**МІНІСТЕСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КІБЕРНЕТИКИ,**
**ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ІНЖЕНЕРІЇ**
**НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ**
**КАФЕДРА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ ТА ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ**

**Допущено до захисту:**

Завідувач кафедри

_____ д. е. н., проф. П. М. Грицюк

«_____» _____ 2024 р.

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня «бакалавр»

за освітньо-професійною програмою «Інформаційні системи і технології»

спеціальності 126 «Інформаційні системи та технології»

на тему: «Мобільний застосунок служби кур'єрської доставки»

**Виконав:**

здобувач вищої освіти 2 курсу із скороченим терміном навчання, групи ІСТ-21інт

**Антонюк Богдан Сергійович**

**Керівник:**

д. е. н., проф. П. М. Грицюк

**Рецензент:**

канд. техн. наук, доцент

Барановський С. В.

Рівне 2024

**ЗМІСТ**

# СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

| | | |
|---|---|---|
| ПК | – | персональний комп’ютер |
| БД | – | база даних |
| АТ | – | акціонерне товариство |
| АСУ | – | автоматизована система управління |
| АІС | – | автоматизована інформаційна система |
| ТЗ | – | технічне завдання |

# ВСТУП

«Не марнуй часу, бо це матеріал, з якого зроблене життя» - ця влучна фраза, одного з засновників США Бенджаміна Франкліна, саме сьогодні актуальна, як ніколи. За останні сто років наш світ сильно змінився, з'явилося неймовірна кількість різноманітних приладів, технологій які полегшують нам щоденні турботи: пилососи, пральні машини, комп'ютери та телефони. Варто відзначити, що більшість сучасних технологій, без яких складно уявити сучасний світ з'явилися буквально декілька десятків років тому. Зважаючи на це ми можемо сказати що нас світ не просто стрімко розвивається, розвиток ніби прискорює сам себе, і з кожним роком ми все швидше та в більшій кількості отримуємо доступ до, здавалося б, фантастичних технологій. На мою думку в епоху настільки стрімкого розвитку, для того аби мати змогу встигати за всіма новинками, що з'являються у світі, важливим вмінням є раціональне використання часу.

В сучасному світі люди користуються різноманітними сервісами та послугами, що допомагають їм вберегти час. Одним з таких популярних сервісів є доставка готової їжі. Особливого розвитку ця сфера зазнала під час локдауну викликаного хворобою «Covid-19». Заборона відвідування різноманітних закладів харчування змусила власників знаходити альтернативи, для збереження власного бізнесу. Це дало величезний поштовх для розвитку сфери доставки, а разом з тим виникла необхідність в створені максимально зручних та інтуїтивно зрозумілих додатків для смартфонів, через які можна з легкістю замовити їжу.

**Актуальність теми досліджень.** Актуальність теми дослідження полягає в тому, що з кожним роком все більше людей користується послугами доставки їжі, що зумовлює необхідність створення сучасних, зручних та надійних мобільних додатків. Розробка таких додатків сприяє не лише зростанню комфорту користувачів, а й розвитку бізнесу у сфері громадського харчування. Виходячи з усього вище перерахованого, можна сміливо сказати, що сервіси з

доставки будуть ще набирати популярність, що в свою чергу потребуватиме розробки нових зручних додатків та їхньої постійної модернізації.

**Розробленість теми роботи.** Розробкою мобільних додатків для кур'єрської доставки їжі займалися численні компанії, зокрема такі гіганти, як Uber Eats, Glovo, та локальні стартапи. Проте, незважаючи на успіхи, існує ряд невирішених проблем, таких як: забезпечення високого рівня безпеки даних користувачів, інтеграція з різними платіжними системами, оптимізація для різних мобільних платформ та забезпечення безперебійної роботи додатків в умовах високого навантаження.

**Об'єктом дослідження** даної дипломної роботи є сфера громадського харчування.

**Предметом дослідження** є проектування та розробка мобільного додатку для служби кур'єрської доставки їжі.

**Мета досліджень і завдання.** Метою дослідження є розробка зручного, надійного та функціонального мобільного застосунку для кур'єрської доставки їжі, який задовольнятиме потреби як користувачів, так і бізнесу. ***Завдання дослідження включають:***

1. Аналіз ринку існуючих мобільних застосунків для доставки їжі.
2. Визначення вимог користувачів та бізнесу до мобільного застосунку.
3. Розробка технічного завдання для мобільного застосунку.
4. Проєктування архітектури та структури застосунку.
5. Реалізація основних функцій застосунку.
6. Тестування та оптимізація застосунку.
7. Впровадження та аналіз результатів роботи застосунку.

**Методи дослідження.** У даній роботі використані такі методи досліджень:

***Аналіз літературних джерел*.** Аналіз літературних джерел дозволяє вивчити теоретичні основи розробки мобільних додатків, а також ознайомитися

з найновішими дослідженнями та технологіями в даній сфері. Цей метод включає перегляд наукових статей, книг, технічних документів, а також онлайн-ресурсів, що стосуються програмування, дизайну та користувацького досвіду (UX/UI).

***Аналіз ринку.*** Аналіз ринку мобільних додатків для доставки їжі спрямований на визначення актуальних вимог користувачів і бізнесу. Цей метод передбачає дослідження існуючих додатків-конкурентів, їхніх функцій, переваг та недоліків. Аналіз також включає збір даних про користувацькі відгуки, рейтинги додатків у магазинах App Store та Google Play.

***Методи проєктування.*** Методи проєктування використовуються для створення архітектури та структури мобільного додатка.

*До них належать:*

- Розробка технічного завдання (ТЗ): визначення вимог до функціоналу додатка, його інтерфейсу та взаємодії з користувачем.
- Моделювання архітектури: створення блок-схем, діаграм та моделей, що відображають логічну і фізичну структуру додатка.
- Прототипування: розробка інтерактивних прототипів інтерфейсу, які дозволяють протестувати користувацький досвід ще до початку програмування.

***Програмування.*** Програмування включає безпосередню реалізацію функціоналу мобільного додатка.

*Використовуються такі методи:*

- Кодування: написання коду на мові програмування, що обрана для розробки (наприклад, Kotlin або Java для Android).
- Інтеграція з серверами та базами даних: налаштування взаємодії додатка з серверною частиною та базами даних, які зберігають інформацію про замовлення, меню та користувачів.
- Інтеграція з зовнішніми сервісами: підключення картографічних сервісів для відстеження доставки та інших необхідних API.

***Тестування***. Тестування проводиться для виявлення та виправлення помилок у роботі додатка.

Методи тестування включають:

- Функціональне тестування: перевірка роботи всіх функцій додатка відповідно до технічного завдання.
- Тестування користувацького інтерфейсу (UI): оцінка зручності та інтуїтивності інтерфейсу додатка.
- Тестування на різних пристроях: перевірка сумісності додатка з різними моделями смартфонів і версіями операційних систем.
- Тестування продуктивності: оцінка швидкості роботи додатка, часу завантаження та обробки запитів.

Використання цих методів дозволяє комплексно підійти до розробки мобільного додатка для кур’єрської доставки їжі, забезпечити його високу якість та відповідність потребам користувачів та бізнесу.

# 1. Розділ. Дослідження обраної сфери

## 1.1 Аналіз сфери громадського харчування

Аналіз сфери громадського харчування Сфера громадського харчування є надзвичайно важливою частиною соціально-економічного життя суспільства. Вона включає в себе широкий спектр різноманітних закладів, починаючи від ресторанів та кафе, закінчуючи різноманітними фаст-фудами, стріт-фудами та іншими закладами швидкого харчування. У сучасних умовах розвиток індустрії громадського харчування набуває особливого значення. Вона впливає не лише на економічні показники, але й на якість життя населення, культуру споживання, а також здоров'я громадян.

***Мета дослідження***. Метою цього розділу є детальне дослідження та аналіз сучасного стану сфери громадського харчування, порівняння різних підходів до даної сфери. У розділі буде представлено огляд основних видів закладів громадського харчування, їх класифікацію, а також огляд нових трендів у сфері громадського харчування. Особлива увага буде приділена інноваційним технологіям у громадському харчуванні та впливу пандемії COVID-19 на цю сферу. Саме пандемія внесла певні корективи та дала поштовх до розвитку сфери доставок.

***Статистичні дані.*** Для кращого розуміння поточного стану сфери громадського харчування в м. Рівне, розглянемо декілька ключових статистичних даних за допомогою сайту [25]:

Кількість закладів громадського харчування: станом на 2024 рік у м. Рівне функціонують близько 869 закладів, включаючи ресторани, кафе, фаст-фуди та стріт-фуди.

