Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Нативний програмний додаток функціонування кав'ярні»

Студента 4 курсу, 7 групи,
спеціальності 121 «Інженерія програмного забезпечення»
освітньої програми «Інженерія програмного забезпечення» _____ Лук’янця Богдана Борисовича
підпис студента

Науковий керівник
старший викладач кафедри інженерії програмного забезпечення та кібербезпеки _____ Гнатченко Дмитро Дмитрович
підпис керівника

Гарант освітньої програми
доктор технічних наук,
доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Рзаєва Світлана Леонідівна
підпис гаранта

КИЇВ – 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О.В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Лук'янцю Богдану Борисовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Нативний програмний додаток функціонування кав'ярні»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту розробка програмного забезпечення для автоматизації процесів обліку замовлень та продажів кав'ярні; отримання навичок проєктування, розробки й реалізації якісного програмного продукту.

Об'єкт дослідження використання мобільних додатків в автоматизації бізнес-процесів роботи кав'ярні.

Предмет дослідження розробка нативного мобільного додатку на платформі ОС Android.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ВСТУП

РОЗДІЛ 1. ЛІТЕРАТУРНИЙ ОГЛЯД ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ НА МОБІЛЬНІЙ ПЛАТФОРМІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальна характеристика діяльності кав'ярні та опис проблеми

1.2. Сучасні принципи функціонування інформаційних систем на мобільній платформі

1.3. Огляд аналогів розробки

1.4. Технічне завдання

1.5. Висновок до розділу 1

РОЗДІЛ 2. МЕТОДИ ТА ЗАСОБИ ПРОЄКТУВАННЯ Й РОЗРОБКИ НАТИВНОГО ДОДАТКУ ФУНКЦІОНУВАННЯ КАВ'ЯРНІ

2.1. Проєктування нативного програмного додатку функціонування кав'ярні

2.2. Обґрунтування вибору платформи та мови програмування

2.3. Архітектура нативного додатку

2.4. Висновок до розділу 2

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ НАТИВНОГО ПРОГРАМНОГО ДОДАТКУ ФУНКЦІОНУВАННЯ КАВ'ЯРНІ

3.1. Структура програмного проєкту

3.2. Огляд розробленого додатку функціонування кав'ярні, його тестування та валідація

3.3. Висновок до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Літературний огляд використання інформаційних систем на мобільній платформі та постановка задачі*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Методи та засоби проєктування й розробки нативного додатку функціонування кав'ярні*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Практична реалізація нативного програмного додатку функціонування кав'ярні*	*14.04.2023*	*14.04.2023*
7.	*Висновки*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*	*29.05.2023 – 02.06.2023*	*31.05.2023*
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*05.06.2023*	*05.06.2023*
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*	*05.06.2023*	*05.06.2023*
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту _____ Гнатченко Д.Д.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми _____ Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент _____ Лук'янець Б.Б.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента _____Лук'янця Б.Б.____

(прізвище, ініціали)

може бути допущений до захисту екзаменаційній комісії.

Гарант освітньої програми _____Рзаєва С.Л._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О.В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці програмного забезпечення для автоматизації обліку замовлень та продажів кав'ярні. В результаті дослідження предметної області та порівняльного аналізу аналогічних програмних рішень визначено потребу у розробці мобільного додатку, орієнтованого на специфіку діяльності малого бізнесу, зокрема невеликих кав'ярень. Складено технічне завдання на розробку нативного мобільного Android-додатку. На основі останніх технологічних тенденцій було спроєктовано інтерфейс та логіку додатку і сформовано висновки та пропозиції щодо подальшого розвитку розробленого програмного забезпечення.

Результатом роботи є створений за допомогою сучасного інструментарію мобільний додаток на платформі операційної системи Android з інтуїтивно зрозумілим інтерфейсом, який повністю відповідає поставленому завданню та вимогам розробки. Готовий програмний додаток «Coffee Proffy» було успішно протестовано.

Ключові слова: автоматизація, кав'ярня, нативний мобільний додаток, Android-додаток.

ABSTRACT

The qualification work is devoted to the development of software for automating the accounting of orders and sales in a coffee shop. As a result of the study of the subject area and a comparative analysis of similar software solutions, the need to develop a mobile application focused on the specifics of small businesses, in particular small coffee shops, has been identified. The terms of reference for the development of a native Android mobile application were drawn up. Based on the latest technological trends, the interface and logic of the application were designed, and conclusions and suggestions for further development of the developed software were formed.

The result of the work is a mobile application based on the Android operating system platform with an intuitive interface that fully meets the task and development requirements. The finished software application, "Coffee Proffy", has been successfully tested.

Keywords: automation, coffee shop, native mobile application, Android application.

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ARM (англ. Advanced RISC Machines) – архітектура процесорів зі зменшеною набором інструкцій

CSS (англ. Cascading Style Sheets) – мова опису стилів, що використовується для візуального оформлення веб-сторінок

GPS (англ. Global Positioning System) – глобальна система позиціонування, що використовує супутникову навігацію для визначення місцезнаходження

HTML (англ. HyperText Markup Language) – мова розмітки гіпертексту, що використовується для створення веб-сторінок

IDE (англ. Integrated Development Environment) – інтегроване середовище розробки

JVM (англ. Java Virtual Machine) – віртуальна машина Java

БД – база даних

ІС – інформаційна система

ОС – операційна система

ПЗ – програмне забезпечення

ПК – персональний комп'ютер

ТЗ – технічне завдання

ЗМІСТ

					ДТЕУ 121 07-16.БР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		23.12.22	Нативний програмний додаток функціонування кав'ярні	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Гнатченко Д.Д.		23.12.22		*З*	*3*	*50*
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій 4 курс, 7 група*		
Розробив		Лук'янець Б.Б.		23.12.22	*Зміст*			

ВСТУП

Актуальність. Сучасні компанії, незалежно від розміру, мають перед собою безліч завдань, які надають клієнти, користувачі послуг та замовники. Виникає необхідність автоматизації повторюваних процесів та документообігу, систематизованої обробки й впорядкування даних.

Малий бізнес, не маючи у вільному розпорядженні значних грошових ресурсів, переважно використовує базові та ситуативні рішення автоматизації. Технічно їх можливо реалізувати через окремі сервіси, але набагато краще - об'єднати у повноцінне програмне забезпечення. Однак, більшість аналогічних програмних рішень з автоматизації розраховані лише на стаціонарні ПК або планшети.

Таким чином, виникає потреба у проєктуванні та розробці програмного забезпечення, яке зможе задовольнити потреби та особливості саме малого бізнесу і стане ефективно використовуватися в якості інструменту для організації та автоматизації багатьох бізнес-задач, зокрема обліку замовлень та продажів, із використанням відповідних технологій.

Мета дослідження: розробка програмного забезпечення для автоматизації обліку замовлень та продажів кав'ярні; отримання навичок проєктування, розробки й реалізації якісного програмного продукту.

Об'єкт дослідження: використання мобільних додатків для автоматизації бізнес-процесів роботи кав'ярні.

Предмет дослідження: розробка нативного мобільного додатку на платформі ОС Android.

У відповідності з метою дослідження поставлені наступні завдання:

					ДТЕУ 121 07-16.БР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		23.12.22	Нативний програмний додаток функціонування кав'ярні	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Гнатченко Д.Д.		23.12.22		*В*	*4*	*50*
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій*		
Розробив		Лук'янець Б.Б.		23.12.22	*Вступ*	*4 курс, 7 група*		

- провести дослідження предметної області кав’ярні, з виділенням специфіки її функціонування та бізнес-процесів, які підлягатимуть автоматизації;
- скласти технічне завдання, спираючись на результати проведеного аналізу;
- спроєктувати логічний (функціональний) та графічний дизайн додатку;
- за допомогою сучасного технічного інструментарію розробити мобільний застосунок, який відповідатиме вимогам, визначеним у ТЗ;
- провести тестування та впровадження додатку;
- сформувати висновки та пропозиції щодо подальшого розвитку та модифікації створеного ПЗ відповідно до досвіду розробки та експлуатації.

Методи дослідження і технології розробки: порівняння й аналіз, проєктування, візуалізація, розробка інтерфейсу користувача, програмування на мові Kotlin в середовищі розробки Android Studio з використанням об’єктно-орієнтованого підходу.

РОЗДІЛ 1
ЛІТЕРАТУРНИЙ ОГЛЯД ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ СИСТЕМ НА МОБІЛЬНІЙ ПЛАТФОРМІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальна характеристика діяльності кав’ярні та опис проблеми

Кав’ярня – це заклад громадського харчування, який спеціалізується на продажі кави високої якості, а також інших гарячих та холодних напоїв, десертів та кулінарних виробів легкого приготування. Компонентами даної предметної області є співробітники, замовлення та пропонований товар.

Бізнес-процеси кав'ярні зазвичай охоплюють ведення та обробку замовлень, управління персоналом, закупівлю та контроль продукції, фінансовий облік, маркетинг та рекламу тощо. Від їх ефективності безпосередньо залежить конкурентоспроможність і прибутковість закладу. Автоматизація цих процесів допоможе звільнити ресурси персоналу від рутинних повторюваних завдань, таких як ручне ведення замовлень та документації, і дозволить більше фокусуватися на взаємодії з клієнтами та інших важливих аспектах бізнесу. Загалом, автоматизація робочих процесів може допомогти покращити ефективність, знизити кількість помилок, оптимізувати управління запасами, покращити аналітику та звітність, що призведе до зниження витрат, покращення якості обслуговування, збільшення прибутковості та конкурентоспроможності кав'ярні. Саме тому автоматизація бізнес-процесів кав’ярні є актуальним завдання для працівників та власників закладів цієї спеціалізації.

Однак процес автоматизації повинен бути ретельно спланований, з урахуванням витрат на впровадження, навчання персоналу та інтеграцію систем.

Також необхідно враховувати специфіку бізнесу, його масштаби, сферу діяльності, потреби та ресурси. Тож далі будуть описані основні аспекти роботи середньостатистичної кав’ярні в Україні, які, однак, можуть варіюватися в залежності від розташування, розміру та концепції закладу.

Організаційна структура кав'ярні, побудована на принципах ієрархічності рівнів управління та поділу праці на окремі ланки спеціалізації працівників із виконуваних функцій [20]. Чисельність персоналу визначається в залежності від функціональної доцільності, зазвичай це від 1 до 6 осіб. Це можуть бути бариста, касир, а також особи відповідальні за приготування кулінарних виробів, адміністратор, який контролює роботу персоналу та забезпечує комунікацію з клієнтами, або прибиральники, охоронці та інші працівники, залежно від потреб кав'ярні.

Зазвичай кав’ярні працюють з 9:00 до 22:00, з можливими варіаціями в робочих годинах в різні дні тижня. Потік клієнтів може бути різним, але в середньому кав'ярня може обслуговувати від 50 до 200 людей на день. Найбільші навантаження зазвичай бувають в період ранкової та обідньої годин, коли багато людей зупиняються на швидкий сніданок або обід, а також у вихідні. Ціни на напої, десерти та закуски у кав’ярнях України стартують від 25-50 грн в залежності від виду, розміру порції, основних та додаткових інгредієнтів, рівня обслуговування, розташування тощо. Загалом, щомісячний прибуток кав'ярні складає в середньому від 20 до 150 тис. грн.

Ведення замовлень у кав'ярнях може бути ручним або автоматизованим. Ручне ведення замовлень може включати використання паперових або електронних чеків, які потім вручну вводяться в систему каси або облікової програми. Автоматизоване ведення реалізується за допомогою спеціалізованих касових апаратів або програмного забезпечення, яке дозволяє клієнтам робити замовлення самостійно на сайті кав'ярні або використовуючи спеціалізовані настільні пристрої.

Отже, основними особливостями роботи кав’ярні є вузька спеціалізація; невеликий штат співробітників, обсяг послуг та площа їх надання; необхідність швидкого й ефективного обслуговування клієнтів та задоволення потреб цільової аудиторії, зокрема за допомогою цінової політики. Враховуючи цю специфіку, необхідною є розробка автоматизованої системи обліку роботи та продажів кав'ярні з можливостями розрахунку розмірів та вартості, а також зберігання замовлень.

Найкращим рішення є створення програмного забезпечення, яке буде зручним для працівників закладу й не потребуватиме додаткових витрат. Його застосування буде мати безліч переваг: скорочення витрат часу та ресурсів, збільшення точність та швидкість обробки інформації, покращення контролю продажів тощо. А оскільки автоматизована система аналітики та звітності допоможе збирати, аналізувати та відображати дані про продажі та інші показники ефективності кав'ярні, це створить можливості для оперативного виявлення актуальних тенденцій бізнесу та вимог клієнтів. Адміністрації кав'ярні використання подібного функціоналу надасть важливу інформацію для прийняття рішень для подальшої оптимізації бізнес-процесів та стратегії розвитку кав’ярні.

1.2. Сучасні принципи функціонування інформаційних систем на мобільній платформі

Інформаційні системи – це сукупність взаємопов'язаних компонентів, які збирають, обробляють, зберігають і поширюють дані для досягнення поставленої мети у процесі управління. Вони використовують обладнання, програмне забезпечення, дані та телекомунікаційні мережі для автоматизації та інтеграції бізнес-процесів.

Існують різні типи інформаційних систем, які можна класифікувати за

їхньою функціональністю та сферою застосування. Залежно від ступеня автоматизації розрізняють ручні, автоматизовані та автоматичні інформаційні системи.

Автоматизована система – це система, яка реалізує виконання встановлених функцій за допомогою інформаційних технологій, керованих комплексними автоматизованими засобами й частково вручну.

Мобільні інформаційні системи — це програмні додатки, які працюють на мобільних пристроях, таких як смартфони та планшети, і розробляються з урахуванням їх унікальних особливостей.

Сучасні аспекти функціонування інформаційних систем на мобільній платформі зосереджені на принципах гнучкої розробки високодоступних, безпечних та інтуїтивно зрозумілих додатків, які можна ефективно налаштовувати, для стимулювання розвитку бізнесу та покращення процесу прийняття рішень. Засади функціонування мобільних ІС також передбачають використання:

- адаптивного дизайну, який забезпечує оптимізацію додатків під різні розміри екранів і роздільну здатність;
- хмарних технологій, які дозволяють зберігати дані та отримувати до них доступ з будь-якого місця з наявним Інтернет-підключенням;
- GPS для забезпечення персоналізованості контенту.

Одним з різновидів мобільних інформаційних систем є мобільний додаток – спеціально розроблене програмне забезпечення, яке надає користувачам доступ до певних послуг або інформації. Їх можна завантажити та встановити з спеціалізованих магазинів додатків або інших онлайн-платформ. Мобільні додатки можна розробляти за допомогою різних інструментів і технологій, з використанням нативного, веб та гібридного підходів.

Нативна розробка додатків передбачає створення програми спеціально для певної мобільної операційної системи, наприклад, Android або iOS. Таке ПЗ

пишеться мовою та фреймворком, характерними для даної платформи, і має доступ до повного спектру апаратних і програмних функцій пристрою.

Веб-додатки – це програмне забезпечення, яке, по суті, є оптимізованим для мобільних пристроїв веб-сайтом і використовується через браузер. Зазвичай вони написані на HTML, CSS та/або JavaScript. Веб-додатки прості в обслуговуванні, не потребують інсталяції та доступні з будь-якого пристрою, підключеного до Інтернету.

Гібридні додатки – це поєднання нативних і веб-додатків. Вони комбінують переваги веб-розробки з можливістю доступу до функціональних можливостей пристроїв та кросплатформну сумісність. Такий підхід може заощадити час і витрати на розробку, але може призвести до того, що додатки будуть менш оптимізовані та адаптивні.

Зрештою, вибір типу мобільного додатку залежить від конкретних потреб і цілей, а також від таких факторів, як бюджет або цільова аудиторія. Однак значний ринковий вплив мають саме нативні рішення розробки, особливо в контексті бізнес-додатків. Така популярність ґрунтується на низці переваг, які пропонує нативна технологія розробки, а саме:

Нативні додатки оптимізовані для конкретної платформи та написані її «рідною» мовою, тому вони працюють швидше та ефективніше порівняно з іншими типами застосунків. Це може бути особливо важливо для програмного забезпечення, яке вимагає інтенсивної обробки великої кількості даних для управління.

Також однією з головних переваг нативних мобільних додатків для малого бізнесу є те, що вони пропонують високий рівень кастомізації та функціональності. Нативні застосунки можуть бути розроблені відповідно до конкретних потреб і вимог бізнесу та інтегруватися з іншими інформаційними системами, такими як інструменти управління взаємовідносинами з клієнтами або бухгалтерське програмне забезпечення.

Ще однією перевагою нативних додатків є їхня здатність працювати в автономному режимі, оскільки вони здатні зберігати дані на самому пристрої, дозволяючи користувачам отримувати доступ до програми та користуватися нею, навіть якщо у них немає доступу до інтернету. Окрім цього, вони можуть використовувати вбудовані функції безпеки конкретної платформи і їх можна поширювати через офіційні магазини додатків, що забезпечує додатковий рівень безпеки та надійності.

Однак головною перевагою розробки саме нативного мобільного додатку в контексті автоматизації системи обліку продажів кав'ярні є те, що він доступний для встановлення на смартфон будь-якого співробітника кав'ярні, без потреби придбання окремого пристрою.

Використовуючи ці переваги, малий бізнес може ефективніше взаємодіяти зі своїми клієнтами, забезпечити кращий користувацький досвід, а також приймати більш обґрунтовані бізнес-рішення. Враховуючи вищенаведені факти та специфіку бізнес-процесів кав'ярні, було прийнято рішення про розробку саме нативного мобільного додатку.

