстрація нового акаунту

Функція для створення нового користувача:

```
val email = mEmail
if (email != null) {
        mAuth.createUserWithEmailAndPassword(email, password) {
                val user = mkUser(fullname, email)
                mDatabase.createUser(it.user.uid, user) {
                        startHomeActivity()
                }
        }
}
```

3.3 Основні можливості системи

Якщо користувач щойно зареєструвався в системі, він може зайти у налаштування профілю, де можна, наприклад, змінити ім’я профіля та заповнити наступні дані про себе:

– власне ім’я;

– номер телефону;

– змінити адресу електронної пошти (і потім використовувати її замість попередньої при авторизації).

– додати або змінити фото профіля.

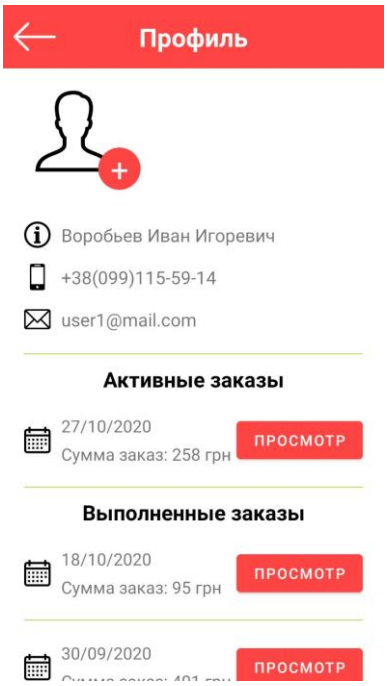

Рисунок 3.9 – Профіль користувача

У програмі всі дані користувача представляють собою клас «User.kt», з яким надалі можна легко працювати та зчитувати поля з бази даних.

```
data class User (
    @Exclude val uid: String = "")
    val name: String = "",
    val surname: String = "",
```

```
    val email: String = "",
    val phone: String? = "",
    val photo: String? = null
)
```

Для того, щоб користувач мов змогу додати фото свого профіля, потрібно прописати в маніфесті наступне, бо доступу до файлів власника мобільного пристрою Android Studio за замовчуванням не має.

```
<uses–permission
android:name="android.permission.READ_EXTERNAL_STORAGE">
</uses–permission>
<uses–permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE">
</uses–permission>
```

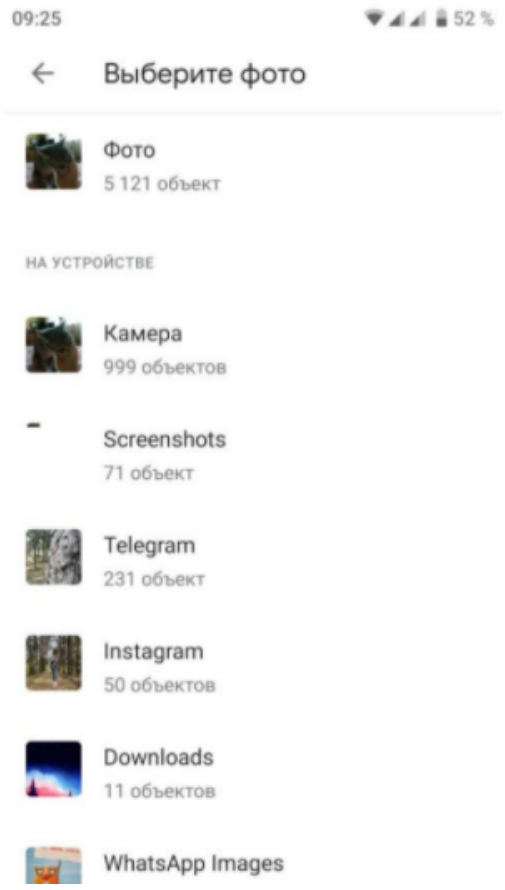

Рисунок 3.10 – Вибір фото з галереї

Для того, що можна було використати камеру для додавання фотографії, був написаний клас CameraHelper.kt, екземпляр якого створюється при запуску ShareCameraActivity. Також до маніфесту потрібно додати код, який попереджає, що у застосунку буде використана камера пристрою (див. рис. 3.11).