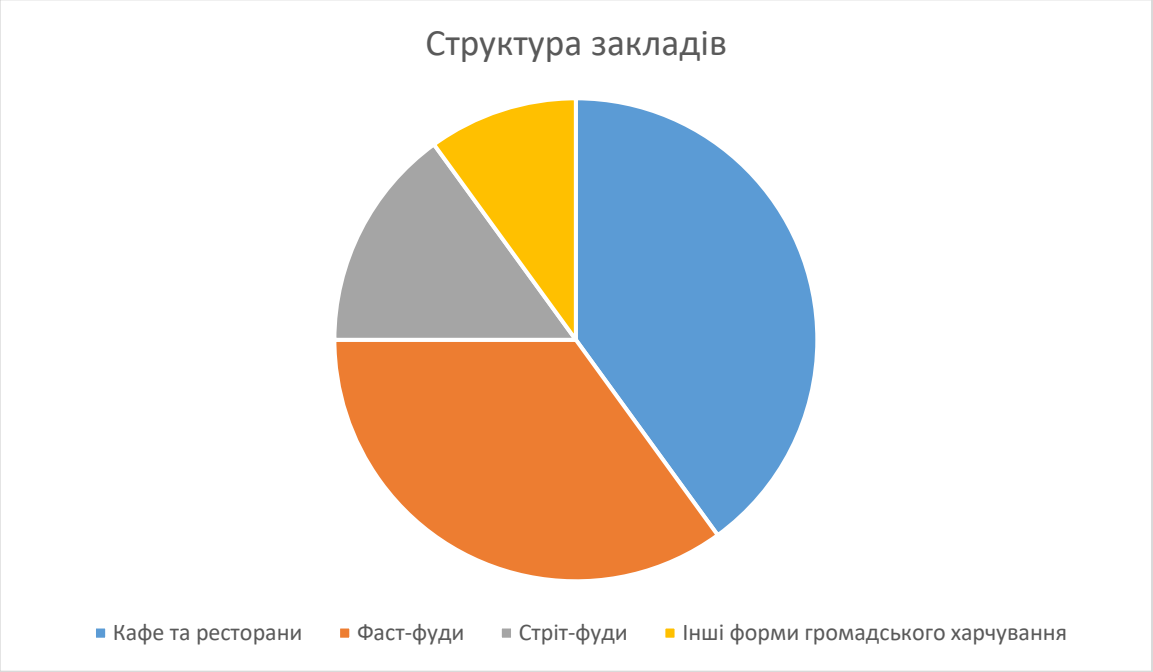



Рис. 1 Структура закладів громадського харчування у місті Рівне

***Структура закладів:*** приблизно 40% складають кафе та ресторани, 35% — фаст-фуди, 15% — стріт-фуди, 10% — інші форми громадського харчування.

***Обсяги ринку:*** згідно з даними джерела [26], у 2024 році обсяг ринку громадського харчування в місті склав близько 500 млн гривень.

**Види закладів громадського харчування**.

*Заклади громадського харчування можна класифікувати за різними критеріями:*

- Тип обслуговування: заклади з самообслуговуванням, обслуговуванням офіціантами та комбінованим обслуговуванням.
- Цінова категорія: бюджетні, середньоцінові та преміум-класу.
- Концепція меню: традиційні ресторани, спеціалізовані кафе (кава, десерти), заклади швидкого харчування (фаст-фуди), стріт-фуди.

**Нові тренди у сфері громадського харчування.**

*Сучасні тенденції розвитку сфери громадського харчування включають:*

- Інноваційні технології: впровадження електронних меню, систем безконтактної оплати, мобільних додатків для замовлення їжі.

- «Еко-френдлі» підхід: використання біорозкладних матеріалів, зменшення харчових відходів, впровадження енергозберігаючих технологій.
- Здорове харчування: збільшення попиту на органічні, вегетаріанські та безглютенові страви.
- Доставка їжі: зростання популярності сервісів доставки їжі, особливо після пандемії COVID-19. У 2024 році близько 30% всіх замовлень у м. Рівне здійснювались через сервіси доставки.

**Вплив пандемії COVID-19.** Пандемія COVID-19 внесла значні корективи у сферу громадського харчування. У період локдаунів багато закладів були змушені тимчасово закритися або скоротити свою діяльність. Зміна моделей споживання, збільшення попиту на доставку їжі. Це призвело активного впровадження безконтактних сервісів, мобільних додатків для замовлення їжі.

## 1.2 Порівняння традиційного закладу харчування та службою доставки їжі.

В даному розділі ми порівняємо два підходи у сфері громадського харчування. З одного боку буде більш класичний варіант традиційне кафе, з іншого боку буде виступати служба з доставки їжі. Ми спробуємо з'ясувати який варіант бізнесу більш прибутковий та є більш актуальним на сьогоднішній день.

***Для порівняння використаємо теоретичне кафе за наступним описом:***

**1. Площа приміщення**. Для кафе, яке обслуговує 20-50 людей на день, рекомендується така загальна площа:

- Зал для відвідувачів: 30-50 квадратних метрів.
- Кухня: 15-20 квадратних метрів.
- Складські приміщення: 5-10 квадратних метрів.
- Санвузли та інші допоміжні приміщення: 5-10 квадратних метрів.

*Загальна площа кафе становитиме приблизно 55-90 квадратних метрів.*

**2. Кількість персоналу.** Для ефективного обслуговування відвідувачів та належного функціонування кафе, необхідно мати такий склад персоналу:

- Кухарі: 2-3 людини.
- Офіціанти: 2-3 людини.
- Посудомийник: 1 людина.
- Адміністратор: 1 людина.
- Прибиральник (може бути сумісник з іншою посадою): 1 людина.

***Таким чином, загальна кількість персоналу складає 6-9 людей.***

**3. Додаткові аспекти.**

*Меблі та обладнання для залу:*

- Столи та стільці для відвідувачів.
- Меблі для очікування та зона відпочинку.

*Обладнання для кухні:*

- Плити та духовки.

- Холодильники та морозильні камери.
- Мийні стійки та посудомийні машини.
- Робочі поверхні для приготування їжі.
- Витяжні системи та вентиляція.

*Складські приміщення:*

- Полички та стелажі для зберігання продуктів.
- Окремі зони для зберігання сухих та свіжих продуктів.

**4. Комунальні послуги.** Посилаючись на теоретичну інформацію з інтернету, можемо визначити скільки ресурсів використовує даний заклад.

- Електроенергія (від 930 кВт до 1200 кВт)
- Вода (від 14 $м^3$ до 17 $м^3$)
- Вивіз сміття

Тепер завдяки досить точному опису нашого закладу ми можемо порахувати скільки грошей потрібно для відкриття даного закладу та вартість його утримування щомісяця. Дані по цінам на послуги та обладнання візьмемо з інтернету.

Орієнтовна вартість оренди приміщення у центрі міста Рівне, 250 гривень за $м^2$, отже очікувана вартість оренди на місяць буде складати від 13 750 гривень до 22 500 гривень.

Зарплати персоналу, ще один щомісячний платіж який варто врахувати в розрахунках. Дані по заробітнім платам у місті рівне також були взяті з інтернету.

Кухарі – 18 000 гривень, з урахуванням податків отримаємо суму 27 300 гривень.

Офіціанти – 15 000 гривень, з урахуванням податків отримаємо суму 22 750 гривень.

Адміністратор – 22 000 гривень, з урахуванням податків отримаємо суму 33 450 гривень.

Посудомийник / прибиральник – 10 000 гривень, з урахуванням податків отримаємо суму 15 155 гривень.

Орієнтовні витрати на персонал складають від 148 705 до 198 755 гривень.

Далі варто обрахувати витрати на комунальні послуги, знаючи приблизну кількість приладів що будуть працювати можемо обрахувати скільки електроенергії вони потребують, орієнтовно ми отримали від 930 кВт до 1200 кВт, використовуючи сайт «Мінфін», можемо дізнатися вартість електроенергії для підприємців у рівному, а саме 243,49 за 1 мВт електроенергії орієнтовна вартість енергопостачання становить від 226,5 до 292,2 гривень.

Використовуючи цей самий сайт можемо побачити вартість водопостачання та водовідведення, що у Рівненській області складає 35,52 за $м^3$ орієнтована вартість водопостачання та водовідведення становить від 470 до 570 гривень.

Орієнтовна вартість вивозу сміття становить від 642 до 1384 гривень.

Далі обрахуємо вартість побутової техніки яку необхідно буде закупити для функціонування кафе. При перерахунку меблів вартість вказана за одиницю, дані про ціну взято з інтернету.

**Зал для відвідувачів**

***Стільці та столи:***

- 5 столів на 4 особи (загалом 20 місць). – від 20 000 до 35 000 гривень.
- 5 столів на 2 особи (загалом 10 місць). – від 17 000 до 24 000 гривень.
- Додаткові стільці для запасу (10-15 стільців). – від 3250 до 6500 гривень.

***Стійка/барна зона:***

- Барна стійка. – від 17 000 до 26 000 гривень.
- Барні стільці (4-6 шт). – від 3500 до 7200 гривень.

***Меблі для очікування:***

- Диван або лавка для зони очікування (1-2 шт). – від 5000 до 7000 гривень.
- Кавові столики (1-2 шт.). – від 3000 до 6250 гривень.