1.3. Огляд аналогів розробки

Існує досить багато додатків та сервісів для автоматизації процесів роботи ресторанів, кафе та кав'ярень.

Однією з таких автоматизованих систем є «Poster». Це програмне забезпечення, яке встановлюється на планшетні пристрої і підтримує під'єднання додаткового обладнання для автоматизації, наприклад термінала чи ваг [13]. Ця система обліку виконує функції повноцінного касового пристрою, а також має інтегровані програми для доставлення та модифікації замовлень, аналітики й інвентаризації. Користуватися цим програмним забезпеченням можна по передплаті у 480 – 1770 грн на місяць в залежності від функціоналу.

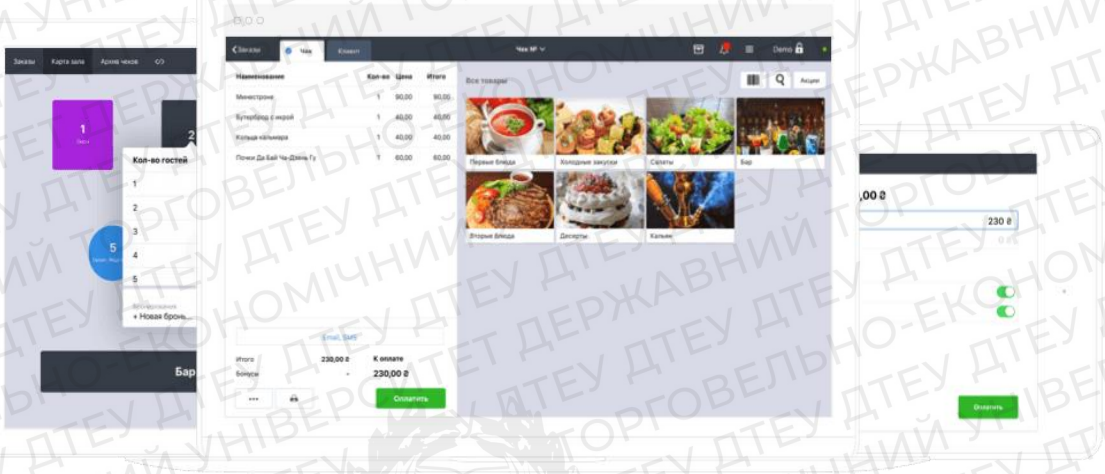

Рис. 1.1. Вигляд головного екрану системи Poster

Однак, як зазначалось раніше, повноцінна версія даної системи доступна лише для планшетів, на пристроях з меншою діагоналлю екрану, наприклад смартфонах, можна лише переглядати актуальну статистику та налаштувати сповіщення про касові зміни та транзакції, що створює необхідність у закупці відповідного обладнання. Також слід враховувати обмеження або навпаки перенасиченість функціоналу для потреб невеликої кав'ярні у залежності від тарифного плану. Так, за стартовою ціною, існує обмеження меню у 50 позицій і відсутня функція фінансового обліку. При цьому у даному тарифному пакеті є доступ до контролю закупівель та інших опцій інвентаризації, що не завжди є актуальною потребою для малого бізнесу.

Іншою системою з аналогічним функціоналом є «Palma Box». Дане програмне забезпечення встановлюється на планшет або ноутбук і виконує функції каси або мобільного терміналу [17].

Рис. 1.2. Вигляд інтерфейсу Palma Box на різних пристроях

Ця система також надає можливості складського обліку, інтегрованого меню, аналітики та проведення касових операцій і передбачає поділ користувачів за правами доступу (в програму включено електронне меню для клієнтів закладу та окрема версія для його працівників).

Palma Box більш оптимізована під роботу кав'ярень, що робить її функціонал ефективнішим. Однак пропоновані тарифні плани стартують від 550 грн на місяць, а програмне забезпечення все ще не розраховане під повноцінне використання зі смартфона.

Отже, розглянуті системи автоматизації мають недостатню оптимізацію під специфіку невеликих кав'ярень, які обмежені в своїх фінансових можливостях і потребують ефективного цільового функціоналу з максимальною доступністю реалізації. Це створює необхідність розробки програмного забезпечення саме для мобільних пристроїв з невеликими діагоналями, яке буде мати необмежений функціонал обліку замовлень та продажів, інтегрованого меню й аналітики.

1.4. Технічне завдання

1. Загальні відомості

1.1. Найменування системи

Найменування програми – мобільний додаток «Coffee Proffy»

1.2. Планові терміни початку та закінчення робіт

В результаті проведеного аналізу предметної області та проєктування всіх етапів розробки програмного продукту, було визначено плановий термін виконання: 28 січня 2023 – 14 квітня 2023 року.

1.3. Головний бенефіціар та потенційні користувачі системи

Головний бенефіціар відсутній.

Мобільний додаток розрахований на користування однією особою. Потенційними користувачами додатку будуть працівники кав'ярень або невеликих кафе та ресторанів. Кваліфікація основного користувача мобільного додатку повинна оцінюватись як задовільна для роботи зі стандартними мобільними програмами.

2. Мета та призначення створення системи

2.1. Призначення системи

Мобільний додаток «Coffee Proffy» призначений для ведення обліку замовлень і їх зберігання у базі даних для подальшого загального аналізу роботи кав'ярні, проєктування вартості складових меню та формування стратегії розвитку бізнесу на основі отриманих даних.

2.2. Мета створення системи

Метою створення додатку є автоматизації обліку замовлень та продажів кав'ярні, що дозволить оптимізувати її роботу, ефективніше взаємодіяти з клієнтами, а також приймати більш обґрунтовані бізнес-рішення.

3. Вимоги до системи, на якій буде здійснюватися робота з програмним забезпеченням

Для коректної роботи мобільного додатку «Coffee Proffy» рекомендовано використовувати смартфон, який працює на платформі ОС Android не старше 7 версії, з можливістю підключення до мережі Інтернет.

4. Вимоги до програмного забезпечення

4.1. Основні вимоги до функціональних характеристик

Додаток має задовольнити мету свого створення, а отже обов'язково повинен підтримувати функціонал для створення, зберігання та видалення замовлень в локальній базі даних й аналізу роботи кав'ярні на основі ключових параметрів її діяльності, таких як кількість, вартість та час замовлень.

Режим управління системою користувача включатиме підрозділи, які відповідатимуть функціоналу, навігація між ними відбуватиметься відповідно до виконаних дій.

4.2. Структура проєкту «Coffee Proffy»

4.2.1. Вхідний екран завантажування

Проміжний екран з назвою додатку, який відображається під час завантаження баз даних та налаштувань. Після цього змінюється на екран «Головне меню».

4.2.2. Головне меню додатку

В головному меню розташовано список всіх розділів додатку для навігації по ньому. Після вибору необхідного розділу, здійснюється перехід на екран відповідної сторінки.

Структурні елементи головного меню:

- Нове замовлення;
- Журнал замовлень;
- Аналіз роботи кав'ярні;
- Розділи меню;
- Налаштування.

4.2.3. Сторінка «Нове замовлення»

Екран для додавання нового замовлення. Для формування складу замовлення на екрані відображатиметься кнопка, що відкриватиме список всіх позицій меню з можливістю їх додавання або видалення із замовлення. При додаванні нового замовлення йому буде присвоєно номер пов’язаний з поточною датою створення, а також обраховано його загальну вартість, згідно з цінами позицій меню. Після підтвердження замовлення, воно зберігатиметься у базі даних та здійснюватиметься автоматичне повернення до головного меню.

4.2.4. Сторінка «Журнал замовлень»

На екрані цієї сторінки буде розміщуватись перелік усіх введених замовлень з можливістю їх видалення та встановлення обраного діапазону дат в якості фільтра. Для кожного замовлення можна буде переглянути дані введені на етапі його додавання. Натиснувши на кнопку «Деталі замовлення», буде здійснено перехід на однойменний екран для перегляду деталізованого опису складу замовлення та його редагування за необхідності.

4.2.5. Сторінка «Аналіз роботи»

Екран для перегляду ключових показників роботи кав’ярні: кількості та вартості замовлень, п’яти найпопулярніших позицій меню тощо. Також передбачено функцію вибору діапазону дат в якості фільтра.

4.2.6. Сторінка «Розділи меню»

Екран для перегляду, додавання та видалення розділів меню, позицій меню та їх вартості.

4.2.7. Сторінка «Налаштування»

На даній сторінці буде розміщуватись декілька блоків для налаштування додатку «Coffee Proffy» :

- Блок для редагування назви кав’ярні, яка за замовчуванням використовується на екрані звантаження додатку;
- Блок для редагування паролю, де буде розташовуватись перемикач «Так/Ні» (можливість здійснення входу до застосунку без

додаткового захисту у вигляді паролю) та поля для вводу та підтвердження змінених даних для входу;

- Блок для налаштування відображення поточної загальної кількості та вартості замовлень на екрані головного меню, реалізований у вигляді перемикача «Так/Ні».

5. Вимоги до інтерфейсу користувача

Основною вимогою до графічного дизайну та загального виду елементів користувацького інтерфейсу є їх неперевантаженість і максимально можлива інформативність використовуваних зображень задля створення інтуїтивно зрозумілого дизайну.

6. Вимоги до технічного забезпечення

Інструменти розробки:

- середовище – Android Studio;
- мова програмування – Kotlin;
- шаблон проєктування архітектури додатку – MVVM;
- система збірки – Gradle;
- база даних – бібліотека Room ;
- навігація за допомогою бібліотеки Navigations component.

7. Засоби реалізації

Оскільки програмний продукт розробляється для смартфонів на платформі ОС Android, встановлення на пристрій виконуватиметься через ліцензійний магазин платформи – Play Market. Оптимізації під інші мобільні платформи не передбачено.

Дані про замовлення зберігатимуться у локальній базі даних на пристрої користувача.

1.5. Висновок до розділу 1

Проведений аналіз діяльності кав'ярні та специфіки її бізнес-процесів довів актуальність розробки автоматизованої системи обліку. В результаті порівняльного аналізу аналогічних програмних рішень обґрунтовано доцільність створення нативного мобільного додатку з можливостями розрахунку розмірів, кількості та вартості замовлень, їх зберігання та аналізу.

На основі проведеного дослідження також було визначено ключові завдання, що потребують вирішення в рамках цього проєкту, та функціональні вимоги до майбутнього продукту. Сформоване детальне технічне завдання допоможе розробити якісний програмний додаток з інтуїтивно зрозумілим інтерфейсом, що відповідатиме усім ключовим функціональним вимогам.

Розробка цього додатку має за мету покращення роботи кав'ярні внаслідок оптимізації бізнес-процесів, збільшення ефективності взаємодії з клієнтами та прийняття більш обґрунтованих бізнес-рішень.

РОЗДІЛ 2
МЕТОДИ ТА ЗАСОБИ ПРОЄКТУВАННЯ Й РОЗРОБКИ НАТИВНОГО ДОДАТКУ ФУНКЦІОНУВАННЯ КАВ'ЯРНІ

2.1. Проєктування нативного програмного додатку функціонування кав'ярні

Розробка будь-якого програмного забезпечення умовно поділяється на декілька етапів [18]:

1) Аналітика;
2) Проєктування;
3) Дизайн;
4) Розробка;
5) Тестування;
6) Реліз;

Кожен з них має свою специфіку і впливає на якість та успішність впровадження програмного продукту.

Етап проєктування мобільного додатку відіграє важливу роль у створенні успішного продукту, оскільки він визначає ідею, призначення, вигляд та функціональність майбутнього застосунку. На цьому етапі також створюються макети користувацького інтерфейсу, функціональної та візуальної архітектури ПЗ та технічні специфікації, що дозволяє знизити можливість появи помилок, зокрема пов'язаних з взаємозв'язком компонентів та оптимальним використанням ресурсів. Крім того, правильне проєктування зменшує час, необхідний для виконання робіт на наступних етапах, таких як програмування, тестування та впровадження.

Проєктування додатку починається з детального дослідження предметної області та цільової аудиторії, її потреб, вимог та поведінки. Також визначаються технічні завдання та функціональні можливості додатку. Аналіз та визначення вищевказаних параметрів було проведено першому розділі даної роботи.

Наступним етапом проєктування є розробка прототипу, який відображає основні елементи, структуру та навігацію мобільного застосунку. Метою є формування зручного та привабливого користувацького інтерфейсу з ефективною та інтуїтивно зрозумілою навігацією і логічною структурою.

Етап функціонального та технічного проєктування являє собою визначення основних модулів і необхідних опції додатку, а також технічних аспектів його розробки, таких як вибір платформи, мови програмування, схеми і виду бази даних, системної архітектури, інтеграції зі сторонніми сервісами тощо. Більш детальний огляд всіх аспектів цього етапу та обґрунтування вибору буде здійснено далі в цьому розділі.

Отже, відповідно до технічного завдання, спроєктовано прототип користувацького інтерфейсу мобільного додатку «Coffee Proffy»:

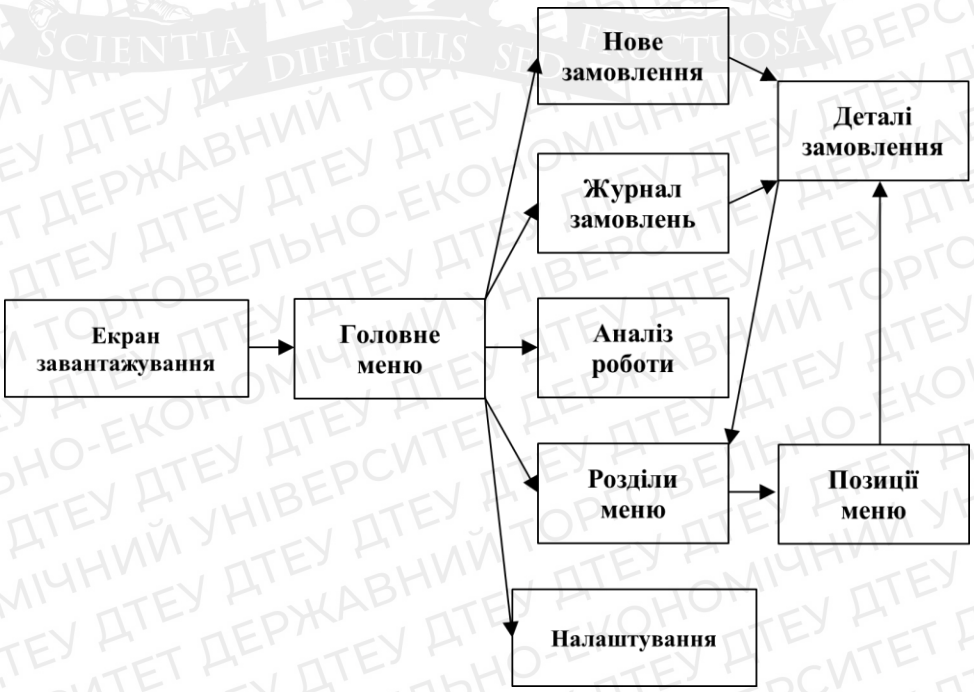

Рис. 2.1. Структурна схема взаємодії екранів додатку «Coffee Proffy»

Для реалізації наведеної на рисунку 2.1 взаємодії необхідно створити відповідні інтерактивні елементами для кожного екрану, які будуть певним чином реагувати на дії користувача. У наступному розділі буде більш детально розглянуто особливості даних структурних елементів, а також кінцевий вигляд вищевказаних екранів.

2.2. Обґрунтування вибору платформи та мови програмування

Android і iOS є найпопулярнішими операційними системами для мобільних пристроїв.

Android – це операційна система для мобільних пристроїв, розроблена компанією Google, яка базується на ядрі Linux і має відкритий вихідний код, що створює можливості для його модифікації та доповнення. Android також дозволяє розробляти мобільні додатки, які можуть працювати на пристроях з різними характеристиками та розмірами екранів.

iOS – це мобільна операційна система, розроблена компанією Apple для їхніх пристроїв, зокрема iPhone, iPad та iPod Touch. Вона працює на основі архітектури ARM і має своє власне середовище розробки (Xcode) та мову програмування (Swift).

Розробка програмного забезпечення для iOS зазвичай складніша та вимагає більших вкладень, ніж для Android. Це пов’язано з тим, що, як зазначалось раніше, розробка застосунків для iOS вимагає використання IDE Xcode, яке доступне тільки для Mac, що створює необхідність додаткових витрат на купівлю відповідного пристрою чи оренду хмарної інфраструктури [6].

Проведемо аналіз ринку мобільних ОС в Україні за останні три роки.