<uses–feature android:name="android.hardware.camera"
android:required="true" />

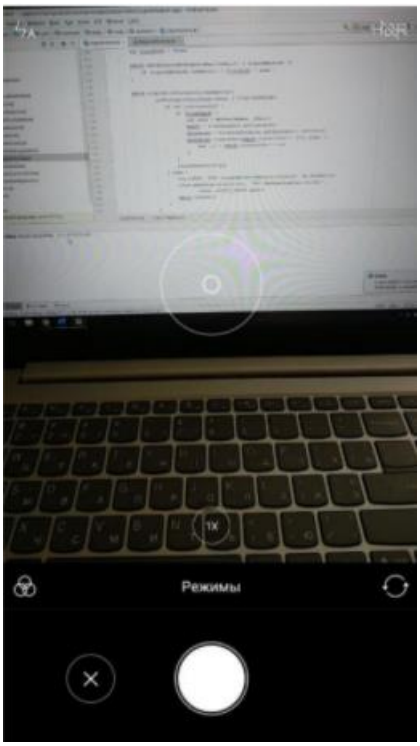

Рисунок 3.11 – Використання камери пристрою

Для кожного користувача програмно створюється папка де буде зберігатися файл (див. рис. 3.12), назва якої є унікальний ідентифікатор акаунта. А в самій базі даних Firebase Realtime Database зберігається лише посилання на відповідний файл.

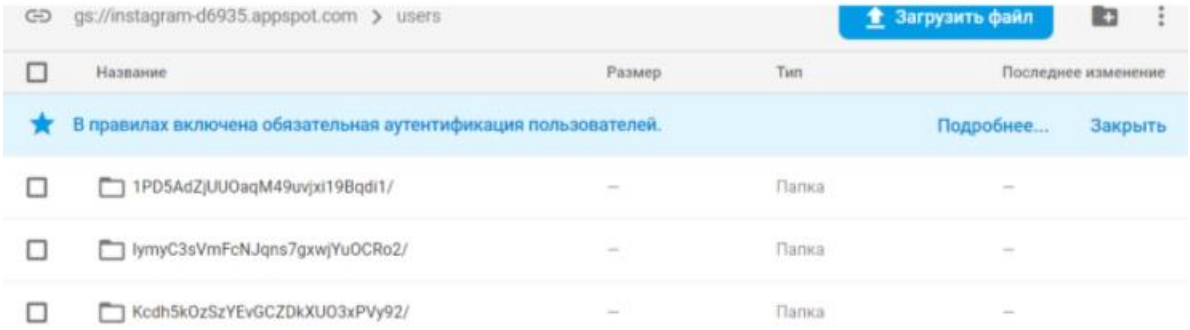

Рисунок 3.12 – Папки користувачів у Firebase Storage

Камера пристрою також використовується для сканування штрих–кодів. При першому запуску, додаток спитає дозвіл на використання камери (див. рис. 3.13).