***Декоративні елементи:***

- Рослини в горщиках(7-10 шт.). – від 2500 до 3200 гривень.
- Настінні полиці для декору та зберігання.(5-7 шт.) – від 1500 до 2100 гривень.

**Кухня**

***Робочі поверхні:***

- Робочі столи з нержавіючої сталі (2-3 шт). – від 2200 до 2600 гривень.
- Духова пічка. – від 20 000 до 23 000 гривень.
- Індукційна плита. – від 20 000 до 26 000 гривень.
- Мийка з нержавіючої сталі. – від 5500 до 6800 гривень.
- Полиці для сушіння посуду. – від 5400 до 7200 гривень.

***Зберігання:***

- Холодильники для зберігання продуктів (1-2 шт.). – від 96 000 до 150 000 гривень.
- Полички та стелажі для інвентарю та посуду. – від 5400 до 7200 гривень.

***Інші меблі:***

- Шафи для зберігання особистих речей персоналу. – від 4600 до 5500 гривень.
- Відкриті стелажі для посуду та кухонного обладнання. – від 5400 до 7200 гривень.

***Санітарні зони:***

- Умивальники (1-2 шт). – від 750 до 3200 гривень.
- Унітаз. – від 2900 до 4500 гривень.

- Дзеркала. – від 450 до 800 гривень
- Меблі для зберігання туалетного паперу, рушників та інших засобів гігієни. – від 1700 до 6800 гривень

***Санвузол для персоналу:***

- Умивальник. – від 750 до 3200 гривень.
- Унітаз. – від 2900 до 4500 гривень.
- Шафи для зберігання особистих речей(5-7шт). – від 1700 до 4200 гривень

***Адміністративна зона:***

- Робочий стіл. – від 2500 до 4500 гривень
- Крісло для адміністратора. – від 4500 до 7800 гривень
- Шафа для документів. – від 2900 до 4500 гривень.

Орієнтована вартість всього необхідного обладнання становить від 494 750 до 1 000 000 гривень.

Тепер ми можемо подивитися загальні витрати на меблі складають від 494 750 до 1 000 000 гривень, також щомісячні витрати на функціонування нашого кафе ми витратимо від 163 795 до 223 502 гривень.

***Розглянемо теоретичну службу доставки їжі, яка теж орієнтована приблизно на 30-50 замовлень у день.***

**1. Площа приміщення**. Для закладу, що обслуговує 20-50 людей на день, рекомендується така загальна площа:

- Кухня: 15-20 квадратних метрів.
- Складські приміщення: 5-10 квадратних метрів.
- Санвузли та інші допоміжні приміщення: 5-7 квадратних метрів.

*Загальна площа становитиме приблизно 25-37 квадратних метрів.*

**2. Кількість персоналу.** Для ефективного обслуговування клієнтів, необхідно мати такий склад персоналу:

- Кухарі: 2-3 людини.
- Кур’єри: 2-3 людини.
- Адміністратор: 1 людина.

***Таким чином, загальна кількість персоналу складає 5-7 людей.***

**3. Додаткові аспекти.**

*Обладнання для кухні:*

- Плити та духовки.
- Холодильники та морозильні камери.
- Мийні стійки та посудомийні машини.
- Робочі поверхні для приготування їжі.
- Витяжні системи та вентиляція.

*Складські приміщення:*

- Полички та стелажі для зберігання продуктів.
- Окремі зони для зберігання сухих та свіжих продуктів.

**4. Комунальні послуги.** Посилаючись на теоретичну інформацію з інтернету, можемо визначити скільки ресурсів використовує даний заклад.

- Електроенергія (від 720 кВт до 890 кВт)
- Вода (від 9 $м^3$ до 12 $м^3$)
- Вивіз сміття

Перевагою даного закладу є можливість серйозно зекономити на оренді не лише через малу площу, за відсутності залу для відвідувачів, а також тому що приміщення може знаходитись у спальному районі, де оренда плата є значно меншою - 180 гривень за $м^2$, отже очікувана вартість оренди на місяць буде складати від 4 500 гривень до 6 660 гривень.

Зарплати персоналу, ще один щомісячний платіж який варто врахувати в розрахунках. Дані по заробітнім платам у місті рівне також були взяті з інтернету.

Кухарі – 18 000 гривень, з урахуванням податків отримаємо суму 27 300 гривень.

Кур'єри – 21 000 гривень, з урахуванням податків отримаємо суму 32 600 гривень.

Адміністратор – 22 000 гривень, з урахуванням податків отримаємо суму 33 450 гривень.

Орієнтовні витрати на персонал складають від 153 250 до 213 150 гривень.

Далі ми обрахуємо витрати на комуналі послуги використовуючи сайт «Мінфін», дізнаємось вартість надання комунальних послух для підприємців у м. Рівне, а саме 243,49 за 1 мВт електроенергії. Теоретично обрахувавши витрати електроенергії ми отримали від 720 кВт до 890 кВт на місяць, отже орієнтовна вартість енергопостачання становить від 174 до 217 гривень.

Використовуючи цей самий сайт можемо побачити вартість водопостачання та водовідведення, що у Рівненській області складає 35,52 за $м^3$ орієнтована вартість водопостачання та водовідведення становить від 320 до 450 гривень.

Орієнтовна вартість вивозу сміття становить від 600 до 1300 гривень.

Далі обрахуємо вартість побутової техніки яку необхідно буде закупити для функціонування кафе. При перерахунку меблів вартість вказана за одиницю, дані про ціну взято з інтернету.

**Кухня**

***Робочі поверхні:***

- Робочі столи з нержавіючої сталі (2-3 шт). – від 2200 до 2600 гривень.
- Духова пічка. – від 20 000 до 23 000 гривень.
- Індукційна плита. – від 20 000 до 26 000 гривень.
- Мийка з нержавіючої сталі. – від 5500 до 6800 гривень.
- Полиці для сушіння посуду. – від 5400 до 7200 гривень.

***Зберігання:***

- Холодильники для зберігання продуктів (1-2 шт.). – від 96 000 до 150 000 гривень.
- Полички та стелажі для інвентарю та посуду. – від 5400 до 7200 гривень.

***Інші меблі:***

- Шафи для зберігання особистих речей персоналу. – від 4600 до 5500 гривень.
- Відкриті стелажі для посуду та кухонного обладнання. – від 5400 до 7200 гривень.

***Санвузол для персоналу:***

- Умивальник. – від 750 до 3200 гривень.
- Унітаз. – від 2900 до 4500 гривень.
- Шафи для зберігання особистих речей(4-7шт). – від 1700 до 4200 гривень

***Адміністративна зона:***

- Робочий стіл. – від 2500 до 4500 гривень
- Крісло для адміністратора. – від 4500 до 7800 гривень
- Шафа для документів. – від 2900 до 4500 гривень.

Орієнтована вартість всього необхідного обладнання становить від 187 050 до 444 600 гривень.

Тепер ми можемо подивитися загальні витрати на меблі складають від 187 050 до 444 600 гривень, також щомісячні витрати на функціонування нашого кафе ми витратимо від 158 845 до 221 777 гривень.

## ВИСНОВОК

Спираючись на вище перераховані суми, ми можемо зробити наступні висновки. Сервіс що займається лише доставкою їжі потребує значно менше ресурсів для свого запуску ніж звичайне кафе. Це обумовлено відсутністю необхідності пошуку приміщення з великою площею, з хорошим розташуванням та місцем для паркування. Варто зазначити, що хоч і початкові витрати менші, щомісячні платежі практично однакові, через досить високі зарплати у кур'єрів, що майже у двічі вищі ніж у офіціантів, решта персоналу працює без змін.

Тепер до переваг традиційного кафе, в першу чергу будь який заклад такого типу, першочергово це місце відпочинку людей. Відвідувачі приходять з друзями або родиною посидіти в приємній атмосфері, поспілкуватися, приємно провести час, тому середня сума покупок в таких закладах буде вищою ніж у службі доставки, де першочерговим завданням є економія часу на приготуванні їжі. Але додаток

Доставка їжі є важливим інструментом для початку бізнесу в ресторанній сфері. Це розширює можливості для нових підприємців, які можуть почати свій шлях без необхідності значних інвестицій у фізичний простір, також мобільні додатки дозволяють збільшити охоплення ринку. Дана технологія створює можливість для малих кафе та ресторанів стати доступними для ширшої аудиторії, що підвищує їхню конкурентоспроможність та популярність.

Для вже наявних кафе, інтеграція служби доставки через мобільний додаток може стати потужним засобом для популяризації закладу. Завдяки зручному інтерфейсу клієнти можуть легко та оперативно зробити замовлення, що підвищує лояльність і задоволення споживачів. Крім того, мобільні додатки дозволяють впроваджувати програми лояльності, спеціальні акції та пропозиції, що будуть лише стимулювати повторні замовлення та привертати нових клієнтів.

Таким чином, впровадження мобільних додатків для кур'єрської доставки їжі є стратегічно важливим кроком для будь-якого бізнесу в сфері харчування. Це не лише відкриває нові можливості для розвитку та збільшення прибутків, але й допомагає закладам стати більш доступними, сучасними та привабливими для споживачів. В умовах швидкозмінного ринку, здатність адаптуватися до нових технологій стає вирішальною для збереження та зміцнення позицій на ринку.