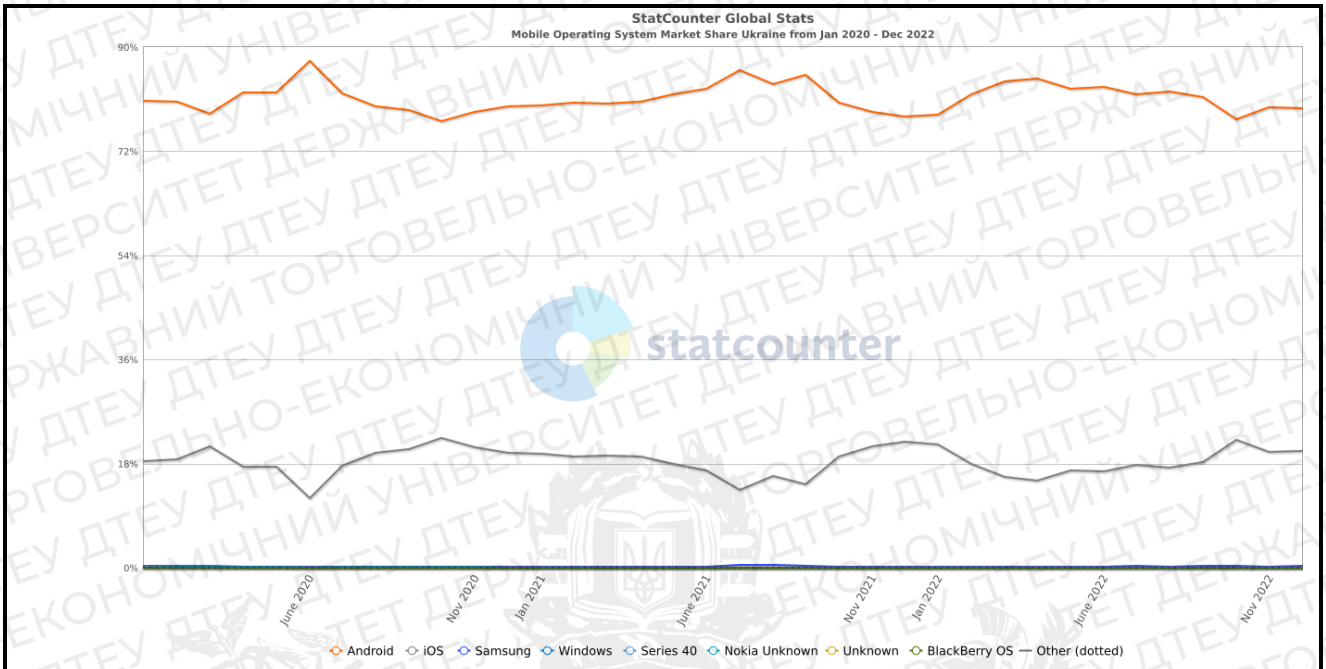

Рис. 2.2. Рейтинг використання мобільних ОС (2020 - 2023 рр.) [9].

Наведений графік статистичних показників свідчить про те, що операційну систему Android в Україні стабільно використовують понад 80% усіх користувачів. У той час як IOS посідає друге місце з часткою відставання у майже 60%. А програмні продукти на основі інших мобільних операційних систем використовують менше ніж 1% споживачів. Такий відрив обумовлюється тим, що ОС Android, окрім зазначених раніше особливостей, підтримує велику кількість пристроїв різних виробників, забезпечує більшу гнучкість вибору для користувачів та має інтеграцію з продуктами Google.

Отже, задля досягнення максимальної доступності майбутнього програмного продукту, доцільною є розробка мобільного додатку із використанням технологій нативної розробки саме на платформі ОС Android.

Android Studio – офіційне інтегроване середовище розробки виробництва Google, за допомогою якого розробникам стають доступні потужні інструменти та ресурси для створення, тестування, налагодження та публікації додатків на платформі ОС Android [5].

Android Studio базується на системі автоматичного збирання Gradle. Ця система була розроблена для збірок великих проєктів, з можливістю їх

подальшого масштабування. За допомогою набору конфігураційних файлів вона може визначати, які компоненти проєкту були змінені й потребують нової збірки, а які не потрібно перезапускати. Це значно спрощує і пришвидшує процес розробки програмного забезпечення.

Основними особливостями Android Studio є:

- розширений редактор коду (з підтримкою Java і Kotlin) надає функції автодоповнення, перевірку синтаксису, рефакторингу коду та інші корисні інструменти для зручної розробки;
- інтегрований редактор макетів, що дозволяє візуально створювати та редагувати користувацький інтерфейс, налаштовувати його властивості та елементи;
- вбудовані емулятори для легкого тестування додатків на віртуальних пристроях з різними версіями Android та конфігураціями. Крім того, існує функція підключення фізичного пристрою для проведення прямого тестування додатків;
- інтегровані інструменти для автоматичного тестування додатків;
- підтримка інших популярних інструментів розробки;
- вбудована система контролю версій Git, яка дозволяє ефективно керувати версіями коду, здійснювати коміти, розгалуження, злиття та інші операції з управління версіями безпосередньо з IDE;
- засоби для збірки та розгортання додатків, які дозволяють безпосередньо генерувати APK-файл для розповсюдження додатку, а також виконувати налаштування для Google Play Store або інших магазинів додатків.

Найчастіше для програмування мобільних додатків використовується Java. Це загальноприйнята, об'єктно-орієнтована мова програмування, яка була розроблена компанією Sun Microsystems у 1995 році та стала однією з найпопулярніших мов програмування у світі.

Єдиним конкурентним аналогом Java є Kotlin. Це сучасна статично типізована мова програмування, яка працює на JVM, що виконує байт-код програм, написаних на мові Java. Вона була розроблена компанією JetBrains з метою створення більш ефективної та зручної альтернативи мові Java. З 2017 року Kotlin є офіційною мовою для розробки Android-додатків і має глибоку інтеграцію з Android Studio.

Зараз використання Java для будь-яких завдань з мобільної розробки є нераціональним, оскільки не можна сподіватись, що проєкт, розроблений цією мовою, буде достатньо гнучким, високопродуктивним та швидким. Основними недоліками є необхідність використання нескінченних блоків «try-catch», а також проблеми з розширюваністю, відсутності підтримки функцій та прогалини у безпеці. З цих причин Kotlin за багатьма параметрами перевершує Java.

Проаналізовано індекс вподобання мов програмування для мобільної розробки (відношення розробників, які для свого наступного проєкту повторно її оберуть):

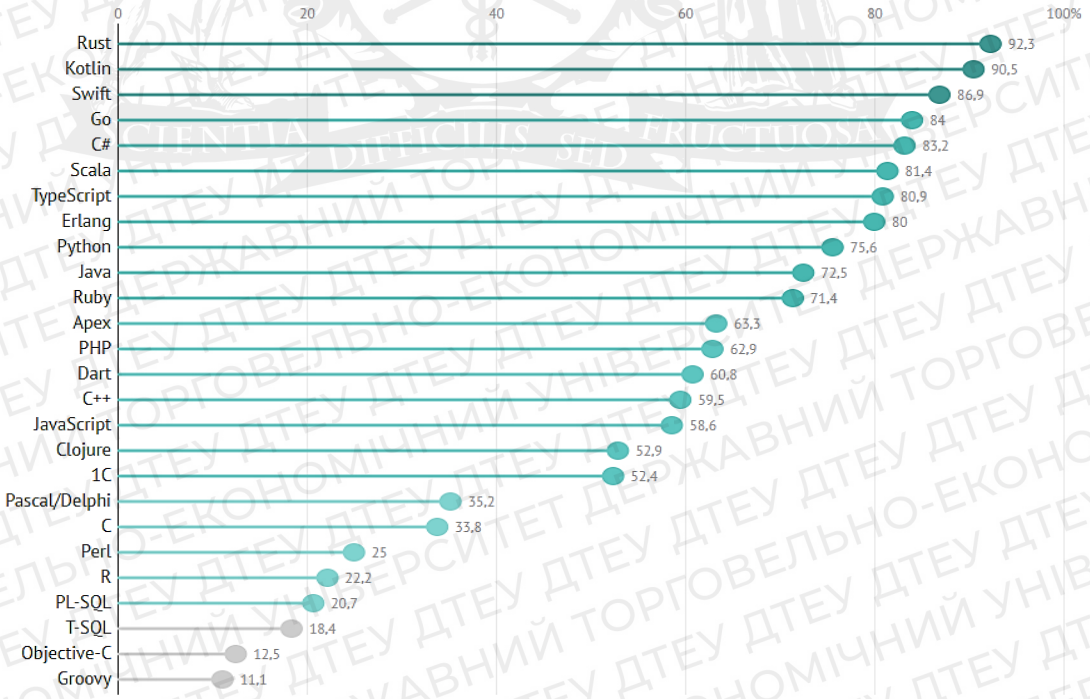

Рис. 2.3. Рейтинг мов програмування для мобільної розробки за індексом вподобання розробників [19]

Отже, Kotlin визнано однією з найкомфортніших мов для мобільної розробки. Вона займає друге місце після Rust з рейтингом у 90,5% вподобань. Це пояснюється тим, що вона добре спеціалізована для своєї предметної області, а отже має багато переваг.

Зокрема, однією з ключових переваг Kotlin є її простий синтаксис, що дозволяє писати більш компактний і зрозумілий код.

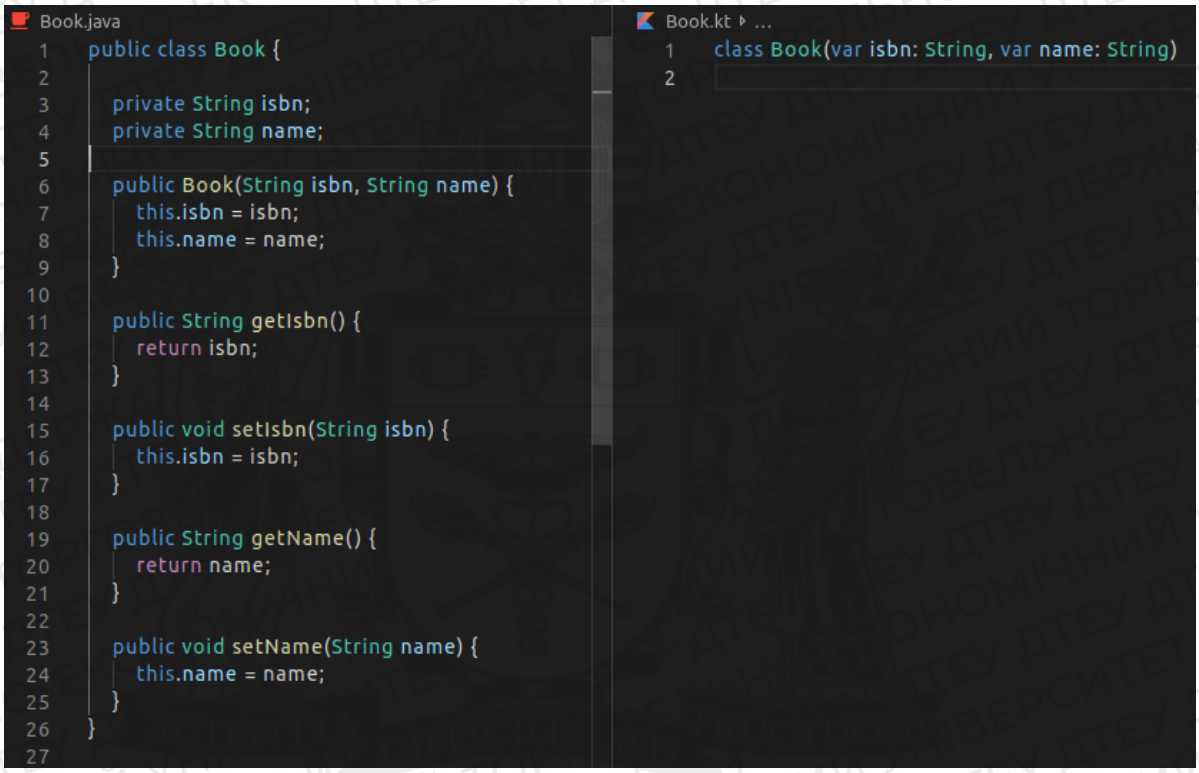

Рис. 2.4. Порівняння однакового за функціональністю коду на Java та Kotlin [8]

При цьому Kotlin повністю сумісна з Java, що створює можливості для легкої інтеграції існуючого Java-коду в Kotlin-проєкт без виникнення будь-яких проблем.

Крім того, Kotlin:

- підтримує функціональне програмування, а отже, дозволяє використовувати концепції функцій вищого порядку, лямбда-виразів і корутин та створює можливості для асинхронних операцій.

- надає вбудовану підтримку для роботи з нульовими значеннями, що робить код більш безпечним і унеможливлює появу багатьох типових помилок, пов'язаних з нульовими посиланнями.
- дозволяє створювати розширення для вже існуючих класів, навіть без доступу до вихідного коду.
- має велику спільноту розробників, що означає наявність багатьох допоміжних ресурсів та документації.

Також, як офіційні інструменти, Kotlin у поєднанні з Android Studio забезпечують багатофункціональне середовище для програмування Android-додатків, що робить розробку більш зручною та ефективною та дозволяє створювати чистіший, експресивний і підтримуваний код.

Загалом, Kotlin є потужним і прогресивним інструментом для розробки мобільних додатків на платформі Android

Враховуючи ці переваги, прийнято рішення вести розробку мобільного Android-додатку «Coffee Proffy» у середовищі Android Studio мовою програмування Kotlin, використовуючи систему збірки Gradle.

2.3. Архітектура нативного додатку

Архітектура програмного забезпечення – це визначення структури, організації та взаємодії компонентів ПЗ. Вона визначає основні принципи, шаблони, протоколи та підходи до проєктування системи. Загалом, це процес перетворення характеристик ПЗ в структуроване рішення, яке відповідає бізнес-потребам і технічним вимогам.

При проєктуванні архітектури необхідно розбити систему на модулі, а модулі – на підмодулі, класи та інші менші елементи. При групуванні модулів необхідно стежити за тим, щоб кожен із них був розроблений під певну функцію для досягнення тісного зв'язку всередині модуля, але при цьому високого рівня незалежності його від інших. Всі зв’язки повинні бути логічними та

структурованими.

Мета створення програмної архітектури – написання коду, який легше підтримувати, шляхом створення можливостей для модульного тестування ПЗ.

Критеріями гарної архітектури є:

- ефективність;
- гнучкість: при внесеннях змін в систему вона не повинна зламатися або видати безліч помилок;
- розширюваність: можливість додавати нові елементи до ПЗ зі збереганням коректності його роботи;
- масштабованість;
- тестованість;
- зрозумілість.

На платформі Android існує декілька популярних архітектурних парадигм для проєктування програмного забезпечення. Для застосування будь-якої з них необхідно розробити модель і представлення програми, а також прошарок-з'єднувач для встановлення зв'язків між іншими елементами.

У контексті архітектури мобільних додатків, «модель» (model) являє собою компонент, що відповідає за управління даними та бізнес-логікою додатку. Вона зберігає дані, обробляє їх і забезпечує їх доступність для інших компонентів архітектури. Модель може містити в собі структури даних та логіку їх обробки, класи, функції.

«Представлення» (view) відповідає за візуалізацію даних користувачу. Воно охоплює інтерфейс додатку та інтерактивні елементи, які дозволяють взаємодіяти з ним. Представлення отримує дані з моделі та відображає їх відповідно до вимог дизайну і функціональності.

Модель та представлення взаємодіють через посередника, що забезпечує обмін даними та взаємодію між компонентами архітектури.

Таке розділення забезпечує модульність та гнучкість розробки додатків,

що значно спрощує та прискорює цей процес.

Архітектура MVC (Model-View-Controller) заснована на абстракції даних і їх представленні у складних програмах, де користувачеві надається великий обсяг інформації. Роль з'єднувача у цьому підході відіграє контролер (controller), який обробляє взаємодію користувача з додатком та керує потоком даних між моделлю та представленням. Проблемою цього шаблону є те, що контролер дуже тісно взаємодіє з представленням, що ускладнює модульне тестування та зменшує гнучкість додатку (при змінах у представленні необхідно переписувати контролер).

Іншим архітектурним рішенням є MVP (Model-View-Presenter). MVP насправді є розвиненим шаблоном MVC зі зміненими залежностями. У цій архітектурі за зв'язок між моделлю та представленням відповідає презентер (presenter), обробляючи вхідні події та оновлюючи дані відповідно до дій користувача. Однак, презентори, як і контролери, характеризуються схильністю до надмірного накопичення додаткової бізнес-логіки, що призводить до появи громіздких класів, з якими складно взаємодіяти.

Архітектура MVI (Model-View-Intent) розділяє додаток на три основні компоненти: модель, представлення та інтент (intent). Інтент представляє дії або події, які стимулюють зміни в стані додатку. Загалом, MVI було розроблено з урахуванням реактивного програмування. Цей підхід акцентується на однозначному потоці даних та унідірекційному потоці керування, через це моделі у MVI повинні бути незмінними.

Ще однією моделлю архітектури ПЗ є MVVM (Model-View-ViewModel). Вона також є сучасним шаблоном MVC, однак замість контролера в ній створюється з'єднувальний прошарок ViewModel. Однією з ключових особливостей MVVM є використання зв'язку даних (Data Binding) для автоматичного оновлення представлення при зміні даних у ViewModel. Це зменшує кількість коду, потрібного для оновлення інтерфейсу користувача, і

полегшує синхронізацію даних між моделлю і представленням.