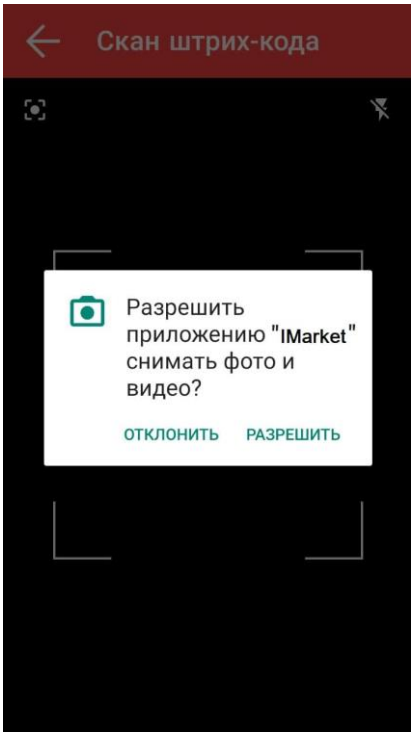

Рисунок 3.13 – Запит на використання камери

Екран що наведено вище, виконує фукнцію сканування кодів QR та штрих–кодів і є графічним елементом CodeScannerView (див. рис 3.14) із бібліотеки com.budiyev.android.codescanner.

```
<!--CONTENT-->
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintTop_toBottomOf="@id/container_toolbar"
    app:layout_constraintBottom_toBottomOf="parent">

    <com.budiyev.android.codescanner.CodeScannerView
        android:id="@+id/scanner_view"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:autoFocusButtonColor="@android:color/white"
        app:autoFocusButtonVisible="true"
        app:flashButtonColor="@android:color/white"
        app:flashButtonVisible="true"
        app:frameColor="@android:color/white"
        app:frameCornersSize="50dp"
        app:frameCornersRadius="0dp"
        app:frameAspectRatioWidth="1"
        app:frameAspectRatioHeight="1"
        app:frameSize="0.75"
        app:frameThickness="2dp"
        app:maskColor="#77000000"/>
</FrameLayout>
```

Рисунок 3.14 – Елемент CodeScannerView для сканування штрих–коду

Для того, щоб додати бібліотеку у проект, у конфігураційний файл gradle було додано її імплементацію. На сьогоднішній день, версія «2.1.0» є останньою:

implementation 'com.budiyev.android:code–scanner:2.1.0'.

Щоб відсканувати штрих–код товару, потрібно розмістити його у межах білої рамки (див. рис 3.15). Є можливість включення ліхтарика пристрою та автофокус.

Рисунок 3.15 – Сканування штрих–коду

Обробка штрих–коду відбувається у функції *decodeCallback* (див. рис. 3.16).

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    val scannerView = view.findViewById<CodeScannerView>(R.id.scanner_view)
    val activity = requireActivity()
    codeScanner = CodeScanner(activity, scannerView)
    codeScanner.decodeCallback = DecodeCallback { it: Result
        activity.runOnUiThread {
            val qr = it.text
            val product = findProductByQr(qr)
            if(product != null) viewModel.showProduct(product)
            else couponNotActivated()