# 2. Розділ. Проектування архітектури та структури програмного продукту

## 2.1 Проектування програмного продукту.

**Варіанти використання ПЗ.**

***Замовлення їжі сценарій:***

- Користувач відкриває додаток "FoodFlyer".
- Після реєстрації або входу в обліковий запис, користувач переходить до розділу меню.
- Обирає страви з різних категорій (піца, бургери, шаурма), переглядає фотографії та читає описи.
- Додає обрані страви до кошика.
- Перевіряє замовлення, вказує адресу доставки.
- Підтверджує замовлення.
- Користувач має змогу зв’язатися з кур’єром.

Отже мій додаток має виконувати наступні функції (Рис. 2):

a) Авторизація користувача.
b) Додавання товарів до кошика.
c) Перегляд обраних товарів та формування замовлення.
d) Перегляд даних щодо замовлення.
e) Можливість встановлення зв’язку з кур’єром.

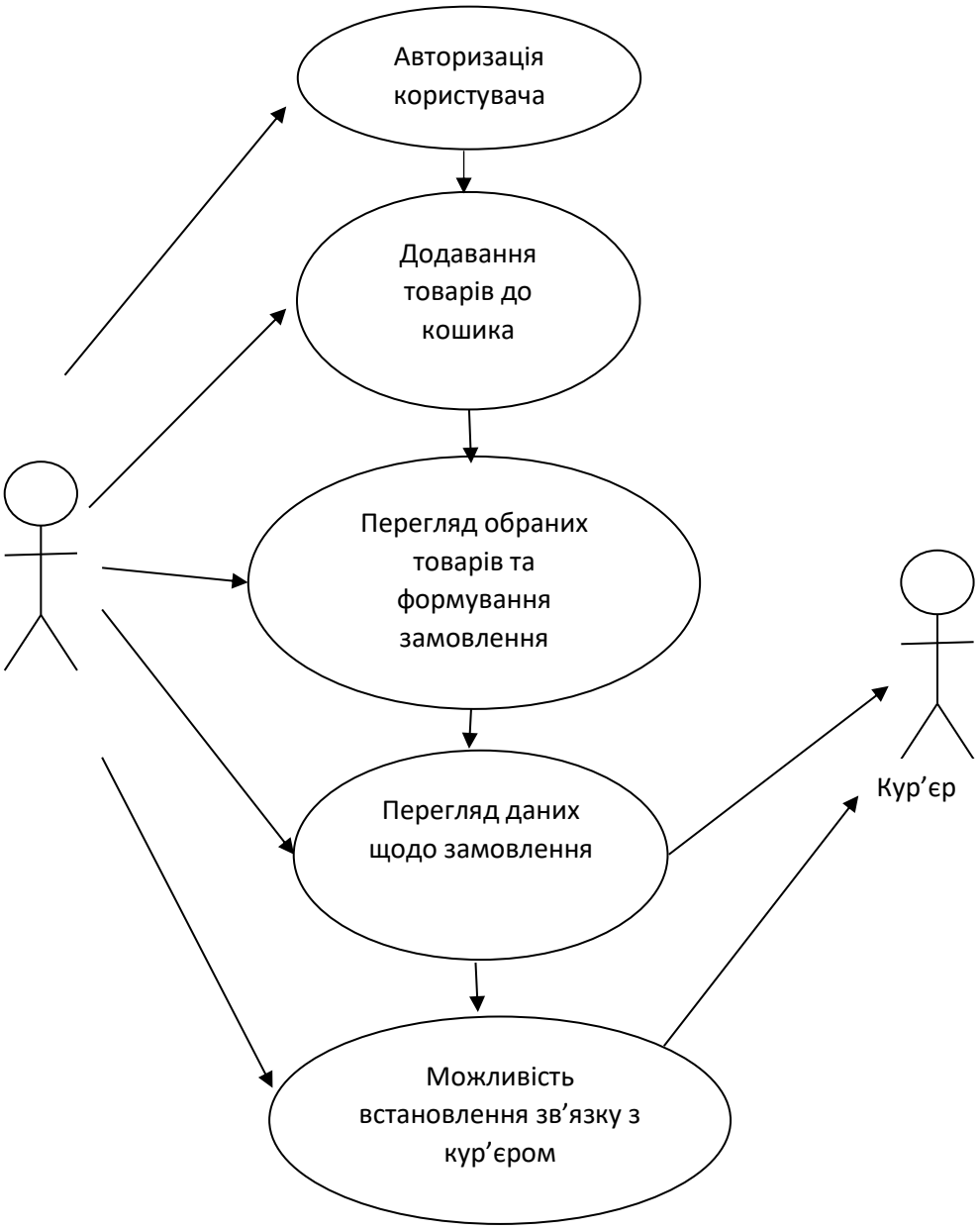



Рис. 2 Функції що реалізовані у додатку «FoodFlayer»

## Технічне завдання до програмного продукту

*Див. додаток А.*

## Створення моделі розробки програмного продукту

V-подібна модель (Рис. 3), є широко використовуваною в інженерії розробки програмного забезпечення. Ця модель є розширенням каскадної моделі і підкреслює важливість валідації і верифікації на кожному етапі розробки.

У V-подібній моделі розробка і тестування інтегровані паралельно, що дозволяє забезпечити більш високий рівень якості кінцевого продукту.

**Деталізований опис роботи моделі.**

**Ліва сторона V-моделі.**

- Планування проекту та вимог. Команда проекту проводить зустрічі з зацікавленими сторонами для збору вимог.
- Аналіз вимог та специфікацій. Вимоги аналізуються та документуються у вигляді специфікації вимог.
- Розробка структури. Визначення архітектури системи. Створення високорівневих діаграм і опису компонентів.
- Деталізована розробка. Детальна розробка кожного компоненту, включаючи внутрішні алгоритми та інтерфейси. Створення схем баз даних, специфікацій для кожного модуля.

**Права сторона V-моделі:**

- Модульне тестування. Кожен модуль або компонент тестується окремо для виявлення помилок та перевірки його коректної роботи. Використовуються тести, створені на основі деталізованого дизайну.
- Інтеграційне тестування. Перевірка взаємодії між різними модулями. Тести проводяться на основі архітектурного дизайну, щоб забезпечити, що всі частини системи правильно взаємодіють між собою.
- Системне тестування. Перевірка повної системи на відповідність всім специфікованим вимогам. Включає функціональні та нефункціональні тести.

- Кінцеве тестування. Перевірка системи або продукту кінцевими користувачами або замовниками. Оцінка на відповідність бізнес-вимогам та критеріям прийняття.


Рис. 3 V-подібна модель

## 2.2 Розробка архітектури програмного продукту

**Створення функціональної схеми програмного продукту.**

Функціональна схема– це графічне представлення системи, яке демонструє її компоненти та їхні взаємозв'язки. Вона відображає основні функції системи, їх послідовність і взаємодію між різними частинами системи.

Дана блок-схема (Рис. 4) надає наочне уявлення про структуру та функціонування системи, що полегшує її розуміння. Допомагає ідентифікувати основні компоненти та їх взаємозв'язки, що сприяє виявленню потенційних проблем та недоліків.

За допомогою поданої схеми ми бачимо алгоритм роботи програми, що починається з авторизації користувача, за допомогою зв’язку з БД. Наступний крок залежить від вибору користувача та має декілька сценаріїв виконання, що показується на схемі. Далі додаток знову використовує БД і в залежності від обраного сценарію, або ж зберігає дані (додавання товару до кошика користувача, формування замовлення), або ж слугує для виведення даних з БД (вивід вмісту кошика користувача, перегляд наявних замовлень).