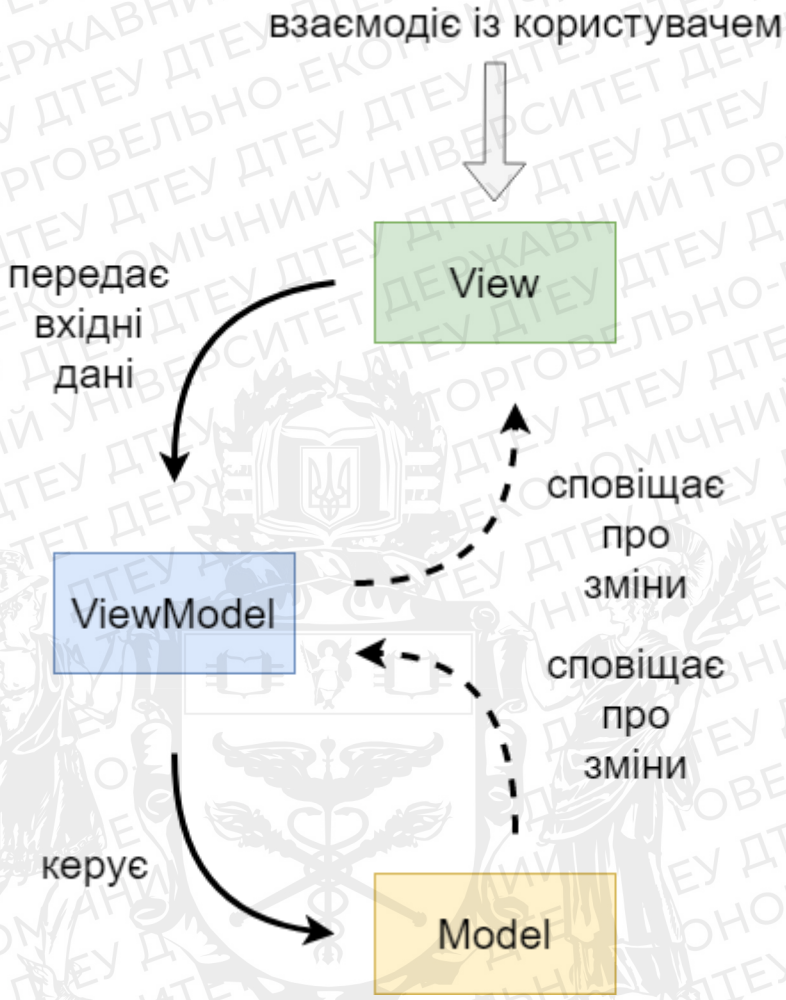

Рис. 2.5. Схема взаємодії компонентів моделі MVVM [10]

MVVM значно спрощує тестування і створює можливості для модульності цього процесу, оскільки характерне розділення компонентів у цій архітектурі дозволяє тестувати їх окремо. Ця особливість також дозволяє легко розширювати функціональність додатку, що робить програмне забезпечення більш гнучким.

Ще однією перевагою MVVM є можливість легкої обробки асинхронних операцій, такі як завантаження даних з мережі. Завдяки використанню асинхронних запитів і реактивного програмування, можна ефективно управляти завантаженням та оновленням даних без блокування інтерфейсу користувача.

Також шаблон MVVM підтримує використання Android Architecture Components, таких як LiveData і ViewModel, які спрощують роботу з життєвим

циклом програмного забезпечення, розробленого на платформі ОС Android.

Отже, вищеперелічені переваги використання архітектури MVVM для проєктування Android-додатку стали передумовами її обрання як шаблон розробки програмного проєкту даної кваліфікаційної роботи – мобільного застосунку «Coffee Proffy».

2.4. Висновок до розділу 2

Етап технічного проєктування є одним з найважливіших для створення якісного та ефективного програмного забезпечення, оскільки на ньому визначаються основні технічні аспекти та інструментарій розробки застосунку.

У ході проєктування Android-додатку «Coffee Proffy» сформовано детальний макет програмного інтерфейсу та обґрунтовано вибір стеку технологій для розробки, а саме: програмування застосунку буде здійснено мовою Kotlin у середовищі Android Studio з використанням архітектурної парадигми MVVM.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ НАТИВНОГО ПРОГРАМНОГО ДОДАТКУ ФУНКЦІОНУВАННЯ КАВ'ЯРНІ

3.1. Структура програмного проєкту

На наступному рисунку наведено схему розробленої бази даних для нативного Android-додатку функціонування кав'ярні «Coffee Proffy», на якій продемонстровані ключові зв'язки та загальна взаємодія між її елементами.

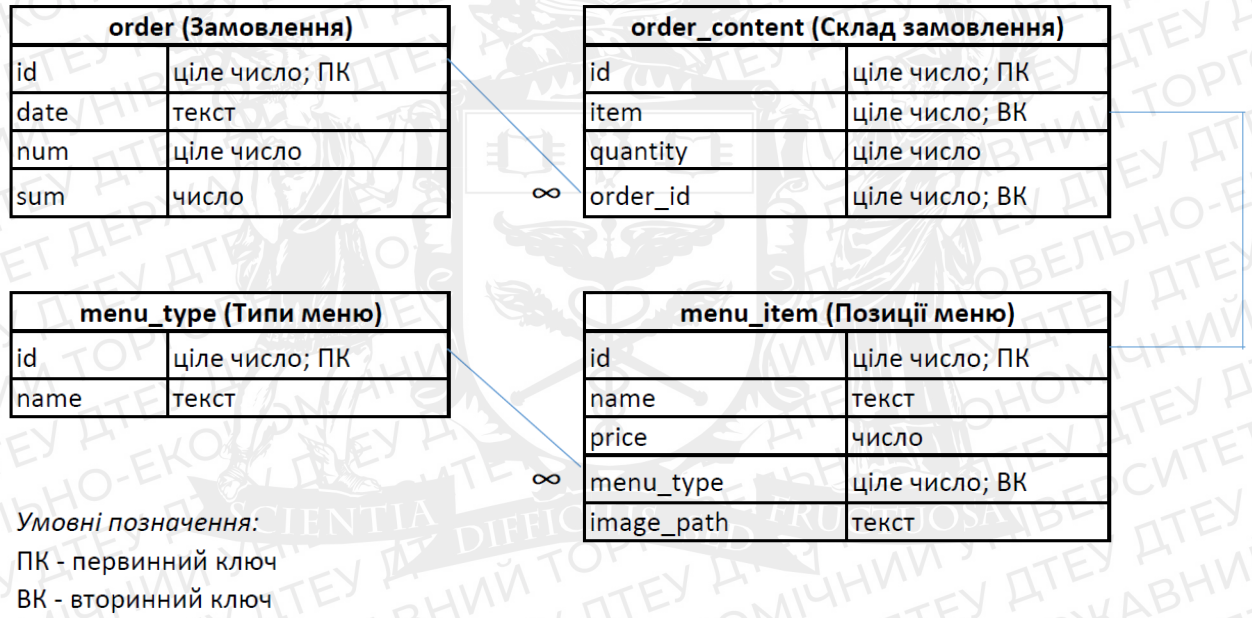

Рис. 3.1. Схема бази даних

Таблиця «Замовлення» містить основні відомості про кожне замовлення: ідентифікатор, дата замовлення, номер замовлення за день та сума замовлення.

Таблиця «Склад замовлення» описує вміст кожного замовлення, а саме: ідентифікатор, позиція меню, кількість позицій в асоціації з конкретним замовленням.

Таблиця «Типи меню» відповідає за різноманітність розділів меню та містить ідентифікатор і найменування розділу.

Таблиця «Позиції меню» призначена для зберігання усіх позицій меню, в асоціації з розділом меню, і містить наступні поля: ідентифікатор, найменування, ціна, шлях до зображення відповідної позиції, розділ меню.

У таблиці 3.1 представлено набір класів, у яких описано функціональну логіку розробленого Android-додатку «Coffee Proffy».

Таблиця 3.1.

Перелік програмних класів Android-додатку «Coffee Proffy»

Назва класу	Призначення класу
MainActivity (дод. А)	Клас головного екрану, у якому користувач спостерігає зміну фрагментів інтерфейсу в залежності від своїх дій
SplashFragment (дод. Б)	Клас фрагмента завантаження додатку
MainMenuFragment (дод. В)	Клас фрагмента головного меню
NewOrderFragment (дод. Г.1)	Клас фрагмента оформлення нового замовлення; також використовується для перегляду та редагування вже створеного замовлення
NewOrderViewModel (дод. Г.2)	Клас моделі представлення процесу оформлення замовлення
OrderAdapter (дод. Г.3)	Клас адаптера списку – складу замовлення
OrderInterface (дод. Г.4)	Інтерфейс взаємодії адаптера складу замовлення та фрагмента замовлення
OrderHistoryFragment (дод. Д.1)	Клас фрагмента журналу замовлень
OrderHistoryViewModel (дод. Д.2)	Клас моделі представлення журналу замовлень

Продовження табл. 3.1.

Назва класу	Призначення класу
OrderHistoryAdapter (дод. Д.3)	Клас адаптера списку – замовлень у журналі
OrderHistoryOnClickInterface (дод. Д.4)	Інтерфейс взаємодії адаптера замовлень з журналу та фрагмента журналу замовлень
AnalyticalFragment (дод. Е.1)	Клас фрагмента аналізу роботи кав'ярні за ключовими показниками
MenuItemAdapter (дод. Е.2)	Клас адаптера списку – топ-5 позицій меню за період
CatalogsFragment (дод. Ж.1)	Клас фрагмента розділів меню
CatalogListAdapter (дод. Ж.2)	Клас адаптера списку – розділів меню кав'ярні
CatalogInterface (дод. Ж.3)	Інтерфейс взаємодії адаптера розділів меню та фрагмента розділів меню
NewCatalogFragment (дод. И)	Клас фрагмента створення нового розділу меню; також використовується для редагування вже створеного розділу меню
CatalogFragment (дод. К.1)	Клас фрагмента перегляду позицій меню відповідного розділу
MenuItemListAdapter (дод. К.2)	Клас адаптера списку – позицій меню відповідного розділу меню кав'ярні
MenuItemInterface (дод. К.3)	Інтерфейс взаємодії адаптера позицій меню та фрагмента позицій меню
NewMenuItemFragment (дод. Л)	Клас фрагмента створення нової позиції меню
SettingsFragment (дод. М)	Клас фрагмента налаштувань додатку

Кінець таблиці 3.1.

Назва класу	Призначення класу
SettingsFragment (дод. М)	Клас фрагмента налаштувань додатку
RoomDatabase (дод. Н)	Абстрактний клас опису ініціалізації БД за використання Room
CoffeeProffyDao (дод. П)	Набір методів маніпулювання даними у БД шляхом запитів до таблиць
CoffeeProffyRepository (дод. Р)	Клас-посередник для взаємодії компонентів додатку із класом-маніпулятором БД
CoffeeOrder (дод. С.1)	Клас даних таблиці «Замовлення»
OrderContent (дод. С.2)	Клас даних таблиці «Склад замовлення»
MenuType (дод. С.3)	Клас даних таблиці «Розділ меню»
CoffeeMenuItem (дод. С.4)	Клас даних таблиці «Позиція меню»

3.2. Огляд розробленого додатку функціонування кав'ярні, його тестування та валідація

1. Реалізація інтерфейсу користувача

При запуску додатку «Coffee Proffy» перше, що бачить користувач – це вхідний проміжний екран завантажування. На ньому відображено назву додатку, яку можна редагувати в налаштуваннях.

Рис. 3.2. Вхідний екран завантажування

Після завантаження баз даних та початкових налаштувань, користувач потрапляє у головне меню зі списком усіх розділів додатку. Всього функціонал цього екрану складається з п’яти активних елементів, представлених у вигляді кнопок: «Нове замовлення», «Журнал замовлень», «Аналіз роботи кав’ярні», «Розділи меню» та «Налаштування». Вигляд зазначених розділів продемонстровано на рисунках 3.4 - 3.10.

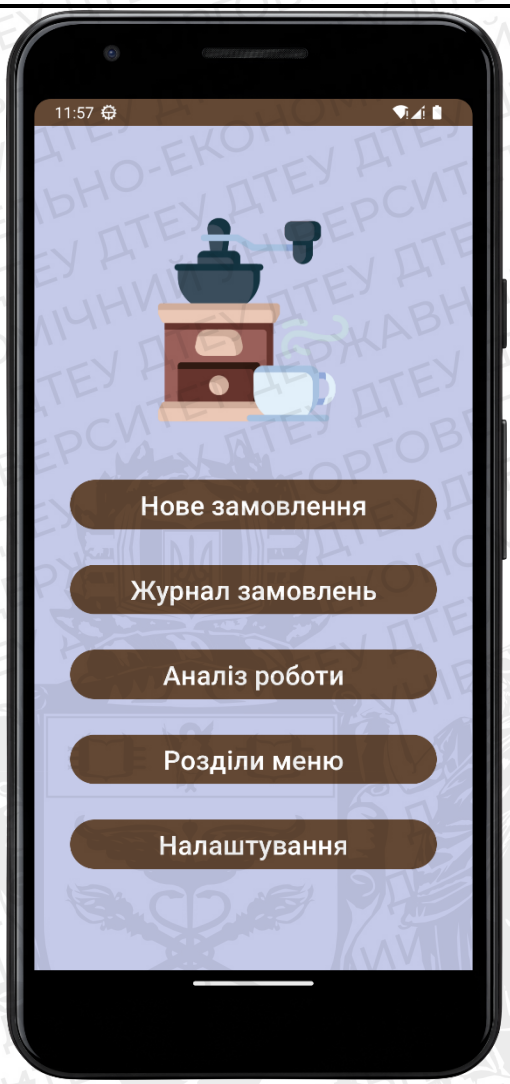

Рис. 3.3. Екран головного меню

Екран «Нове замовлення» розроблений на основі макету зі списком текстових полів, які описують наповнення замовлення. Додавання нового замовлення здійснюється шляхом натискання на інтерактивне зображення «+», після чого автоматично відкриватимуться розділи меню, представлені на рисунку 3.7, де можна буде обрати необхідні позиції та їх кількість. До нового замовлення буде автоматично додаватись поточна дата та обраховуватись його загальна вартість. Для закінчення формування замовлення та його подальшого зберігання у базі даних необхідно натиснути на кнопку «Готово» в нижній частині екрану.

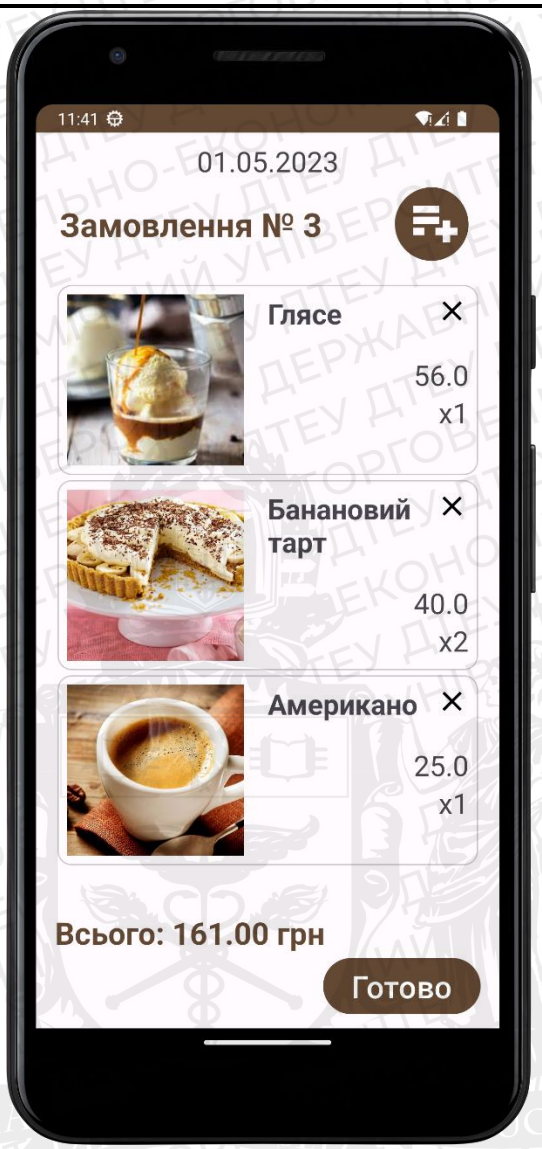

Рис. 3.4. Екран формування нового замовлення

Усі створені в додатку замовлення зберігатимуться у вищеописаній базі даних і відображатимуться на екрані «Журнал замовлень», де їх можна буде сортувати за датою. Цей екран, подібно до попереднього, представлено у вигляді текстового списку. За допомогою інтерактивних елементів можна видалити замовлення чи переглянути/редагувати його детальний склад. У другому зазначеному випадку буде здійснено перехід на другорядний екран «Деталі замовлення», аналогічний до рисунка 3.4.

Рис. 3.5. Екран «Журнал замовлень»

На екрані аналізу роботи кав’ярні автоматично відображатиметься список п’яти найпопулярніших позицій меню. Загалом, у цьому розділі користувач зможе ознайомитись з середньою та загальною кількістю та вартістю замовлень, а також статистикою замовлень по днях і годинах, що створює можливості для якісного аналізу актуальних показників продажів кав’ярні і проєктування плану розвитку бізнесу на основі отриманих даних.

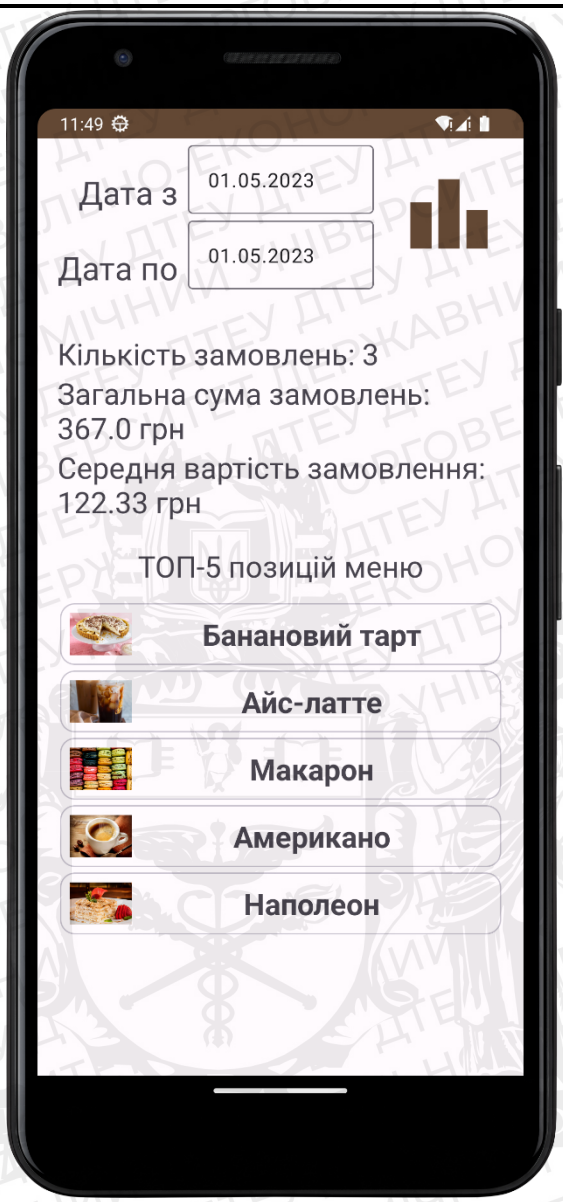

Рис. 3.6. Екран «Аналіз роботи кав'ярні»

Екран «Розділи меню» має вигляд списку інтерактивних кнопок, на кожній з яких вказана назва відповідного розділу меню. При натисканні на вказані елементи відкриватиметься екран перегляду усіх раніше доданих позицій меню, кожна з яких описуватиметься її назвою, ціною та фото.