        }
    }
    scannerView.setOnClickListener { it: View!
        codeScanner.startPreview()
    }
    onInit()
}
```

Рисунок 3.16 – Обробка зчитаного штрих–коду

Після успішного сканування, програма зробить співставлення зчитаного штрих–коду із тими, що є у базі даних. Після цього, перенаправить користувача на екран інформації про цей товар (див. рис. 3.17).

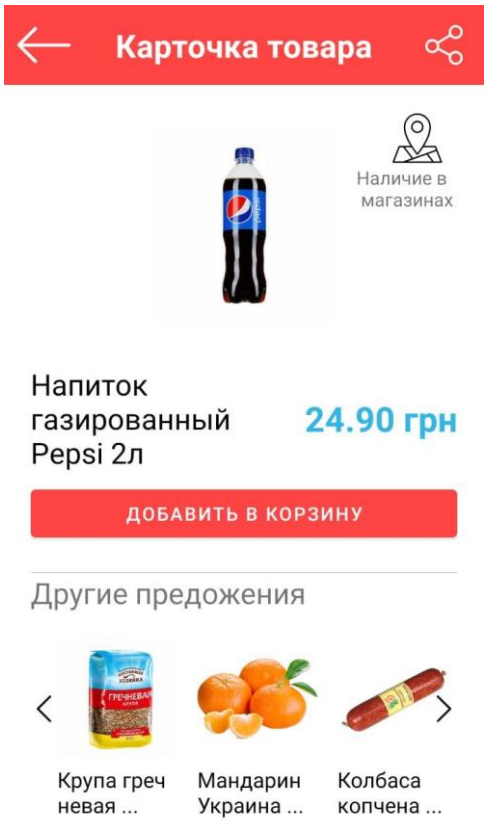

Рисунок 3.17 – Екран інформації про товар

На цьому екрані відображається фото товару, його ціна, назва. Користувач може додати його собі у кошик, натиснувши кнопку «Додати у кошик». Може поділитися посиланням на цей товар, натиснувши на кнопку, що розташована у верхньому тулбарі. Під сканованим товаром також є перелік, із декількох позицій, запропонованих товарів, шо були обрані випадковим чином. Цей маркетингових хід допоможе збільшити середній чек клієнта та час, що від проведе у додатку, переходячи з одних пропозицій на інші.

У користувача є можливість переглянути перелік магазинів, в котрих цей товар є в наявності на мапі. Для цього, йому потрібно натиснути на напис

або іконку «Наявність в магазинах». Він потрапить на екран із мапою, де червоними мітками позначено розташування магазинів (див. рис. 3.18).

Щоб повернутися на попередній екран, треба натиснути апаратну кнопку «повернутися» на своєму девайсі.

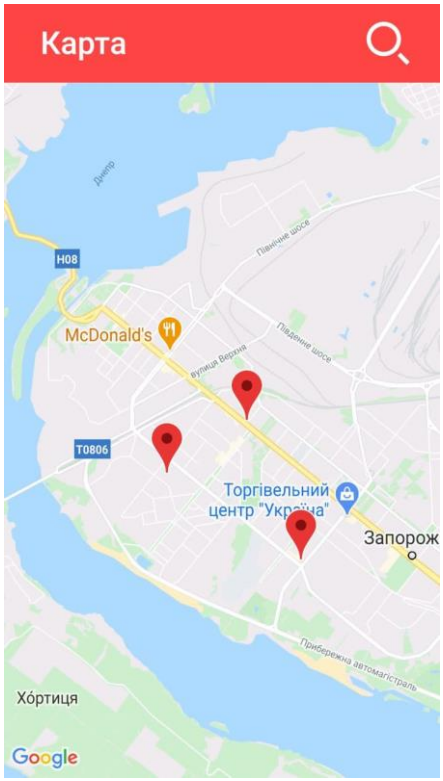

Рисунок 3.18 – Мапа з відмітками адрес магазинів

У базі даних у таблиці Product є перелік адрес магазинів, де цей товар є у наявності. Для кожного з цих записів, додається геомітка з відповідною широтою на довготою (див. рис. 3.19)