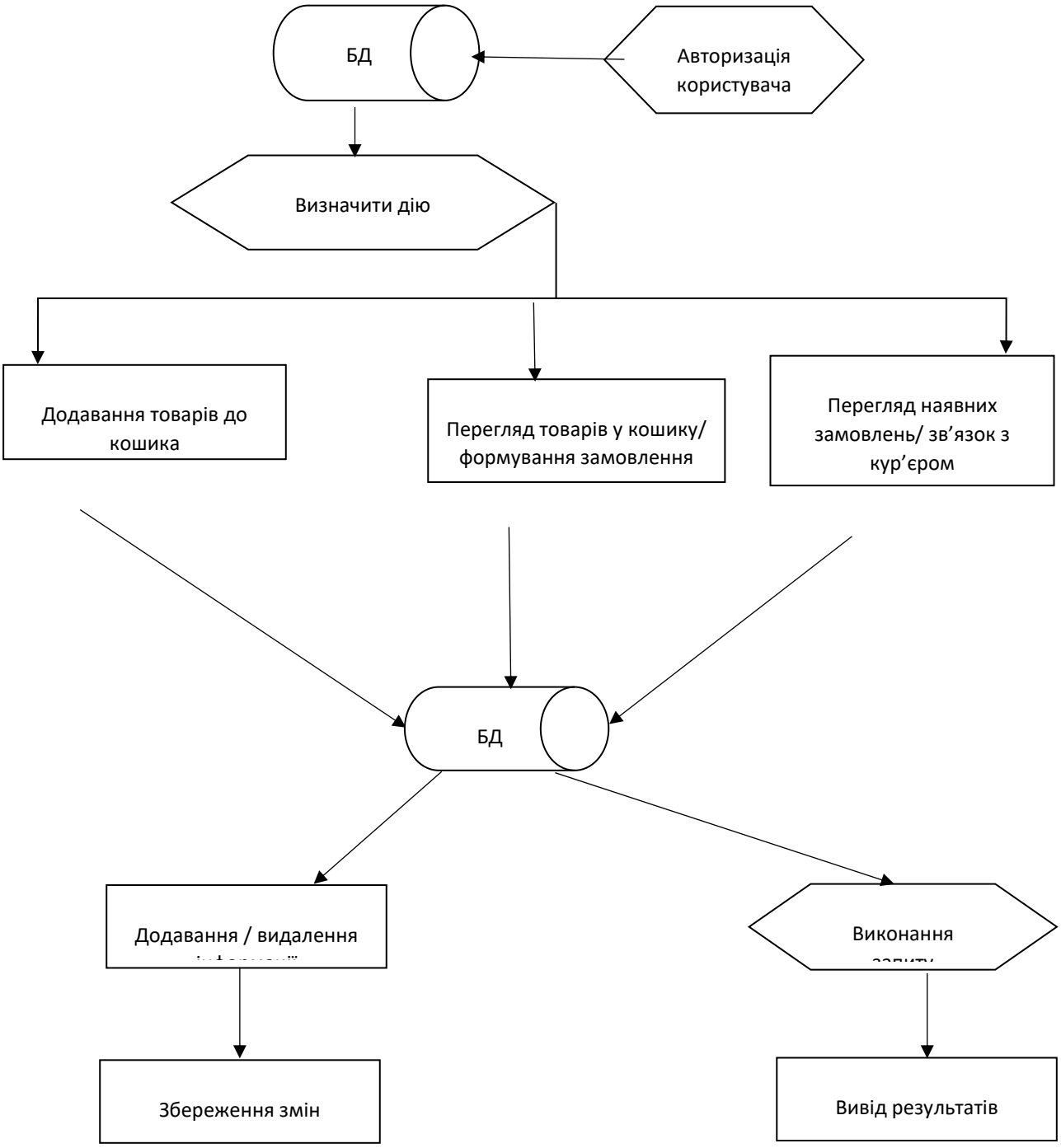



Рис. 4 Функціональна блок-схема

**Створення структурної схеми програмного продукту.**

Структурна схема (Рис. 5) є важливим інструментом для візуалізації компонентів системи та їх взаємозв'язків. Для додатку "FoodFlyer", призначеного для кур'єрської доставки їжі, структурна схема може бути поділена на кілька основних модулів: підсистема візуалізації, підсистема товарообігу, підсистема зберігання, підсистема формування замовлення.

Підсистема візуалізації. Цей компонент відповідає за взаємодію користувача з додатком.

Він включає:

- Реєстрація та Вхід. Форми для створення облікового запису та входу до системи.
- Профіль користувача. Екрани для перегляду та редагування особистої інформації, збереження адрес доставки.
- Каталог товарів. Списки страв з фотографіями, описами та цінами.
- Замовлення. Інтерфейси для додавання страв до кошика, перегляду та редагування замовлення перед підтвердженням.
- Зв’язок. Інтерфейс для зв’язку з кур'єром.

Підсистема товарообігу. Цей компонент відповідає за збереження даних з продажу товарів.

Він включає:

- Запис даних. Збереження даних про кількість та загальну вартість проданих товарів.

Підсистема формування замовлення. Цей компонент відповідає за збереження даних про всі замовлення користувача.

Він включає:

- Запис даних. Збереження даних про замовлення користувача за весь період.

Підсистема зберігання даних. Цей компонент відповідає за збереження даних користувача.

Він включає:

- Запис даних. Збереження даних з особистого кабінету користувача.

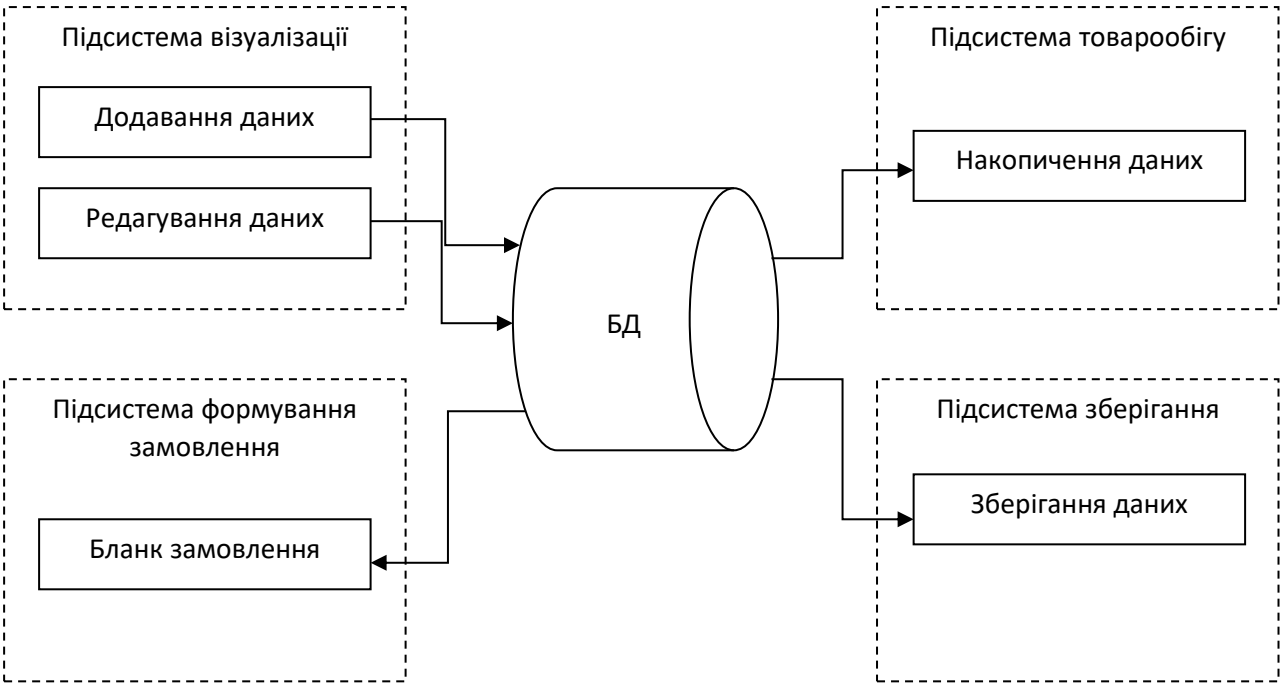



Рис. 5 Структурна схема додатку.

**Створення алгоритму функціонування програмного продукту та його підпрограм.**

Алгоритм роботи додатку "FoodFlyer"

1) Користувач відкриває мобільний додаток "FoodFlyer". Завантажуються початкові дані (попередньо збережені сесії).

2) Якщо це перший запуск програми користувач проходить реєстрацію за допомогою облікового запису Google. Додаток надсилає ці дані на бекенд-сервер. Сервер перевіряє унікальність даних. Успішна реєстрація зберігає дані користувача в базі даних.

3) Після реєстрації відбувається аутентифікація користувача додаток надсилає дані на бекенд-сервер для перевірки. Сервер аутентифікує користувача, далі користувач потрапляє до головного екрану.

4) Користувач може переглянути і редагувати особисту інформацію, у своєму профілі. Додаток дозволяє зберігати адресу доставки для зручності.

5) Користувач переглядає меню з фотографіями, описами та цінами страв. Додаток дозволяє фільтрувати страви за категоріями (наприклад, піца, бургери, шаурма).

6) Користувач вибирає страви та додає їх до кошика. Додаток оновлює локальний стан кошика та відображає підсумкову вартість замовлення.

7) Користувач може переглядати вміст кошика. Додаток дозволяє редагувати кількість страв або видаляти їх з кошика.

8) Після формування замовлення, додаток надсилає дані у БД. Сервер зберігає інформацію про замовлення в базі даних.

9) Користувач може зв’язатися з кур’єром у вкладці доставка обравши потрібне замовлення.