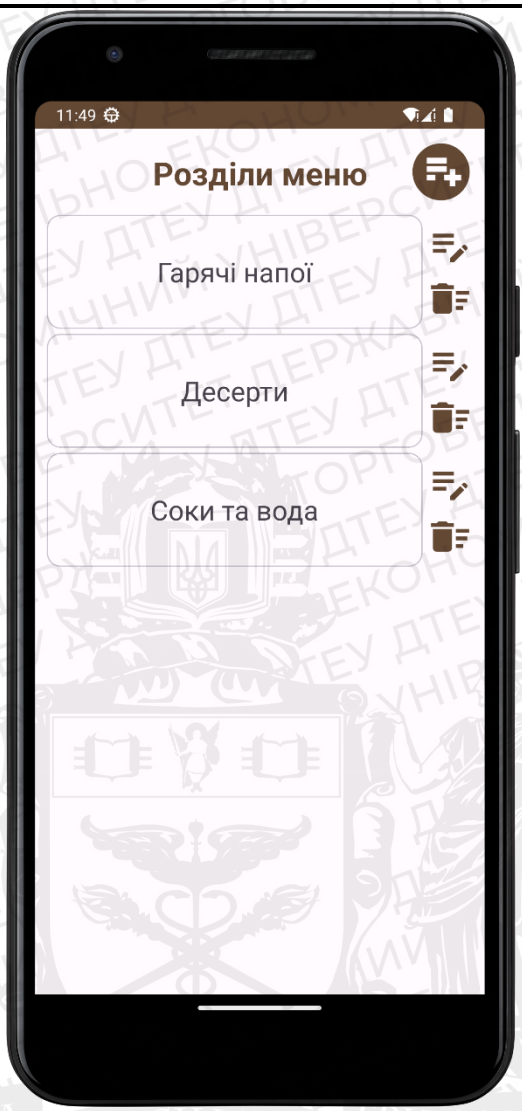

Рис. 3.7. Екран «Розділи меню»

При натисканні на зображення «+» у верхньому правому кутку вищезазначеного екрану, відкриватиметься другорядний екран для створення нового розділу меню, який міститиме текстове поле для введення та кнопку збереження.

Рис. 3.8. Другорядний екран створення нового розділу меню

Аналогічним чином, натискаючи на «+» під час перегляду списку позицій конкретного розділу меню, буде здійснено перехід на екран з двома текстовими полями введення назви нової позиції та її ціни, а також полем для завантаження фото даного товару та кнопкою збереження.

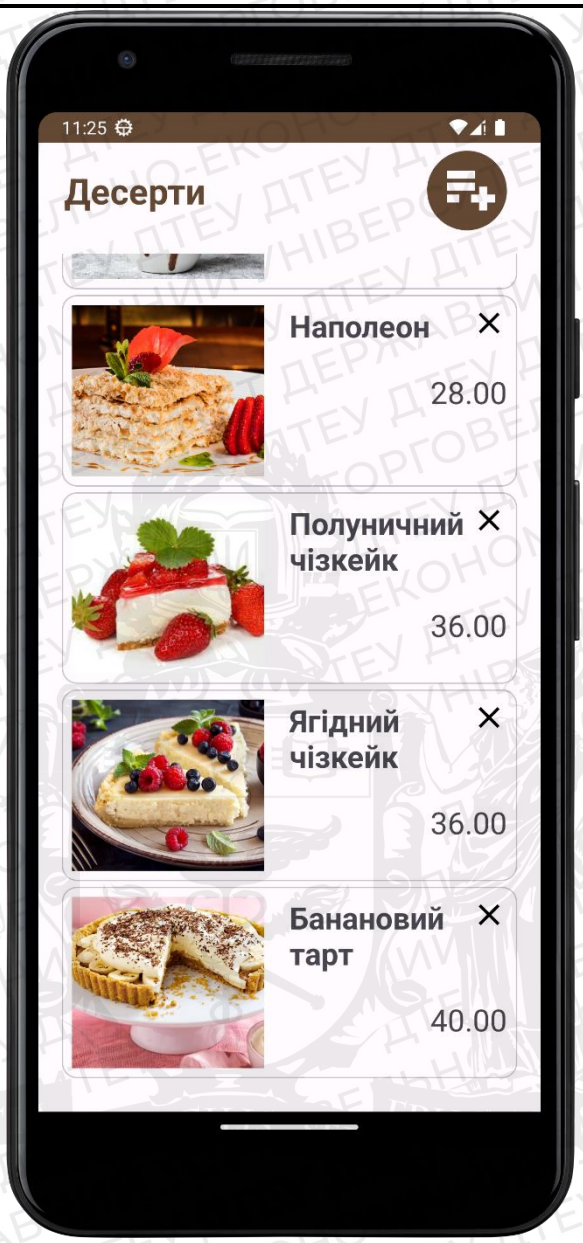

Рис. 3.9. Екран «Позиції меню»

У розділі «Налаштування» можна змінити назву кав’ярні, яка відображатиметься на вхідному екрані, а також встановити чи змінити пароль для входу у додаток і параметр відображення загальний підсумок замовлень на головному екрані. Вказаний функціонал реалізовуватиметься за допомогою використання активних текстових полів введення даних та інтерактивних перемикачів.

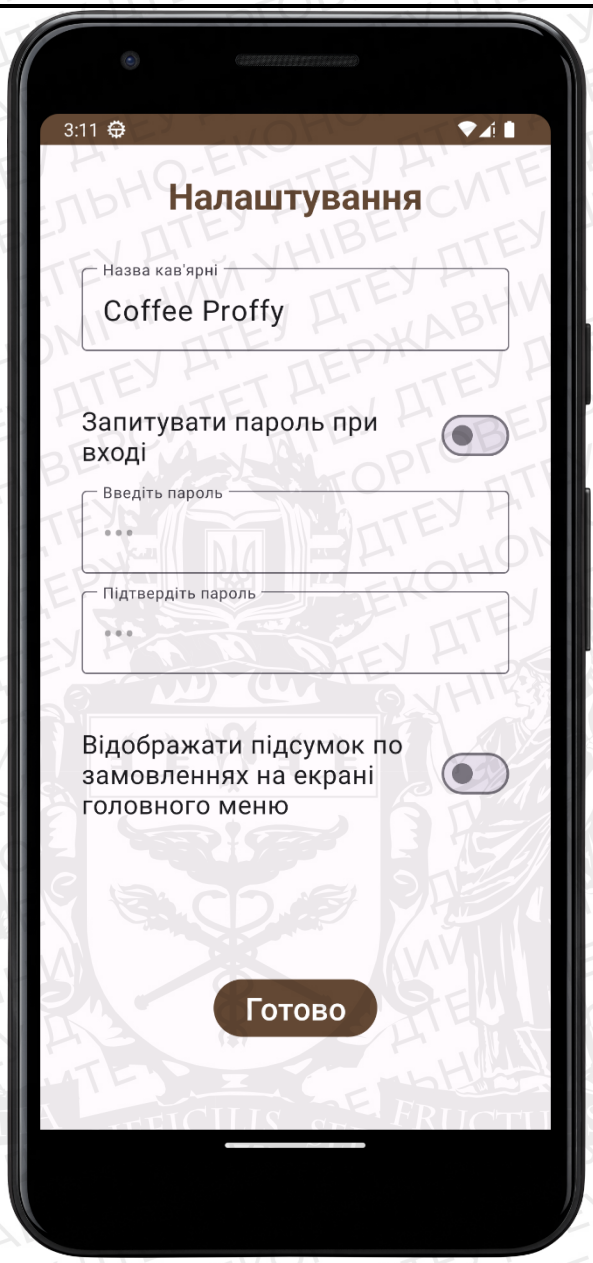

Рис. 3.10. Екран «Налаштування»

Отже, використовуючи новітній інструментарій розробки від Google, повністю сформовано й реалізовано графічний та функціональний інтерфейс користувача програмного додатку «Coffee Proffy».

2. Тестування та впровадження

Наступним етапом роботи після розробки нативного додатку є його тестування з метою підтвердження його високої якості для успішного впровадження і застосування для автоматизації бізнес-процесів замовника.

Тестування програмного додатку «Coffee Proffy» здійснювалося у декілька

етапів:

1) перевірка use-кейсів, сформованих на основі вимог ТЗ;
2) запуск на пристроях з різною версією Android;
3) тестування відображення екранів на пристроях з різною діагоналлю екрана.

В результаті тестування підтверджено стабільність та коректність роботи усього зазначеного у ТЗ функціоналу, після чого нативний додаток «Coffee Proffy» було впроваджено у роботу кав'ярні.

3.3. Висновок до розділу 3

Відповідно до функціональних та технічних вимог було виконано розробку нативного Android-додатку «Coffee Proffy». З цією метою досліджено, опрацьовано та систематизовано структури елементів інтерфейсу користувача, бази даних та програмні модулі; налагоджено навігацію та обмін даними між ними. У процесі розробки використано новітні технології, ресурси та інструменти. Безпосереднє написання програмного коду дозволило на практиці впевнитися у перевагах мови програмування Kotlin порівняно з Java.

Поєднання всіх цих аспектів дозволило створити ефективний та максимально оптимізований програмний продукт з лаконічним інтуїтивно зрозумілим інтерфейсом, якість якого підтверджено внаслідок перевірки як у реальному середовищі, так і за допомогою Unit-тестів. Отже, мобільний додаток «Coffee Proffy» повністю задовольняє поставлені функціональні завдання, вдало виконуючи автоматизацію обліку замовлень та продажів і надаючи максимально об'єктивний аналіз загальної роботи кав'ярні й окремих аспектів її діяльності.

З огляду на актуальні тенденції автоматизації роботи малого бізнесу, було сформовано рекомендований перелік розширень функціоналу застосунку «Coffee Proffy», які дозволять покращити його інтеграцію в бізнес-процеси функціонування кав'ярні.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Метою даної кваліфікаційної роботи була розробка програмного забезпечення для автоматизації функціонування кав'ярні, а саме обліку замовлень та продажів для подальшого аналізу ефективності роботи закладу на основі отриманих даних. В ході дослідження особливостей бізнес-процесів і функціональних аспектів роботи малого бізнесу в Україні, зокрема кав'ярень, було доведено актуальність потреби розроблення програмного продукту з вищеописаним функціоналом.

Проведено аналіз предметної області, а також огляд і порівняння систем з автоматизації з аналогічними функціональними можливостями, в результаті чого було прийнято рішення про створення нативного мобільного додатку, який забезпечуватиме максимальну доступність реалізації.

Було визначено ключові функціональні вимоги для програмного проєкту, на основі яких сформовано детальне технічне завдання, яке стало основою для створення якісного програмного продукту з інтуїтивно зрозумілим інтерфейсом.

Детальне проєктування макетів програмного інтерфейсу й систематизація його структурних елементів, бази даних та інших програмних модулів, а також обрання сучасних технічних інструментів розробки, а саме мови програмування Kotlin, IDE Android Studio з використанням системи збірки Gradle на основі архітектурного шаблону MVVM та об'єктно-орієнтованого підходу забезпечило комфортний, ефективний та швидкий процес створення ПЗ.

У процесі розробки мною набуті практичні навички сучасної Android-розробки, проєктування та реалізації архітектурних шаблонів програмного забезпечення та інтерфейсу користувача, використання інструментарію розробки, які були успішно застосовані у процесі виконання випускної

кваліфікаційної роботи.

Отже, в процесі виконання проєкту, відповідно до поставленого технічного завдання, розроблено нативний Android-додаток «Coffee Proffy», який автоматизує процеси функціонування кав'ярні й забезпечує зручну аналітику й прийняття обґрунтованих бізнес-рішень. Дана розробка повністю задовольняє усі вимоги та стандарти до ПЗ, а також поставлені функціональні завдання. «Coffee Proffy» – це оптимізований, гнучкий та ефективний програмний продукт зі зручним інтуїтивно зрозумілим інтерфейсом, якість якого протестована і підтверджена в результаті практичного використання.

Як один із перспективних напрямків розвитку додатку «Coffee Proffy», пропонується підключення до сервісу Firebase з метою впровадження хмарного збереження БД та авторизації для різних співробітників кав’ярні, а також розширення набору таблиць БД у разі використання додатку мережею кав’ярень для розбиття облікових даних по філіям.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android [Електронний ресурс]. – Режим доступу: https://www.android.com/
2. Android Developers [Електронний ресурс]. – Режим доступу: https://developer.android.com
3. Android Statistics (2023) [Електронний ресурс]. – Режим доступу: https://www.businessofapps.com/data/android-statistics/
4. Android Studio [Електронний ресурс]. – Режим доступу: https://developer.android.com/studio
5. Android Studio: переваги та особливості [Електронний ресурс]. – Режим доступу: https://qagroup.com.ua/publications/android-studio-perevagy-ta-osoblyvosti/
6. Android vs iOS: яку платформу обрати для вашого бізнесу [Електронний ресурс]. – Режим доступу: https://itechua.com/articles/211148
7. Kotlin [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/
8. Kotlin vs Java: що краще для Android-розробки? [Електронний ресурс]. – Режим доступу: https://lampalampa.net/ua/category/articles_ua/
9. Mobile Operating System Market Share Ukraine [Електронний ресурс]. – Режим доступу: https://gs.statcounter.com/os-market-share/mobile/ukraine/#monthly-202001-202212
10. Model-View-ViewModel [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Model-View-ViewModel
11. MVC, MVP, MVI, MVVM and VIPER Design Patterns [Електронний ресурс]. – Режим доступу: https://medium.com/@pinarkocak/mvc-mvp-and-mvvm-design-patterns-82317d6efeac

					ДТЕУ 121-07-16.БР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко		23.12.22	Нативний програмний додаток функціонування кав'ярні	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Гнатченко Д.Д.		23.12.22		*СВД*	48	50
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій 4 курс, 7 група*		
Розробив		Лук'янець Б.Б.		23.12.22	*Список використаних джерел*			

12. Phillips B., Marsicano K., Stewart Ch. Android Programming: The Big Nerd Ranch Guide / B. Phillips, K. Marsicano, Ch. Stewart – Atlanta : Big Nerd Ranch, Inc., 2017. – 688 p.
13. Автоматизація кав'ярні Poster [Електронний ресурс]. – Режим доступу: https://joinposter.com/ua/business/coffeeshop
14. Інформаційні системи [Електронний ресурс]. – Режим доступу: https://pidru4niki.com/1222090547713/informatika/informatsiyni_sistemi
15. Никифоров Д. В., Черних О. П. Порівняння архітектур з використанням патернів MVC, MVP і MVVM при розробці Android додатку / Д. В. Никифоров, О. П. Черних // Інформаційні технології: наука, техніка, технологія, освіта, здоров'я. – Харків : Нац. техн. ун-т «ХПУ», 2017. – Вип. 4. – С. 101.
16. Переваги та недоліки кросплатформної та нативної розробки мобільних додатків [Електронний ресурс]. – Режим доступу: https://merehead.com/ua/blog/cross-platform-native-mobile-development/
17. Програма для автоматизації кав'ярні PalmaBox Horeca [Електронний ресурс]. – Режим доступу: https://horeca.palmabox.com/avtomatizatsiya-kavyarni/?gclid=CjwKCAjw6IiiBhAOEiwALNqncfBCiciBgcX49ro2ZXOu3re2lBdyNqRaMMP8NCMSiJGrNWyC6Mt4tBoCUCgQAvD_BwE
18. Проектування мобільного додатка [Електронний ресурс]. – Режим доступу: https://wezom.com.ua/ua/blog/proektirovanie-mobilnogo-prilozheniya
19. Рейтинг мов програмування 2022 [Електронний ресурс]. – Режим доступу: https://dou.ua/lenta/articles/language-rating-2022/
20. Савченко Д. М., Лещинський В. О. Дослідження методів оптимізації бізнес-процесів кав'ярні / Д. М. Савченко, В. О. Лещинський // Збірник статей учасників двадцять п'ятої всеукраїнської практично-пізнавальної інтернет-конференції «Наукова думка сучасності і майбутнього». – Дніпро : Видавництво НМ, 2019. – С. 6-9. – Режим доступу:

https://naukam.triada.in.ua/index.php/konferentsiji/55-dvadtsyat-p-yata-vseukrajinska-praktichno-piznavalna-internet-konferentsiya/645-doslidzhennya-metodiv-optimizatsiji-biznes-protsesiv-kav-yarni

21. Сеник І. О. Розробка автоматизованої системи управління для кав'ярні : квал. роб. на здобуття ступеня бакалавра / І. О. Сеник – Вінниця : Донецьк. нац. ун-т ім. Василя Стуса, 2021.
22. Типи мобільних додатків [Електронний ресурс]. – Режим доступу: https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications/
23. Ткачук В. О., Шестакова А. В. Використання мобільних додатків для цифровізації бізнес-процесів у ресторанному бізнесі та їх оптимізація на основі методу A/B тестування / В. О. Ткачук, А. В. Шестакова // Економіка, управління та адміністрування. Серія: Економіка. – 2022. – Вип. 4(102). – С. 28-34.
24. Чалий С. Ф. Автоматизоване управління бізнес-процесами (моделі, методи і технології) : дис. на здобуття наук. ступеня д-ра техн. наук / С. Ф. Чалий – Харків : Харків. нац. ун-т радіоелектроніки, 2007. – 502 с.
25. Як пришвидшити процеси в кав'ярні чи магазині: вибір системи автоматизації та перші кроки [Електронний ресурс]. – Режим доступу: https://evo.business/yak-prishvidshiti-procesi-v-kavyarni-chi-magazini-vibir-sistemi-avtomatizaci%D1%97-ta-pershi-kroki/
26. Яшина О. М., Гремечевський Р. В., Братасюк Д. І. Порівняльна характеристика актуальних засобів розробки під мобільні платформи / О. М. Яшина, Р. В. Гремечевський, Д. І. Братасюк / відп. ред. Р.В. Сорокатий – Хмельницьк : ХНУ, 2018 – 7 с.