```
for(merchant in merchantList) {
    val location = LatLng(merchant.position.lat, merchant.position.lng)
    googleMap.addMarker(
        MarkerOptions()
            .position(location)
            .title(merchant.name)
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_RED))
    )
}
```

Рисунок 3.19 – Додання геоміток на мапу

Після того, як користувач переглянув всі необхідні йому товари та сформував кошик, він може переглянути його на сторінці «Кошик» (див. рис. 3.20).

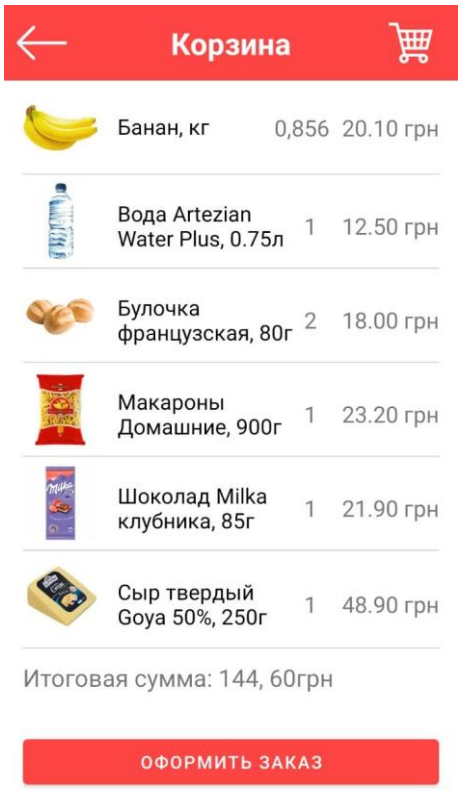

Рисунок 3.20 – Кошик користувача

У кошику користувач може переглянути підсумкову суму покупки, всі товари, що були додані, їх кількість або вагу, ціни, фотографію та видалити деякі якщо потрібно (свайпом вправо).

Якщо він натисне на будь–який товар із кошика – потрапить на сторінку інформації про нього (див. рис. 3.17).

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проаналізовано Android додатки для здійснення замовлень в мережі Інтернет, виявлені їх плюси і мінуси в роботі. Наведено переваги використання таких додатків для бізнесу: продемонстровано поширені ситуації, коли використання подібних технологій надають значні переваги для привернення уваги аудиторії, а в результаті збільшення обсягів продажу. На основі цих даних, було визначено вимоги для створюваного додатку.

За результатами роботи було поглиблено знання у розробленні програм для операційної системи Android. Був спроектований, реалізований та протестований мобільний додаток, що дозволяє оформлювати замовлення. Цей додаток був підключений до хмарної бази даних Firebase Realtime Database, де зберігаються всі дані і файли користувачів.

Дані роботи можна використовувати при проектуванні систем, для вивчення роботи хмарної бази даних Firebase Realtime Database а також Firebase Storage, дослідження Android–додатків, зокрема для створення інтернет магазинів, бо наведено основні способи реалізації роботи з кожним з них в межах середовища розробки Android Studio.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт для розробників Android developer.android.com [Електронний ресурс] – URL: https://developer.android.com
2. Офіційна документація Google для Firebase Datadabase [Електронний ресурс] – URL: https://firebase.google.com/
3. Java Vs. Kotlin: Which One Will Be the Best in 2019? [Електронний ресурс] – URL: https://dzone.com/articles/java–vs–kotlin–which–one–will–be–the–best–in–2019 (дата звернення: 02.09.2020)
4. Майстер–клас «5 переваг Kotlin» [Електронний ресурс] – URL: https://blog.ithillel.ua/ua/events/maister–klas–5–perevah–kotlin–v–dnipri (дата звернення: 04.09.2020)
5. Kotlin vs Java. [Електронний ресурс] – URL: Most Important Differences That You Must Know https://hackr.io/blog/kotlin–vs–java (дата звернення: 09.07.2020)
6. Developers, meet Firebase! [Електронний ресурс] – URL: https://firebase.googleblog.com/2012/04/developers–meet–firebase.html (дата звернення: 09.07.2020)
7. Интеграция Firebase в Android приложения [Електронний ресурс] – URL: https://codelabs.developers.google.com/codelabs/firebase–android–ru/index.html?index=..%2F..lang–ru#0 (дата звернення: 10.07.2020)
8. Начинаем работать с Firebase Cloud Storage на Android [Електронний ресурс] – URL: https://androiddev.apptractor.ru/nachinaem–rabotat–s–firebase–cloud–storage–na–android/ (дата звернення: 28.10.2020)
9. База данных реального времени Firebase: как установить / получить данные [Електронний ресурс] – URL: https://riptutorial.com/ru/android/example/21613 (дата звернення: 28.10.2020)

10. Firebase Android Series: Storage [Електронний ресурс] – URL: https://proandroiddev.com/firebase–android–series–storage–f09524008e14 (дата звернення: 15.10.2020)
11. Работа с RecyclerView [Електронний ресурс] – URL: https://devcolibri.com/unit/tweetsrecyclerview–2/ (дата звернення: 31.10.2020)