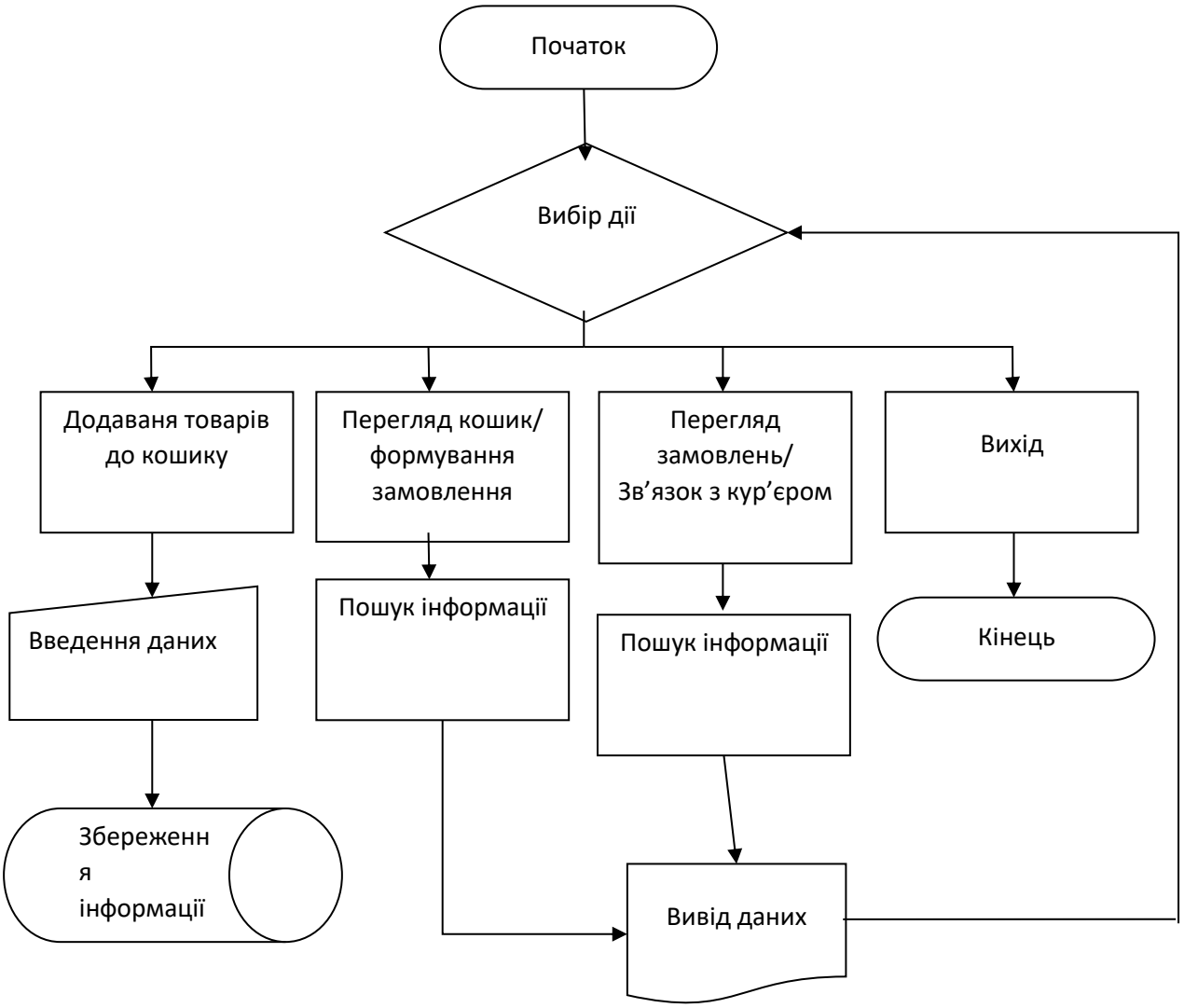



Рис. 6 Алгоритм роботи додатку

## 2.3Аналіз інструментальних засобів розробки

### Обґрунтування вибору апаратних засобів.

Firebase — це платформа розробки мобільних і веб-додатків, створена компанією Google. Вона надає широкий спектр інструментів та послуг, які допомагають розробникам створювати високоякісні додатки, покращувати їхню продуктивність та розширювати користувацьку базу. Firebase пропонує інтегровані сервіси для автентифікації, зберігання даних, хмарних функцій, аналітики та багато іншого.

Основним продуктом Firebase який цікавить нас при розробці даного додатку є «Realtime Database».

**Основні характеристики Firebase Realtime Database**

– Синхронізація в реальному часі. Дані автоматично синхронізуються на всіх клієнтах, які підключені до бази даних. Це забезпечує миттєві оновлення без необхідності в ручному оновленні або перезавантаженні додатку.

– Відправка і отримання даних. Будь-які зміни, внесені в базу даних, відразу ж відображаються на всіх підключених пристроях. Це особливо корисно для чатів, систем обміну повідомленнями.

– NoSQL база даних. Дані зберігаються у форматі JSON, що дозволяє створювати гнучкі та ієрархічні структури даних. Це забезпечує легкість в роботі з різноманітними типами даних та структурами.

– Відсутність фіксованих схем. Realtime Database не вимагає заздалегідь визначених схем таблиць, що дозволяє розробникам легко змінювати структуру даних у процесі розробки.

– Безпека та правила доступу. Використання правил безпеки Firebase дозволяє контролювати доступ до бази даних на основі автентифікації користувачів та їхніх ролей. Це забезпечує захист даних від несанкціонованого доступу.

– Правила читання та запису. Розробники можуть визначати правила, які контролюють, хто і що може читати або записувати в базу даних, що забезпечує гнучке управління доступом.

– **Опис системного програмного забезпечення функціонування програмного продукту.**

*Для можливості інсталяції та коректної роботи програмного продукту потрібно мати телефон з наступними характеристиками:*

- *ОС Android 8.0 і вище;*
- *об’єм ОП від 1 ГБ;*
- *наявність 128 Мб вільного місця;*

# ВИСНОВОК

Розробка мобільного додатку для служби кур'єрської доставки їжі є важливим кроком у задоволенні сучасних потреб споживачів у швидкому, зручному та безпечному отриманні замовлень. Враховуючи зростаючий попит на такі послуги та вплив глобальних подій, як пандемія COVID-19, цей додаток не тільки сприятиме покращенню якості обслуговування клієнтів, але й допоможе закладу харчування розширити свою аудиторію. Використання сучасних технологій забезпечить високу ефективність і прозорість процесів доставки.

Для коректної роботи додатку на пристроях користувачів слід передбачити мінімальні вимоги до апаратного забезпечення, зокрема наявність щонайменше 1 ГБ оперативної пам'яті, а для оптимальної продуктивності — 2 ГБ. У підсумку, створення цього додатку стане значущим внеском у розвиток сфери обслуговування та технологічний прогрес.

Крім того, мобільний додаток дозволить користувачам легко порівнювати різноманітні стави та вибирати найкращі пропозиції.

Застосування інтуїтивно зрозумілого інтерфейсу зробить процес замовлення максимально простим і доступним навіть для тих, хто не має великого досвіду користування технологіями.

Можливість  в середині додатку зв’язатися з кур’єром, що безпосередньо доставляє ваше замовлення зробить використання додатку зручнішим.

# 3. Розділ. Розробка мобільного застосунку

## 3.1 Вибір середовища розробки.

Після детального вивчення сфери громадського харчування, та розгляду сценаріїв використання додатку, я визначив основні критерії та вимоги до середовища розробки. В першу чергу в даному середовищі має бути доступна розробка додатків під мобільну ОС Android. Наступним важливим фактором є можливість легко інтегрувати, за допомогою середовища розробки, хмарну БД. Саме таким середовищем розробки є «Android Studio».

***Коротко про переваги даного середовища :***

- **Інтеграція з IntelliJ IDEA.** «Android Studio» базується на IntelliJ IDEA, що забезпечує доступ до широкого спектру інструментів та функцій для розробки, налагодження та тестування коду.
- **Інтелектуальний редактор коду.** Редактор коду в «Android Studio» підтримує автозавершення, рефакторинг, аналіз коду та інші функції, що значно полегшують процес написання коду.
- **Вбудований емулювальник.** «Android Studio» має потужний емулювальник, який дозволяє тестувати додатки на різних версіях Android та різних пристроях без потреби в реальних пристроях.
- **Розширені інструменти для налагодження.** Включає профайлер для аналізу продуктивності додатків, інструменти для відлагодження пам'яті, CPU та GPU, а також інструменти для тестування продуктивності додатків.
- **Інтеграція з версіонним контролем**. Підтримка систем контролю версій, таких як Git, що дозволяє легко керувати версіями коду, співпрацювати з іншими розробниками та відслідковувати зміни.
- **Інтеграція з «Firebase».** Легка інтеграція з «Firebase» для додавання функцій backend-розробки, таких як аналітика, база даних, аутентифікація, повідомлення та інші.

### 3.2 Авторизація користувача.

При відкритті додатку користувач одразу потрапляє на екран завантаження. На цьому екрані відображається зображення рис. 2.1 - 2.2., що є логотипом додатку. Екран завантаження створює перше враження про додаток та інформує користувача, що додаток завантажується. Це допомагає уникнути відчуття затримки або технічних проблем під час запуску.