ДОДАТКИ

ДОДАТОК А

MainActivity.kt

```
class MainActivity : AppCompatActivity() {
  lateinit var repository: CoffeeProffyRepository

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    repository = CoffeeProffyRepository(this)
  }
}
```

ДОДАТОК Б

SplashFragment.kt

```
class SplashFragment : Fragment() {
    lateinit var binding: SplashBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = SplashBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val preferences = activity?.getPreferences(Context.MODE_PRIVATE)
        val passwordOn = preferences?.getBoolean("PASSWORD_ON", false)
        val caffeeName = preferences?.getString("CAFFEE_NAME", "Coffee Proffy")
        binding.appName.text = caffeeName

        if (passwordOn == true) {
            binding.passLayout.visibility = View.VISIBLE
            val password = preferences.getString("PASSWORD", "")
            binding.password.setOnKeyListener { v, keyCode, event ->
                if (keyCode == EditorInfo.IME_ACTION_DONE
                    || event.action == KeyEvent.ACTION_DOWN
                    || event.action == KeyEvent.KEYCODE_ENTER
                ) {
                    if (binding.password.text.toString() == password) {

findNavController().navigate(R.id.action_splashFragment_to_mainMenuFragment)
                    } else {
                        binding.password.error = "Невірний пароль"
                    }
                }
                return@setOnKeyListener true
            }
        } else {
            binding.password.visibility = View.GONE
            CoroutineScope(Dispatchers.Main).launch {
                delay(2000)

findNavController().navigate(R.id.action_splashFragment_to_mainMenuFragment)
            }
        }

    }
}
```

ДОДАТОК В

MainMenuFragment.kt

```
class MainMenuFragment : Fragment() {
    lateinit var binding: MainMenuBinding
    lateinit var repository: CoffeeProffyRepository
    private val callback = object : OnBackPressedCallback(true) {
        override fun handleOnBackPressed() {
            activity?.finish()
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = MainMenuBinding.inflate(inflater, container, false)
        return binding.root
    }

    @SuppressLint("SimpleDateFormat")
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        requireActivity().onBackPressedDispatcher.addCallback(viewLifecycleOwner,
callback)

        val preferences = activity?.getPreferences(Context.MODE_PRIVATE)
        val resultOn = preferences?.getBoolean("ORDER_INFO", false)
        if (resultOn == true) {
            binding.header.visibility = View.VISIBLE
            repository = (activity as MainActivity).repository

            val dbFormat = SimpleDateFormat("yyyy-MM-dd")
            val currentDate = dbFormat.format(Date())
```

```
                CoroutineScope(Dispatchers.IO).launch {
                    val orders = repository.getAllCoffeeOrders(currentDate, currentDate)
                    withContext(Dispatchers.Main) {
                        binding.header.text = "За сьогодні ${orders.size} замовлень \nна суму
${calcSum(orders)} грн."
                    }
                }
            } else {
                binding.header.visibility = View.GONE
            }

            with(binding) {
                newOrder.setOnClickListener {

findNavController().navigate(R.id.action_mainMenuFragment_to_newOrderFragment)
                }
                orderHistory.setOnClickListener {

findNavController().navigate(R.id.action_mainMenuFragment_to_orderHistoryFragment)
                }
                analysis.setOnClickListener {

findNavController().navigate(R.id.action_mainMenuFragment_to_analyticalFragment)
                }
                catalogs.setOnClickListener {

findNavController().navigate(R.id.action_mainMenuFragment_to_catalogsFragment)
                }
                settings.setOnClickListener {

findNavController().navigate(R.id.action_mainMenuFragment_to_settingsFragment)
                }
            }
        }
```

Продовження дод. В

```
    private fun calcSum(orders: List<CoffeeOrder>): Float {
        var sum = 0f
        for (order in orders) {
            sum += order.sum
        }
        return sum
    }
}
```

ДОДАТОК Г.1

NewOrderFragment.kt

```
class NewOrderFragment : Fragment(), OrderInterface {
    lateinit var binding: NewOrderBinding
    lateinit var adapter: OrderAdapter
    lateinit var repository: CoffeeProffyRepository
    private var orderToEdit: CoffeeOrder? = null
    private val format = SimpleDateFormat("dd.MM.yyyy")
    private val dbFormat = SimpleDateFormat("yyyy-MM-dd")

    private val orderViewModel: NewOrderViewModel by activityViewModels()
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = NewOrderBinding.inflate(inflater, container, false)
        return binding.root
    }

    @SuppressLint("NotifyDataSetChanged")
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        repository = (activity as MainActivity).repository

        arguments?.let {
            orderToEdit = arguments?.getSerializable("ORDER") as CoffeeOrder?
        }

        val orderIsChanging = arguments?.getBoolean("ORDER_CHANGING") ?: false

        val orderDate = setCurrentDateAndGetFormatted()

        if (orderToEdit != null) {
```

```
            CoroutineScope(Dispatchers.IO).launch {
                val orderContents =
                    contentListToMap(repository.getAllOrderContents(orderToEdit!!.id))
                Log.d("NewOrderFragment", "orderContents: $orderContents")
                withContext(Dispatchers.Main) {
                    if (!orderIsChanging)
                        orderViewModel.orderContents.value = orderContents

                    adapter = OrderAdapter(requireContext(),
orderViewModel.orderContents.value!!, repository)
                    adapter.setCallback(this@NewOrderFragment)
                    binding.menuItems.adapter = adapter
                }
            }
            if (!orderIsChanging)
                orderViewModel.orderSum.value = orderToEdit!!.sum
            binding.title.text = activity?.resources?.getString(
                R.string.order, orderToEdit!!.num
            )
            binding.date.text = format.format(dbFormat.parse(orderToEdit!!.date))
        } else {
            setNewOrderNumber(orderDate)

            adapter = OrderAdapter(requireContext(), orderViewModel.orderContents.value!!,
repository)
            adapter.setCallback(this@NewOrderFragment)
            binding.menuItems.adapter = adapter
        }

        binding.sum.text =
            activity?.resources?.getString(
                R.string.sum,
                convertFloatToString(orderViewModel.orderSum.value!!)
            )
```

```
        orderViewModel.orderSum.observe(viewLifecycleOwner) {
            binding.sum.text =
                activity?.resources?.getString(R.string.sum, convertFloatToString(it))
            binding.menuItems.adapter?.notifyDataSetChanged()
        }

        binding.addItem.setOnClickListener {
            val bundle = Bundle()
            bundle.putBoolean("NEW_ORDER", true)
            bundle.putSerializable("ORDER", orderToEdit)
            findNavController().navigate(R.id.action_newOrderFragment_to_catalogsFragment,
bundle)
        }

        binding.done.setOnClickListener {
            if (orderViewModel.orderContents.value?.size == 0) {
                return@setOnClickListener
            }
            CoroutineScope(Dispatchers.IO).launch {
                val lastOrder = repository.getLastOrderOfTheDay(orderDate)
                val orderNum = orderToEdit?.num ?: (lastOrder?.num?.plus(1) ?: 1)
                val order = CoffeeOrder(
                    date = orderDate,
                    num = orderNum,
                    sum = orderViewModel.orderSum.value!!
                )
                if (orderToEdit != null) {
                    val oldContents = repository.getAllOrderContents(orderToEdit!!)
                    for (oldContent in oldContents) {
                        repository.deleteOrderContent(oldContent)
                        Log.d("NewOrderFragment", "orderToEdit!!.sum: ${orderToEdit!!.sum}")
                        Log.d(
                            "NewOrderFragment",
```

```
                    "orderViewModel.orderSum.value!!:
${orderViewModel.orderSum.value!!}"
                )
                orderToEdit!!.sum = orderViewModel.orderSum.value!!
                Log.d("NewOrderFragment", "orderToEdit!!.sum: ${orderToEdit!!.sum}")
                Log.d(
                  "NewOrderFragment",
                  "orderViewModel.orderSum.value!!:
${orderViewModel.orderSum.value!!}"
                )

                repository.updateCoffeeOrder(orderToEdit!!)
              }
            } else {
              repository.insertCoffeeOrder(order)
            }
            val orderId =
              orderToEdit?.id ?: (repository.getLastOrderOfTheDay(orderDate)?.id ?: 0)
            orderViewModel.orderContents.value!!.forEach {
              repository.insertOrderContent(
                OrderContent(
                  item = it.key.id,
                  quantity = it.value,
                  orderId = orderId
                )
              )
            }

            withContext(Dispatchers.Main) {
              orderViewModel.clearOrder()

findNavController().navigate(R.id.action_newOrderFragment_to_mainMenuFragment)
            }
          }
```

```
    }
}

private fun setNewOrderNumber(orderDate: String) {
    CoroutineScope(Dispatchers.IO).launch {
        val lastOrder = repository.getLastOrderOfTheDay(orderDate)
        lastOrder?.let {
            withContext(Dispatchers.Main) {
                binding.title.text = activity?.resources?.getString(
                    R.string.order, lastOrder.num + 1
                )
            }
            return@launch
        }
        if (lastOrder == null) {
            withContext(Dispatchers.Main) {
                binding.title.text = activity?.resources?.getString(
                    R.string.order, 1
                )
            }
        }
    }

}

@SuppressLint("SimpleDateFormat")
private fun setCurrentDateAndGetFormatted(): String {
    val currentDate = Date()
    val format = SimpleDateFormat("dd.MM.yyyy")
    val formattedDate = format.format(currentDate)
    binding.date.text = formattedDate

    val dbFormat = SimpleDateFormat("yyyy-MM-dd")
    return dbFormat.format(currentDate)
```

```
        }

        private fun convertFloatToString(value: Float): String {
            val decimalFormat = DecimalFormat("0.00")
            return decimalFormat.format(value)
        }

        override fun onDeleteClick(item: CoffeeMenuItem) {
            orderViewModel.removeItemFromOrder(item)
        }

        private suspend fun contentListToMap(list: List<OrderContent>):
MutableMap<CoffeeMenuItem, Int> {
            val map = mutableMapOf<CoffeeMenuItem, Int>()
            list.forEach {
                val item = repository.getMenuItemById(it.item)
                map[item!!] = it.quantity
            }
            return map
        }
    }
```

ДОДАТОК Г.2

NewOrderViewModel.kt

```
class NewOrderViewModel : ViewModel() {
    fun addItemToOrder(menuItem: CoffeeMenuItem) {
        if (orderContents.value!!.containsKey(menuItem)) {
            orderContents.value!![menuItem] = orderContents.value!![menuItem]!! + 1
            calcSum()
            return
        }
        orderContents.value!![menuItem] = 1
        calcSum()
    }

    fun removeItemFromOrder(menuItem: CoffeeMenuItem) {
        orderContents.value!!.remove(menuItem)
        calcSum()
    }

    private fun calcSum() {
        Log.d("VIEWMODEL", "before calcSum: ${orderSum.value}")
        var sum = 0f
        orderContents.value!!.forEach {
            sum += it.key.price * it.value
        }
        orderSum.value = sum
        Log.d("VIEWMODEL", "calcSum: ${orderSum.value}")
        Log.d("VIEWMODEL", "orderContents: ${orderContents.value}")
    }

    val orderContents: MutableLiveData<MutableMap<CoffeeMenuItem, Int>> =
MutableLiveData()
    val orderSum: MutableLiveData<Float> = MutableLiveData()

    init {
        orderContents.value = mutableMapOf()
```

Продовження дод. Г.2

```
        orderSum.value = 0f
    }

    fun clearOrder() {
        orderContents.value?.clear()
        orderSum.value = 0f
    }
}
```

ДОДАТОК Г.3

OrderAdapter.kt

```
class OrderAdapter(val context: Context,
    private val orderContents: MutableMap<CoffeeMenuItem, Int>,
    private val repository: CoffeeProffyRepository
) : RecyclerView.Adapter<OrderAdapter.OrderViewHolder>() {
    lateinit var binding: OrderItemBinding
    lateinit var callbackInterface: OrderInterface

    private val itemList = orderContents.keys.toList()
    private val itemQuan = orderContents.values.toList()

    inner class OrderViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): OrderViewHolder
{
      binding = OrderItemBinding.inflate(LayoutInflater.from(parent.context), parent, false)
      return OrderViewHolder(binding.root)
    }

    override fun getItemCount(): Int {
      return orderContents.size
    }

    @SuppressLint("SetTextI18n")
    override fun onBindViewHolder(holder: OrderViewHolder, position: Int) {
      with(binding) {
        val orderItem = itemList[position]
        Log.d("OrderAdapter", "orderItem: $orderItem")
        val orderItemQuan = itemQuan[position]
        Log.d("OrderAdapter", "orderItemQuan: $orderItemQuan")
        itemQuantity.text = "x$orderItemQuan"

        Glide.with(context)
          .load(orderItem.imagePath)
```

Продовження дод. Г.3

```
            .into(photo)

        deleteItem.setOnClickListener {
            callbackInterface.onDeleteClick(orderItem)
        }

        CoroutineScope(Dispatchers.IO).launch {
            val item = repository.getMenuItemById(orderItem.id)
            withContext(Dispatchers.Main) {
                item?.let {
                    itemTitle.text = it.name
                    itemPrice.text = it.price.toString()
                }
            }
        }
    }

    fun setCallback(callbackOnClick: OrderInterface) {
        callbackInterface = callbackOnClick
    }
}
```

ДОДАТОК Г.4

OrderInterface.kt

```
interface OrderInterface {
  fun onDeleteClick(item: CoffeeMenuItem)
}
```

ДОДАТОК Д.1

OrderHistoryFragment.kt

```
class OrderHistoryFragment : Fragment(), OrderHistoryOnClickInterface {
    lateinit var binding: OrderHistoryBinding
    lateinit var adapter: OrderHistoryAdapter
    lateinit var repository: CoffeeProffyRepository
    val orderHistoryViewModel: OrderHistoryViewModel by activityViewModels()

    @SuppressLint("SimpleDateFormat")
    private val format = SimpleDateFormat("dd.MM.yyyy")
    private val currentDate = Date()
    private val formattedDate = format.format(currentDate)

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = OrderHistoryBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        repository = (activity as MainActivity).repository

        setDatesAsToday()
        loadHistory()

        setListeners()
    }

    private fun setListeners() {
        binding.dateFromInput.setOnClickListener {
```

```
            val calendar = Calendar.getInstance()
            val year = calendar.get(Calendar.YEAR)
            val month = calendar.get(Calendar.MONTH)
            val day = calendar.get(Calendar.DAY_OF_MONTH)

            val datePickerDialog = DatePickerDialog(
              requireContext(),
              DatePickerDialog.OnDateSetListener { _, selectedYear, selectedMonth,
selectedDay ->
                val selectedDayString =
                  if (selectedDay < 10) "0$selectedDay" else "$selectedDay"
                val selectedMonthString =
                  if (selectedMonth < 9) "0${selectedMonth + 1}" else "${selectedMonth + 1}"
                val formattedDate = "$selectedDayString.$selectedMonthString.$selectedYear"
                binding.dateFromInput.setText(formattedDate)
                checkDates()
              },
              year,
              month,
              day
            )

            datePickerDialog.show()
          }

          binding.dateToInput.setOnClickListener {
            val calendar = Calendar.getInstance()
            val year = calendar.get(Calendar.YEAR)
            val month = calendar.get(Calendar.MONTH)
            val day = calendar.get(Calendar.DAY_OF_MONTH)
            val datePickerDialog = DatePickerDialog(
              requireContext(),
              DatePickerDialog.OnDateSetListener { _, selectedYear, selectedMonth,
selectedDay ->
```

```
                val selectedDayString =
                    if (selectedDay < 10) "0$selectedDay" else "$selectedDay"
                val selectedMonthString =
                    if (selectedMonth < 9) "0${selectedMonth + 1}" else "${selectedMonth + 1}"
                val formattedDate = "$selectedDayString.$selectedMonthString.$selectedYear"
                binding.dateToInput.setText(formattedDate)
                checkDates()
            },
            year,
            month,
            day
        )

        datePickerDialog.show()
    }
}

private fun checkDates() {
    val dateFrom = format.parse(binding.dateFromInput.text.toString())
    val dateTo = format.parse(binding.dateToInput.text.toString())
    if (dateFrom != null && dateTo != null) {
        if (dateFrom.after(dateTo)) {
            binding.dateToInput.error = "Дата закінчення не може бути раніше дати початку"
        } else {
            binding.dateFromInput.error = null
            binding.dateToInput.error = null
            loadHistory()
        }
    }
}
private fun loadHistory() {
    orderHistoryViewModel.dateFrom = format.parse(binding.dateFromInput.text.toString()) as Date
```

```
            orderHistoryViewModel.dateTo = format.parse(binding.dateToInput.text.toString()) as
Date

            orderHistoryViewModel.loadHistory(repository)
            orderHistoryViewModel.orderHistoryLiveData.observe(viewLifecycleOwner) {
                adapter = OrderHistoryAdapter(it)
                adapter.setCallbackInterface(this)
                binding.orderList.adapter = adapter
            }
        }

        @SuppressLint("SimpleDateFormat")
        private fun setDatesAsToday() {
            binding.dateFromInput.setText(formattedDate)
            binding.dateToInput.setText(formattedDate)
        }

        override fun editOrder(order: CoffeeOrder) {
            val bundle = Bundle()
            bundle.putSerializable("ORDER", order)
            findNavController().navigate(

com.diploma.coffeeproffy.R.id.action_orderHistoryFragment_to_newOrderFragment,
                bundle
            )
        }

        override fun deleteOrder(order: CoffeeOrder) {
            val dialogBuilder = AlertDialog.Builder(requireContext())

            dialogBuilder.setMessage("Замовлення буде повністю видалено. Ви впевнені?")
                .setCancelable(false)
                .setPositiveButton("Так") { dialog, _ ->
                    CoroutineScope(Dispatchers.IO).launch {
```