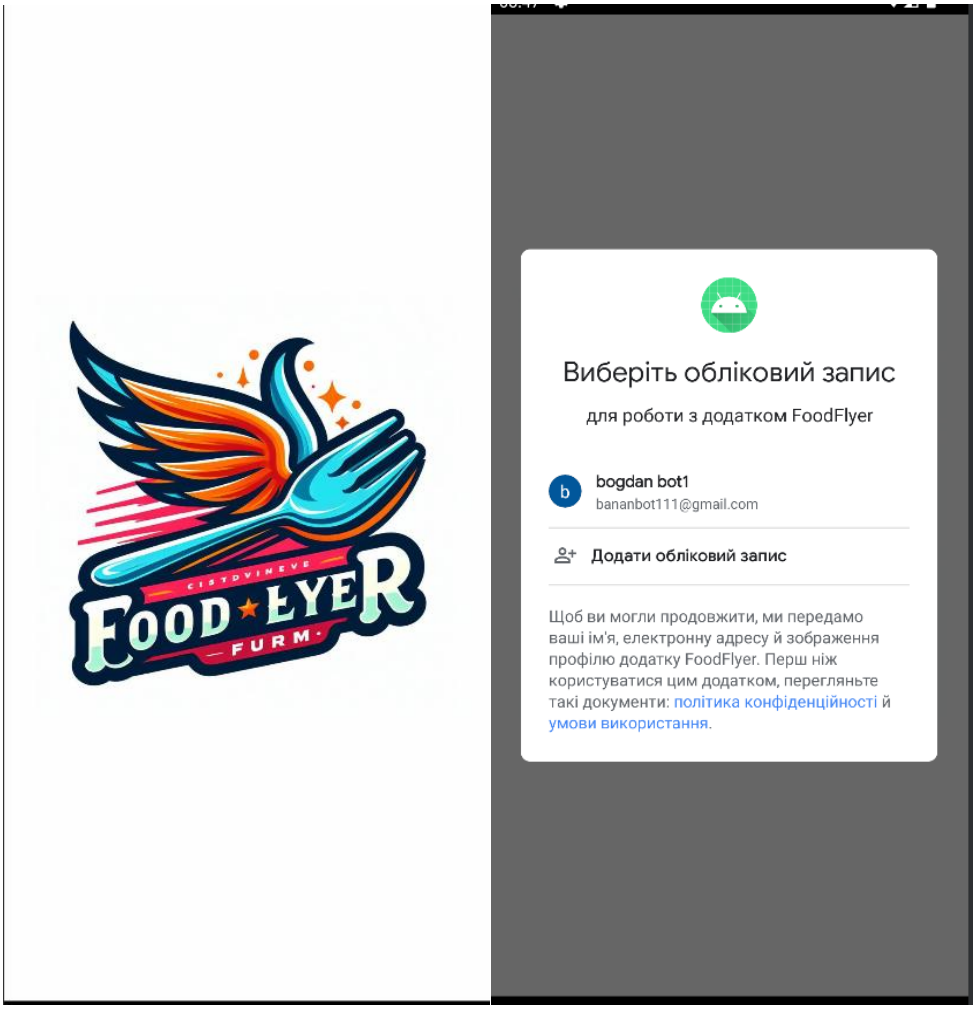



Рис. 7.1-7.2 Екран завантаження

Для кращого усвідомлення процесів які відбуваються при авторизації користувача варто поглянути на алгоритм роботи, що представлений схемою рис. 3.

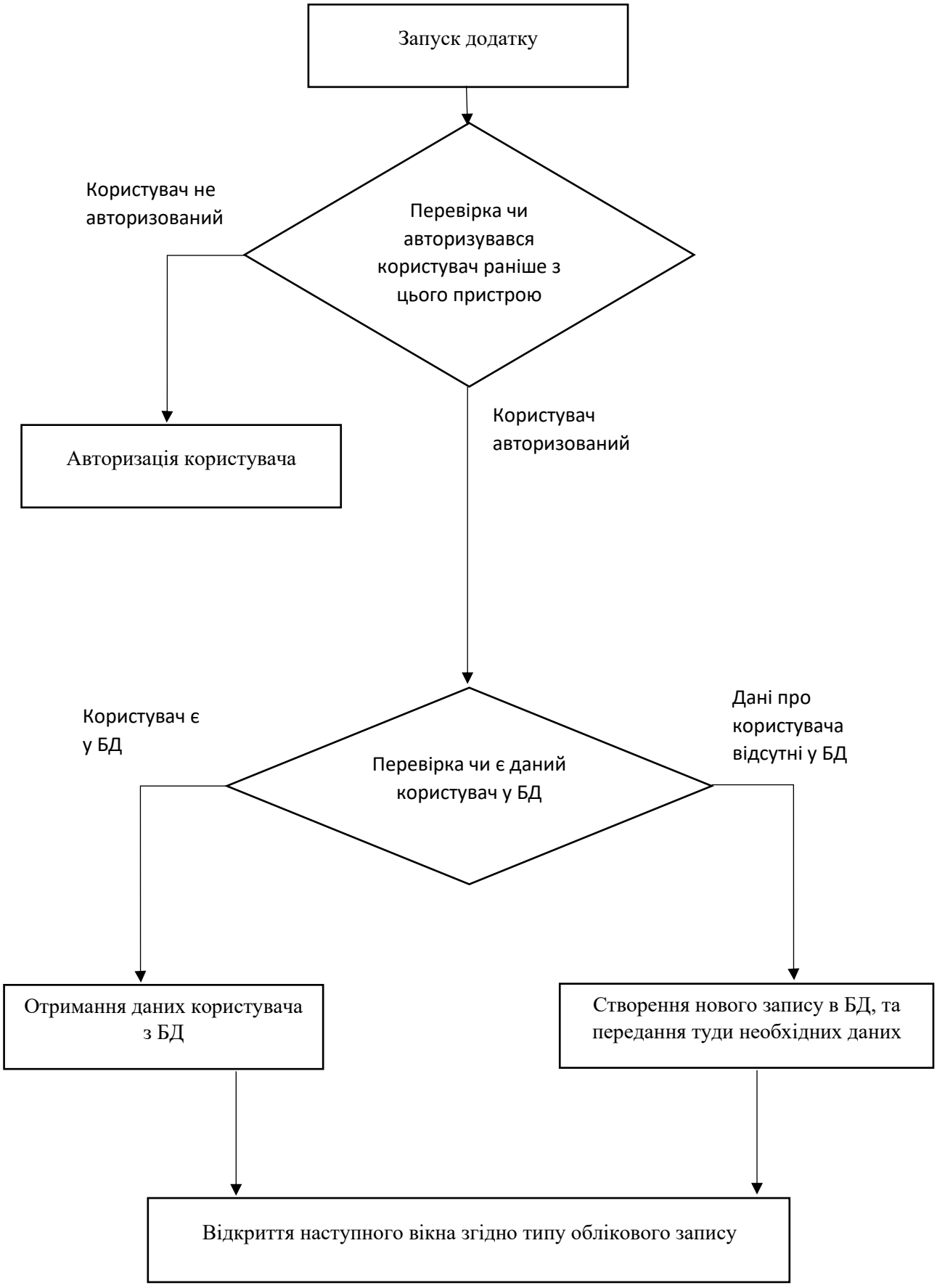



Рис. 8 Схема авторизації користувача

Для реалізації роботи даного алгоритму я написав наступний код рис. 9.1 - 9.5.

```
gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN).requestEmail().build();
gsc = GoogleSignIn.getClient( activity: this, gso);
GoogleSignInAccount account = GoogleSignIn.getLastSignedInAccount( context: this);
if (account != null) {
    String email = account.getEmail();
    String name = account.getGivenName();
    String s_name = account.getFamilyName();
    checkUserInDatabase(email,name,s_name);
}
else singIn();
```

Рис. 9.1 Перевірка чи користувач був авторизований

Цей фрагмент коду перевіряє, чи користувач вже увійшов через Google на своєму пристрої. Якщо так, він отримує інформацію про електронну адресу, ім'я та прізвище користувача. Потім він перевіряє ці дані у моїй базі даних, викликаючи функцію checkUserInDatabase. Якщо користувач не увійшов, виконується функція signIn(), щоб ініціювати процес входу.