Продовження дод. Д.1

```
                orderHistoryViewModel.deleteOrder(order, repository)
            }
            dialog.dismiss()
        }
        .setNegativeButton("Відмінити") { dialog, _ ->
            dialog.dismiss()
        }

    val dialog = dialogBuilder.create()
    dialog.show()
  }
}
```

ДОДАТОК Д.2

OrderHistoryViewModel.kt

```
class OrderHistoryViewModel : ViewModel() {
    val orderHistoryLiveData: MutableLiveData<List<CoffeeOrder>> = MutableLiveData()
    lateinit var dateFrom: Date
    lateinit var dateTo: Date

    @SuppressLint("SimpleDateFormat")
    val dbFormat = SimpleDateFormat("yyyy-MM-dd")

    init {
        orderHistoryLiveData.value = listOf()
    }

    fun loadHistory(repository: CoffeeProffyRepository) {
        CoroutineScope(Dispatchers.IO).launch {
            val orders = repository.getAllCoffeeOrders(
                dbFormat.format(dateFrom),
                dbFormat.format(dateTo)
            )
            withContext(Dispatchers.Main) {
                orderHistoryLiveData.value = orders
            }
        }
    }

    fun deleteOrder(order: CoffeeOrder, repository: CoffeeProffyRepository) {
        CoroutineScope(Dispatchers.IO).launch {
            repository.deleteCoffeeOrder(order)
            loadHistory(repository)
        }
    }
}
```

ДОДАТОК Д.3

OrderHistoryAdapter.kt

```
class OrderHistoryAdapter(private val orderHistoryList: List<CoffeeOrder>) :
    RecyclerView.Adapter<OrderHistoryAdapter.HistoryViewHolder>() {
    lateinit var binding: OrderListItemBinding
    lateinit var callbackHistoryInterface: OrderHistoryOnClickInterface

    inner class HistoryViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)
{

    }

    override fun onCreateViewHolder(
      parent: ViewGroup,
      viewType: Int
    ): OrderHistoryAdapter.HistoryViewHolder {
      binding = OrderListItemBinding.inflate(LayoutInflater.from(parent.context), parent,
false)
      return HistoryViewHolder(binding.root)
    }

    override fun onBindViewHolder(holder: OrderHistoryAdapter.HistoryViewHolder,
position: Int) {
      val order = orderHistoryList[position]
      binding.orderNumber.text = order.num.toString()
      binding.orderSum.text = convertFloatToString(order.sum)

      binding.editOrder.setOnClickListener {
        callbackHistoryInterface.editOrder(order)
      }
      binding.deleteOrder.setOnClickListener {
        callbackHistoryInterface.deleteOrder(order)
      }
    }
```

```
    override fun getItemCount(): Int {
        return orderHistoryList.size
    }

    private fun convertFloatToString(value: Float): String {
        val decimalFormat = DecimalFormat("0.00")
        return decimalFormat.format(value)
    }

    fun setCallbackInterface(onClickInterface: OrderHistoryOnClickInterface) {
        callbackHistoryInterface = onClickInterface
    }
}
```

ДОДАТОК Д.4

OrderHistoryOnClickInterface.kt

```
interface OrderHistoryOnClickInterface {
  fun editOrder(order: CoffeeOrder)
  fun deleteOrder(order: CoffeeOrder)
}
```

ДОДАТОК Е.1

AnalyticalFragment.kt

```
class AnalyticalFragment : Fragment() {
    lateinit var binding: AnalyticsBinding
    lateinit var repository: CoffeeProffyRepository
    lateinit var adapter: MenuItemAdapter

    @SuppressLint("SimpleDateFormat")
    private val format = SimpleDateFormat("dd.MM.yyyy")
    private val currentDate = Date()
    private val formattedDate = format.format(currentDate)

    @SuppressLint("SimpleDateFormat")
    val dbFormat = SimpleDateFormat("yyyy-MM-dd")

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = AnalyticsBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        repository = (activity as MainActivity).repository

        setDatesAsToday()
        setListeners()
        calcData()

        binding.calcData.setOnClickListener {
            calcData()
```

```
        }
    }

    private fun calcData() {
        CoroutineScope(Dispatchers.IO).launch {
            val dateFrom = format.parse(binding.dateFromInput.text.toString())
            val dateTo = format.parse(binding.dateToInput.text.toString())
            val orders = repository.getAllCoffeeOrders(
                dbFormat.format(dateFrom),
                dbFormat.format(dateTo)
            )
            val orderQuan = orders.size
            val orderSum = calcSum(orders)
            val orderAvgPrice = orderSum / orderQuan

            val contents = mutableListOf<OrderContent>()
            orders.forEach {
                contents += repository.getAllOrderContents(it)
            }
            val contentMap = mutableMapOf<CoffeeMenuItem, Int>()
            contents.forEach {
                val item = repository.getMenuItemById(it.item)
                contentMap[item!!] = it.quantity
            }
            val contentSorted = contentMap.toList().sortedByDescending { (_, value) -> value }

            withContext(Dispatchers.Main) {
                binding.orderQuan.text = resources.getString(R.string.order_quantity, orderQuan)
                binding.orderSum.text = resources.getString(R.string.order_sum,
orderSum.toString())

                if (orderAvgPrice.isNaN()) {
                    binding.orderAvgPrice.text = resources.getString(R.string.order_price, "0")
                } else
```

```
                binding.orderAvgPrice.text =
                    resources.getString(R.string.order_price,
convertFloatToString(orderAvgPrice))

                adapter = MenuItemAdapter(requireContext(), contentSorted.subList(0, 5))
                binding.list.adapter = adapter
            }
        }
    }

    private fun calcSum(orders: List<CoffeeOrder>): Float {
        var sum = 0f
        for (order in orders) {
            sum += order.sum
        }
        return sum
    }

    private fun setListeners() {
        binding.dateFromInput.setOnClickListener {
            val calendar = Calendar.getInstance()
            val year = calendar.get(Calendar.YEAR)
            val month = calendar.get(Calendar.MONTH)
            val day = calendar.get(Calendar.DAY_OF_MONTH)

            val datePickerDialog = DatePickerDialog(
                requireContext(),
                DatePickerDialog.OnDateSetListener { _, selectedYear, selectedMonth,
selectedDay ->
                    val selectedDayString =
                        if (selectedDay < 10) "0$selectedDay" else "$selectedDay"
                    val selectedMonthString =
                        if (selectedMonth < 9) "0${selectedMonth + 1}" else "${selectedMonth + 1}"
                    val formattedDate = "$selectedDayString.$selectedMonthString.$selectedYear"
```

```
            binding.dateFromInput.setText(formattedDate)
            checkDates()
        },
        year,
        month,
        day
    )

    datePickerDialog.show()
}

binding.dateToInput.setOnClickListener {
    val calendar = Calendar.getInstance()
    val year = calendar.get(Calendar.YEAR)
    val month = calendar.get(Calendar.MONTH)
    val day = calendar.get(Calendar.DAY_OF_MONTH)

    val datePickerDialog = DatePickerDialog(
        requireContext(),
        DatePickerDialog.OnDateSetListener { _, selectedYear, selectedMonth,
selectedDay ->
            val selectedDayString =
                if (selectedDay < 10) "0$selectedDay" else "$selectedDay"
            val selectedMonthString =
                if (selectedMonth < 9) "0${selectedMonth + 1}" else "${selectedMonth + 1}"
            val formattedDate = "$selectedDayString.$selectedMonthString.$selectedYear"
            binding.dateToInput.setText(formattedDate)
            checkDates()
        },
        year,
        month,
        day
    )
```

```
            datePickerDialog.show()
        }
    }

    private fun checkDates() {
        val dateFrom = format.parse(binding.dateFromInput.text.toString())
        val dateTo = format.parse(binding.dateToInput.text.toString())
        if (dateFrom != null && dateTo != null) {
            if (dateFrom.after(dateTo)) {
                binding.dateToInput.error = "Дата закінчення не може бути раніше дати початку"
                binding.calcData.isEnabled = false
            } else {
                binding.dateFromInput.error = null
                binding.dateToInput.error = null
                binding.calcData.isEnabled = true
            }
        }
    }

    private fun setDatesAsToday() {
        binding.dateFromInput.setText(formattedDate)
        binding.dateToInput.setText(formattedDate)
    }

    private fun convertFloatToString(value: Float): String {
        val decimalFormat = DecimalFormat("0.00")
        return decimalFormat.format(value)
    }
}
```

ДОДАТОК Е.2

MenuItemAdapter.kt

```
class MenuItemAdapter(val context: Context, private val contentSorted: List<Pair<CoffeeMenuItem, Int>>) : RecyclerView.Adapter<MenuItemAdapter.MenuItemViewHolder>() {
    lateinit var binding: OrderItemLittleBinding

    inner class MenuItemViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

    }

    override fun onCreateViewHolder(
      parent: ViewGroup,
      viewType: Int
    ): MenuItemAdapter.MenuItemViewHolder {
      binding = OrderItemLittleBinding.inflate(LayoutInflater.from(parent.context), parent, false)
      return MenuItemViewHolder(binding.root)
    }

    override fun onBindViewHolder(holder: MenuItemAdapter.MenuItemViewHolder, position: Int) {
      val item = contentSorted[position]
      binding.itemTitle.text = item.first.name
      Glide.with(context)
        .load(item.first.imagePath)
        .into(binding.photo)
    }
    override fun getItemCount(): Int {
      return contentSorted.size
    }
  }
```

ДОДАТОК Ж.1

CatalogsFragment.kt

```
class CatalogsFragment : Fragment(), CatalogInterface {
    lateinit var binding: CatalogsBinding
    lateinit var adapter: CatalogListAdapter
    private lateinit var repository: CoffeeProffyRepository
    private var isNewOrder = false

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = CatalogsBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        isNewOrder = arguments?.getBoolean("NEW_ORDER") ?: false
        if (isNewOrder)
            binding.addItem.visibility = View.GONE

        repository = (activity as MainActivity).repository
        binding.addItem.setOnClickListener {

findNavController().navigate(R.id.action_catalogsFragment_to_newCatalogFragment)
        }
        updateAdapter()
    }

    override fun onEditClick(catalog: MenuType) {
        val bundle = Bundle().also {
            it.putSerializable("CATALOG", catalog)
```

```
        }
        Log.d("CATALOG BUNDLE OUT", bundle.toString())
        findNavController().navigate(R.id.action_catalogsFragment_to_newCatalogFragment,
bundle)
    }

    @SuppressLint("NotifyDataSetChanged")
    override fun onDeleteClick(catalog: MenuType) {
        val dialogBuilder = AlertDialog.Builder(requireContext())

        dialogBuilder.setMessage("Тип меню буде повністю видалений. Ви впевнені?")
            .setCancelable(false)
            .setPositiveButton("Так") { dialog, _ ->
                CoroutineScope(Dispatchers.IO).launch {
                    repository.deleteCatalog(catalog)
                    withContext(Dispatchers.Main) {
                        dialog.dismiss()
                    }
                }
            }
            .setNegativeButton("Відмінити") { dialog, _ ->
                dialog.dismiss()
            }

        val dialog = dialogBuilder.create()
        dialog.setOnDismissListener { updateAdapter() }
        dialog.show()
    }

    override fun onCatalogClick(catalog: MenuType) {
        val bundle = Bundle().also {
            it.putSerializable("CATALOG", catalog)
            it.putBoolean("NEW_ORDER", isNewOrder)
            arguments?.getSerializable("ORDER")?.let { order ->
```

Продовження дод. Ж.1

```
                it.putSerializable("ORDER", order)
                it.putBoolean("ORDER_CHANGING", true)
            }
        }
        Log.d("CATALOG BUNDLE OUT", bundle.toString())
        findNavController().navigate(R.id.action_catalogsFragment_to_catalogFragment,
bundle)
    }

    private fun updateAdapter() {
        CoroutineScope(Dispatchers.IO).launch {
            val menuTypes = repository.getAllCatalogs()
            Log.d("CATALOGS", menuTypes.toString())
            adapter = CatalogListAdapter(menuTypes, isNewOrder)
            adapter.setCallback(this@CatalogsFragment)
            withContext(Dispatchers.Main) {
                binding.catalogs.adapter = adapter
            }
        }
    }
}
```

ДОДАТОК Ж.2

CatalogListAdapter.kt

```
class CatalogListAdapter(private val catalogs: List<MenuType>, val isNewOrder: Boolean) :
    RecyclerView.Adapter<CatalogListAdapter.CatalogViewHolder>() {
    lateinit var binding: CatalogListItemBinding
    lateinit var callbackInterface: CatalogInterface

    override fun onCreateViewHolder(
        parent: ViewGroup,
        viewType: Int
    ): CatalogViewHolder {
        binding = CatalogListItemBinding.inflate(LayoutInflater.from(parent.context), parent,
false)
        return CatalogViewHolder(binding.root)
    }

    override fun onBindViewHolder(holder: CatalogViewHolder, position: Int) {
        val catalog = catalogs[position]
        binding.catalogName.text = catalog.name
        binding.catalogName.setOnClickListener {
            callbackInterface.onCatalogClick(catalog)
        }

        if (isNewOrder) {
            binding.editCatalog.visibility = View.GONE
            binding.deleteCatalog.visibility = View.GONE
        } else {
            binding.editCatalog.setOnClickListener {
                callbackInterface.onEditClick(catalog)
            }
            binding.deleteCatalog.setOnClickListener {
                callbackInterface.onDeleteClick(catalog)
            }
        }
    }
```

```
    override fun getItemCount(): Int {
        return catalogs.size
    }

    fun setCallback(callbackOnClick: CatalogInterface) {
        callbackInterface = callbackOnClick
    }

    inner class CatalogViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView)
{


    }
}
```

ДОДАТОК Ж.3

CatalogInterface.kt

```
interface CatalogInterface {
  fun onEditClick(catalog: MenuType)
  fun onDeleteClick(catalog: MenuType)
  fun onCatalogClick(catalog: MenuType)
}
```

ДОДАТОК И

NewCatalogFragment.kt

```
class NewCatalogFragment : Fragment() {
  lateinit var binding: NewCatalogBinding
  private lateinit var repository: CoffeeProffyRepository

  override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
  ): View {
    binding = NewCatalogBinding.inflate(inflater, container, false)
    return binding.root
  }

  override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    repository = (activity as MainActivity).repository

    val catalogToEdit = arguments?.getSerializable("CATALOG") as MenuType?
    if (catalogToEdit != null) {
      binding.catalogNameInput.setText(catalogToEdit.name)
    }

    binding.save.setOnClickListener {
      CoroutineScope(Dispatchers.IO).launch {
        if (catalogToEdit != null) repository.updateCatalog(
          MenuType(
            id = catalogToEdit.id,
            name = binding.catalogNameInput.text.toString()
          )
        )
        else repository.insertCatalog(
          MenuType(name = binding.catalogNameInput.text.toString())
```

```
                )
                withContext(Dispatchers.Main) {

findNavController().navigate(R.id.action_newCatalogFragment_to_catalogsFragment)
                }
            }
        }
    }
```

ДОДАТОК К.1

CatalogFragment.kt

```
class CatalogFragment : Fragment(), MenuItemInterface {

    private val orderViewModel: NewOrderViewModel by activityViewModels()
    lateinit var binding: CatalogBinding
    lateinit var adapter: MenuItemListAdapter
    lateinit var repository: CoffeeProffyRepository
    lateinit var catalog: MenuType
    var isNewOrder = false

    private val callback = object : OnBackPressedCallback(true) {
        override fun handleOnBackPressed() {
            findNavController().navigate(R.id.mainMenuFragment)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = CatalogBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        requireActivity().onBackPressedDispatcher.addCallback(viewLifecycleOwner,
callback)

        repository = (activity as MainActivity).repository

        Log.d("CATALOG BUNDLE IN 1", arguments.toString())
```

```
    isNewOrder = arguments?.getBoolean("NEW_ORDER") ?: false

    catalog = arguments?.getSerializable("CATALOG") as MenuType
    binding.title.text = catalog.name
    updateAdapter(catalog)
}

private fun updateAdapter(catalog: MenuType) {
    CoroutineScope(Dispatchers.IO).launch {
        val menuItems = repository.getAllMenuItems(catalog.id)
        Log.d("CATALOG MENU ITEMS", menuItems.toString())
        withContext(Dispatchers.Main) {
            adapter = MenuItemListAdapter(requireContext(), menuItems, isNewOrder)
            adapter.setCallback(this@CatalogFragment)
            with(binding) {
                menuItemsList.adapter = adapter
                if (isNewOrder) addItem.visibility = View.GONE
                else
                    addItem.setOnClickListener {
                        val bundle = Bundle().also {
                            it.putSerializable("CATALOG", catalog)
                        }
                        findNavController().navigate(
                            R.id.action_catalogFragment_to_newMenuItemFragment,
                            bundle
                        )
                    }
            }
        }
    }
}

override fun deleteItem(menuItem: CoffeeMenuItem) {
    val dialogBuilder = AlertDialog.Builder(requireContext())
```

```
        dialogBuilder.setMessage("Позиція меню буде повністю видалена. Ви впевнені?")
            .setCancelable(false)
            .setPositiveButton("Так") { dialog, _ ->
                CoroutineScope(Dispatchers.IO).launch {
                    repository.deleteMenuItem(menuItem)
                    withContext(Dispatchers.Main) {
                        dialog.dismiss()
                    }
                }
            }
            .setNegativeButton("Відмінити") { dialog, _ ->
                dialog.dismiss()
            }

        val dialog = dialogBuilder.create()
        dialog.setOnDismissListener { updateAdapter(catalog) }
        dialog.show()
    }

    override fun onItemClick(menuItem: CoffeeMenuItem) {
        val bundle = Bundle().also {
            it.putSerializable("CATALOG", catalog)
            it.putSerializable("MENU_ITEM", menuItem)
            arguments?.getSerializable("ORDER")?.let { order ->
                it.putSerializable("ORDER", order)
                it.putBoolean("ORDER_CHANGING", true)
            }
        }
        if (isNewOrder) {
            orderViewModel.addItemToOrder(menuItem)
            findNavController().navigate(
                R.id.action_catalogFragment_to_newOrderFragment,
                bundle
            )
```

```
        } else {
            findNavController().navigate(
                R.id.action_catalogFragment_to_newMenuItemFragment,
                bundle
            )
        }
    }
}
```

ДОДАТОК К.2

MenuItemListAdapter.kt

```
class MenuItemListAdapter(val context: Context, private val menuItems:
List<CoffeeMenuItem>, private val isNewOrder: Boolean) :
    RecyclerView.Adapter<MenuItemListAdapter.MenuItemViewHolder>() {
    lateinit var binding: OrderItemBinding
    lateinit var callbackInterface: MenuItemInterface

    inner class MenuItemViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {

    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
MenuItemViewHolder {
      binding = OrderItemBinding.inflate(LayoutInflater.from(parent.context), parent, false)
      return MenuItemViewHolder(binding.root)
    }

    override fun getItemCount(): Int {
      return menuItems.size
    }

    override fun onBindViewHolder(holder: MenuItemViewHolder, position: Int) {
      val coffeeMenuItem = menuItems[position]
      with(binding) {
        itemTitle.text = coffeeMenuItem.name
        itemPrice.text = convertFloatToString(coffeeMenuItem.price)
        itemQuantity.visibility = View.GONE

        Glide.with(context)
          .load(coffeeMenuItem.imagePath)
          .into(photo)

        if (isNewOrder)
```

```
            deleteItem.visibility = View.GONE
        else
            deleteItem.setOnClickListener {
                callbackInterface.deleteItem(coffeeMenuItem)
            }

        root.setOnClickListener {
            callbackInterface.onItemClick(coffeeMenuItem)
        }
    }
}

fun setCallback(onClickInterface: MenuItemInterface) {
    callbackInterface = onClickInterface
}

private fun convertFloatToString(value: Float): String {
    val decimalFormat = DecimalFormat("0.00")
    return decimalFormat.format(value)
}
}
```

ДОДАТОК К.3

MenuItemInterface.kt

```
interface MenuItemInterface {
  fun deleteItem(menuItem: CoffeeMenuItem)
  fun onItemClick(menuItem: CoffeeMenuItem)
}
```

ДОДАТОК Л

NewMenuItemFragment.kt

```
class NewMenuItemFragment : Fragment() {
    lateinit var binding: NewMenuItemBinding
    lateinit var repository: CoffeeProffyRepository
    var imagePath: String? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = NewMenuItemBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        repository = (activity as MainActivity).repository
        val catalog = arguments?.getSerializable("CATALOG") as MenuType
        Log.d("CATALOG BUNDLE IN 2", catalog.toString())
        val menuItemToEdit = arguments?.getSerializable("MENU_ITEM") as
CoffeeMenuItem?

        with(binding) {
            if (menuItemToEdit != null) {
                header.text = activity?.resources?.getString(R.string.editing)
                menuItemNameInput.setText(menuItemToEdit.name)
                menuItemPriceInput.setText(menuItemToEdit.price.toString())

                if (menuItemToEdit.imagePath != null) {
                    Glide.with(requireContext())
                        .load(menuItemToEdit.imagePath)
                        .into(image)
```

```
    }
}

chooseImage.setOnClickListener {
    val intent = Intent(Intent.ACTION_GET_CONTENT)
    intent.type = "image/*"
    startActivityForResult(intent, GALLERY_REQUEST_CODE)
}

done.setOnClickListener {
    val name = menuItemNameInput.text.toString()
    val price = menuItemPriceInput.text.toString()
    if (name.isEmpty() || price.isEmpty()) return@setOnClickListener

    CoroutineScope(Dispatchers.IO).launch {
        if (menuItemToEdit != null) {
            repository.updateMenuItem(
                CoffeeMenuItem(
                    id = menuItemToEdit.id,
                    name = name,
                    price = price.toFloat(),
                    imagePath = imagePath,
                    menuType = catalog.id
                )
            )
        } else
            repository.insertMenuItem(
                CoffeeMenuItem(
                    name = name,
                    price = price.toFloat(),
                    imagePath = imagePath,
                    menuType = catalog.id
                )
            )
```

```
                withContext(Dispatchers.Main) {
                    val bundle = Bundle().also {
                        it.putSerializable("CATALOG", catalog)
                    }
                    Log.d("CATALOG BUNDLE OUT", bundle.toString())
                    findNavController().navigate(
                        R.id.action_newMenuItemFragment_to_catalogFragment,
                        bundle
                    )
                }
            }
        }
    }

    @Deprecated("Deprecated in Java")
    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)

        if (requestCode == GALLERY_REQUEST_CODE && resultCode ==
Activity.RESULT_OK) {
            if (data != null) {
                val imageUri = data.data
                binding.image.setImageURI(imageUri)
//                imagePath = imageUri?.path
                imagePath = getRealPathFromURI(imageUri)
                Log.d("IMAGE PATH", imagePath.toString())
            }
        }
    }
    private fun getRealPathFromURI(uri: Uri?): String? {
        val returnCursor = requireContext().contentResolver.query(uri!!, null, null, null, null)
        val                                    nameIndex                                    =
returnCursor!!.getColumnIndex(OpenableColumns.DISPLAY_NAME)
```

```
            val sizeIndex = returnCursor.getColumnIndex(OpenableColumns.SIZE)
            returnCursor.moveToFirst()
            val name = returnCursor.getString(nameIndex)
            val size = returnCursor.getLong(sizeIndex).toString()
            val file = File(requireContext().filesDir, name)
            try {
                val inputStream: InputStream? =
requireContext().contentResolver.openInputStream(uri)
                val outputStream = FileOutputStream(file)
                var read = 0
                val maxBufferSize = 1 * 1024 * 1024
                val bytesAvailable: Int = inputStream?.available() ?: 0
                //int bufferSize = 1024;
                val bufferSize = Math.min(bytesAvailable, maxBufferSize)
                val buffers = ByteArray(bufferSize)
                while (inputStream?.read(buffers).also {
                        if (it != null) {
                            read = it
                        }
                    } != -1) {
                    outputStream.write(buffers, 0, read)
                }
                Log.e("File Size", "Size " + file.length())
                inputStream?.close()
                outputStream.close()
                Log.e("File Path", "Path " + file.path)

            } catch (e: java.lang.Exception) {
                Log.e("Exception", e.message!!)
            }
            return file.path
        }
    }
```

ДОДАТОК М

SettingsFragment.kt

```
class SettingsFragment : Fragment() {
  lateinit var binding: SettingsBinding

  override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
  ): View {
    binding = SettingsBinding.inflate(inflater, container, false)
    return binding.root
  }

  override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    val preferences = activity?.getPreferences(Context.MODE_PRIVATE)
    with (preferences!!) {
      val caffeeName = this.getString("CAFFEE_NAME", "Coffee Proffy")
      val password = this.getString("PASSWORD", null)
      val passwordOn = this.getBoolean("PASSWORD_ON", false)
      val resultOn = this.getBoolean("ORDER_INFO", false)

      binding.caffeeNameInput.setText(caffeeName)
      if (password != null) {
        binding.passwordInput.setText(password)
        binding.confirmPasswordInput.setText(password)
      }
      binding.passwordOn.isChecked = passwordOn
      binding.resultOn.isChecked = resultOn
    }

    setEnabilityPassFields(binding.passwordOn.isChecked)
```

```
        binding.passwordOn.setOnCheckedChangeListener { _, isChecked ->
            setEnabilityPassFields(isChecked)
        }

        binding.confirmPasswordInput.doAfterTextChanged { editable ->
            if (editable.toString() != binding.passwordInput.text.toString()) {
                binding.confirmPasswordInput.error = "Пароль має співпадати"
            } else {
                binding.confirmPasswordInput.error = null
            }
        }

        binding.saveButton.setOnClickListener {
            with (preferences.edit()) {
                putString("CAFFEE_NAME", binding.caffeeNameInput.text.toString())
                putString("PASSWORD", binding.confirmPasswordInput.text.toString())
                putBoolean("PASSWORD_ON", binding.passwordOn.isChecked)
                putBoolean("ORDER_INFO", binding.resultOn.isChecked)
                apply()
            }
            findNavController().navigate(R.id.action_settingsFragment_to_mainMenuFragment)
        }
    }

    private fun setEnabilityPassFields(checked: Boolean) {
        if (!checked) {
            binding.passwordInput.isEnabled = false
            binding.confirmPasswordInput.isEnabled = false
        } else {
            binding.passwordInput.isEnabled = true
            binding.confirmPasswordInput.isEnabled = true
        }
    }
}
```

ДОДАТОК Н

RoomDatabase.kt

```
@Database(
    entities = [CoffeeOrder::class, OrderContent::class, MenuType::class, CoffeeMenuItem::class],
    version = 1)
abstract class RoomDataBase : RoomDatabase() {

    abstract fun coffeeProffyDao(): CoffeeProffyDao

    companion object {
        @Volatile
        private var INSTANCE: RoomDataBase? = null

        fun getDatabase(context: Context): RoomDataBase {
            val tempInstance = INSTANCE
            if (tempInstance != null)
                return tempInstance

            synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    RoomDataBase::class.java,
                    "coffeeproffy_db")
                    .build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```

CoffeeProffyDao.kt

```
@Dao
interface CoffeeProffyDao {
    @Insert
    fun insertMenuType(menuType: MenuType)
    @Insert
    fun insertMenuItem(menuItem: CoffeeMenuItem)
    @Insert
    fun insertCoffeeOrder(coffeeOrder: CoffeeOrder)
    @Insert
    fun insertOrderContent(orderContent: OrderContent)

    @Update
    fun updateMenuType(menuType: MenuType)
    @Update
    fun updateMenuItem(menuItem: CoffeeMenuItem)
    @Update
    fun updateCoffeeOrder(coffeeOrder: CoffeeOrder)
    @Update
    fun updateOrderContent(orderContent: OrderContent)

    @Delete
    fun deleteMenuType(menuType: MenuType)
    @Delete
    fun deleteMenuItem(menuItem: CoffeeMenuItem)
    @Delete
    fun deleteCoffeeOrder(coffeeOrder: CoffeeOrder)
    @Delete
    fun deleteOrderContent(orderContent: OrderContent)

    @Query("SELECT * FROM MenuType")
    fun getAllMenuTypes(): List<MenuType>
    @Query("SELECT * FROM CoffeeMenuItem WHERE menuType = :menuTypeId")
    fun getAllMenuItems(menuTypeId: Int): List<CoffeeMenuItem>
```

```
    @Query("SELECT * FROM CoffeeOrder WHERE date BETWEEN :dateFrom AND
:dateTo")
    fun getAllCoffeeOrders(dateFrom: String, dateTo: String): List<CoffeeOrder>
    @Query("SELECT * FROM OrderContent WHERE orderId = :orderId")
    fun getAllOrderContents(orderId: Int): List<OrderContent>

    @Query("SELECT * FROM CoffeeOrder WHERE date = :date ORDER BY num DESC
LIMIT 1")
    fun getLastOrderOfTheDay(date: String): CoffeeOrder?

    @Query("SELECT * FROM CoffeeMenuItem WHERE id = :id")
    fun getMenuItemById(id: Int): CoffeeMenuItem?
}
```

ДОДАТОК Р

CoffeeProffyRepository.kt

```
class CoffeeProffyRepository(context: Context) {
  private val dao: CoffeeProffyDao = RoomDataBase.getDatabase(context).coffeeProffyDao()

  suspend fun insertCatalog(menuType: MenuType) {
    dao.insertMenuType(menuType)
  }

  suspend fun updateCatalog(menuType: MenuType) {
    dao.updateMenuType(menuType)
  }

  suspend fun deleteCatalog(menuType: MenuType) {
    dao.deleteMenuType(menuType)
  }

  suspend fun getAllCatalogs(): List<MenuType> {
    return dao.getAllMenuTypes()
  }

  suspend fun insertMenuItem(menuItem: CoffeeMenuItem) {
    dao.insertMenuItem(menuItem)
  }

  suspend fun updateMenuItem(menuItem: CoffeeMenuItem) {
    dao.updateMenuItem(menuItem)
  }

  suspend fun deleteMenuItem(menuItem: CoffeeMenuItem) {
    dao.deleteMenuItem(menuItem)
  }

  suspend fun getAllMenuItems(menuTypeId: Int): List<CoffeeMenuItem> {
```

```
    return dao.getAllMenuItems(menuTypeId)
}

suspend fun insertCoffeeOrder(coffeeOrder: CoffeeOrder) {
    dao.insertCoffeeOrder(coffeeOrder)
}

suspend fun updateCoffeeOrder(coffeeOrder: CoffeeOrder) {
    dao.updateCoffeeOrder(coffeeOrder)
}

suspend fun deleteCoffeeOrder(coffeeOrder: CoffeeOrder) {
    dao.deleteCoffeeOrder(coffeeOrder)
}

suspend fun getAllCoffeeOrders(dateFrom: String, dateTo: String): List<CoffeeOrder> {
    return dao.getAllCoffeeOrders(dateFrom, dateTo)
}

suspend fun insertOrderContent(orderContent: OrderContent) {
    dao.insertOrderContent(orderContent)
}

suspend fun updateOrderContent(orderContent: OrderContent) {
    dao.updateOrderContent(orderContent)
}

suspend fun deleteOrderContent(orderContent: OrderContent) {
    dao.deleteOrderContent(orderContent)
}

suspend fun getAllOrderContents(orderId: Int): List<OrderContent> {
    return dao.getAllOrderContents(orderId)
}
```

```
    suspend fun getAllOrderContents(coffeeOrder: CoffeeOrder): List<OrderContent> {
        return dao.getAllOrderContents(coffeeOrder.id)
    }

    suspend fun getLastOrderOfTheDay(date: String): CoffeeOrder? {
        return dao.getLastOrderOfTheDay(date)
    }

    suspend fun getMenuItemById(id: Int): CoffeeMenuItem? {
        return dao.getMenuItemById(id)
    }
}
```

ДОДАТОК С.1

CoffeeOrder.kt

```
@Entity
data class CoffeeOrder(
    @PrimaryKey (autoGenerate = true)
    val id: Int = 0,
    var date: String,
    val num: Int,
    var sum: Float
) : Serializable
```

OrderContent.kt

```
@Entity(
    foreignKeys = [
        ForeignKey(
            entity = CoffeeMenuItem::class,
            parentColumns = arrayOf("id"),
            childColumns = arrayOf("item"),
            onDelete = ForeignKey.CASCADE
        ),
        ForeignKey(
            entity = CoffeeOrder::class,
            parentColumns = arrayOf("id"),
            childColumns = arrayOf("orderId"),
            onDelete = ForeignKey.CASCADE
        )]
)
data class OrderContent(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val orderId: Int,
    val item: Int,
    var quantity: Int = 1
)
```

ДОДАТОК С.3

MenuType.kt

```
@Entity
data class MenuType(
  @PrimaryKey (autoGenerate = true)
  val id: Int = 0,
  var name: String
) : Serializable
```

CoffeeMenuItem.kt

```
@Entity(
    foreignKeys = [ForeignKey(
        entity = MenuType::class,
        parentColumns = arrayOf("id"),
        childColumns = arrayOf("menuType"),
        onDelete = ForeignKey.CASCADE
    )]
)
data class CoffeeMenuItem(
    @PrimaryKey (autoGenerate = true)
    val id: Int = 0,
    var name: String,
    var price: Float,
    var imagePath: String? = null,
    var menuType: Int
) : Serializable
```