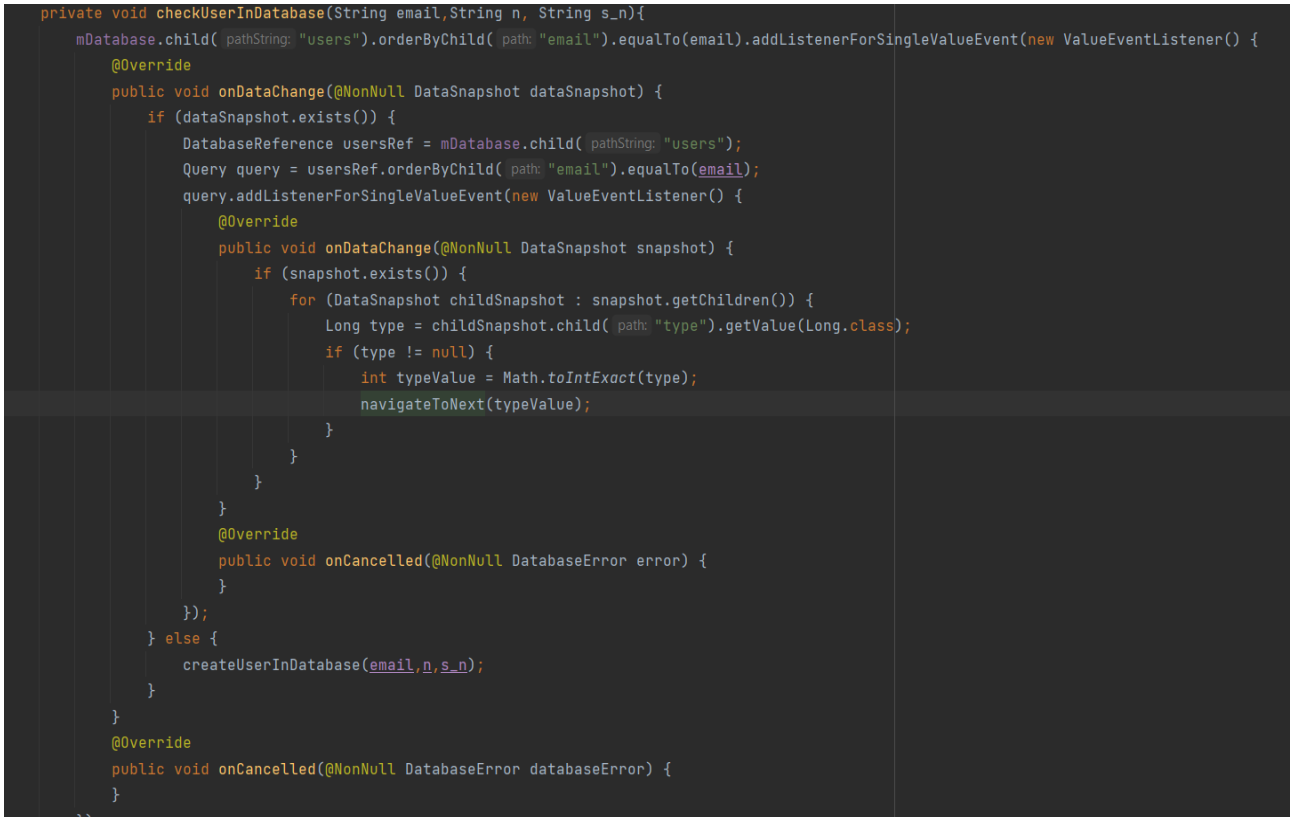
```
private void singIn() {
    Intent signintent = gsc.getSignInIntent();
    startActivityForResult(signintent,  requestCode: 1);
}
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1) {
        try {
            Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
            GoogleSignInAccount account = task.getResult(ApiException.class);
            if (account != null) {
                String email = account.getEmail();
                String name = account.getGivenName();
                String s_name = account.getFamilyName();
                checkUserInDatabase(email,name,s_name);
                register(email);
            } else {
                singIn();
            }
        } catch (ApiException e) {
            e.printStackTrace();

        }
    }
}
```

Рис. 9.2 Запуск екрану авторизації в обліковому записі

Цей код реалізує процес входу через Google. В основному методі signIn() відбувається ініціалізація входу через Google Sign-In API. Він створює намір для

входу, який потім відправляється через startActivityForResult(). У методі onActivityResult() обробляється результат входу. Коли користувач завершує вхід через Google, результат повертається у цей метод. Він перевіряє, чи вдалося отримати обліковий запис Google (GoogleSignInAccount). Якщо отримано, отримуються дані про електронну адресу, ім'я та прізвище користувача. Потім ці дані перевіряються в моїй базі даних за допомогою функції checkUserInDatabase(). Після цього викликається функція register(), яка, ймовірно, реєструє користувача з цими даними. Якщо отриманий обліковий запис Google є нульовим (якщо входу не відбулося), знову викликається метод signIn() для повторної спроби входу.

Цей код демонструє типовий процес роботи з Google Sign-In API для аутентифікації користувачів через їхні облікові записи Google.



```
private void checkUserInDatabase(String email,String n, String s_n){
    mDatabase.child( pathString: "users").orderByChild( path: "email").equalTo(email).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()) {
                DatabaseReference usersRef = mDatabase.child( pathString: "users");
                Query query = usersRef.orderByChild( path: "email").equalTo(email);
                query.addListenerForSingleValueEvent(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot snapshot) {
                        if (snapshot.exists()) {
                            for (DataSnapshot childSnapshot : snapshot.getChildren()) {
                                Long type = childSnapshot.child( path: "type").getValue(Long.class);
                                if (type != null) {
                                    int typeValue = Math.toIntExact(type);
                                    navigateToNext(typeValue);
                                }
                            }
                        }
                    }
                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {
                    }
                });
            } else {
                createUserInDatabase(email,n,s_n);
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {
        }
```

Рис. 9.3 Перевірна наявності запису про користувача у БД.

Цей метод checkUserInDatabase виконує наступні дії: Здійснює запит до бази даних Firebase для перевірки наявності користувача з вказаною електронною адресою. Прослуховує результат цього запиту через addListenerForSingleValueEvent. Якщо користувач з такою електронною адресою існує (dataSnapshot.exists()), виконується подальший запит для отримання додаткових даних користувача. Витягує значення типу користувача (type) з отриманих даних і передає його в метод navigateToNext, який перенаправляє користувача на наступний екран в залежності від цього типу. Якщо користувача з вказаною електронною адресою не знайдено, викликається метод

createUserInDatabase, що створює нового користувача в базі даних з цією адресою і іншими введеними даними. Методи onCancelled в ValueEventListener обробляють випадки, коли запит до бази даних не вдається виконати через помилку або відміну.

Цей код дозволяє ефективно виконувати перевірку та обробку даних користувачів у базі даних Firebase.

```
private void createUserInDatabase(String email, String name, String s_name) {
    DatabaseReference usersRef = mDatabase.child( pathString: "users").push();
    int type = 0;
    Map<String, Object> userData = new HashMap<>();
    userData.put( k: "email", email);
    userData.put( k: "name",name);
    userData.put( k: "s_name",s_name);
    userData.put( k: "kilkst", v: "0");
    userData.put( k: "home", v: "");
    userData.put( k: "phon", v: "");
    userData.put( k: "type", type);
    userData.put( k: "photo",  v: "0");
    usersRef.updateChildren(userData);
    navigateToNext(type);
}
```

Рис. 9.4 Створення нового запису про користувача у БД.

Цей метод createUserInDatabase створює нового користувача у базі даних Firebase з вказаною електронною адресою, іменем та прізвищем. Спочатку він отримує посилання на базу даних Firebase для колекції "users" і створює новий унікальний ключ за допомогою .push(), який створює новий вузол з унікальним ідентифікатором. Визначається початкове значення для поля "type". Дані про користувача зберігаються у Map<String, Object> userData, де кожен ключ відповідає полю в базі даних, а значення - відповідне значення для кожного поля. Викликається updateChildren(userData), щоб зберегти дані користувача у вже створеному вузлі бази даних Firebase. Після збереження даних викликається navigateToNext(type), що перенаправляє користувача на наступний екран в додатку з використанням встановленого типу користувача.

Отже, цей метод дозволяє створювати нові записи користувачів у базі даних Firebase з необхідною інформацією та виконувати подальші дії відповідно до цих даних.

```
private void navigateToNext(int type){
    Intent intent ;
    switch (type){
        case(0):
            intent=  new Intent( packageContext: this, Customer.class);
            break;
        case(1):
           intent = new Intent( packageContext: this, Courier.class);
           break;
        default:  intent=  new Intent( packageContext: this, MainActivity.class);
    }
   finish();
   startActivity(intent);
}
```

Рис. 9.5 Перехід на наступне вікно додатку в залежності від типу облікового запису.

Цей метод navigateToNext визначає, на який екран перенаправити користувача на основі значення типу користувача. Створюється змінна intent, яка буде використовуватися для зберігання наміру переходу. Виконується switch перевірка для змінної type: Якщо type дорівнює 0, створюється намір переходу до Customer класу. Якщо type дорівнює 1, створюється намір переходу до Courier класу. Для всіх інших значень type, створюється намір переходу до MainActivity класу. Викликається метод finish(), щоб закрити поточну активність. Викликається метод startActivity(intent), щоб запустити нову активність, визначену у змінній intent.

Цей метод забезпечує правильне перенаправлення користувача на відповідний екран додатку на основі його типу.

### 3.3 Головний екран користувача.

Після успішної авторизації додаток відкриє головний екран рис. 10. На даному екрані в користувача є можливість обрати вид товару з запропонованих варіантів, також ми бачимо вгорі екрану меню-бар, на якому розміщені кнопки для переходу до інших розділів додатку. Дане вікно слугує лише для переходу у інші розділи.


Рис. 10. Головний екран користувача.

Алгоритм роботи даного вікна наведений у схемі на рис. 11.

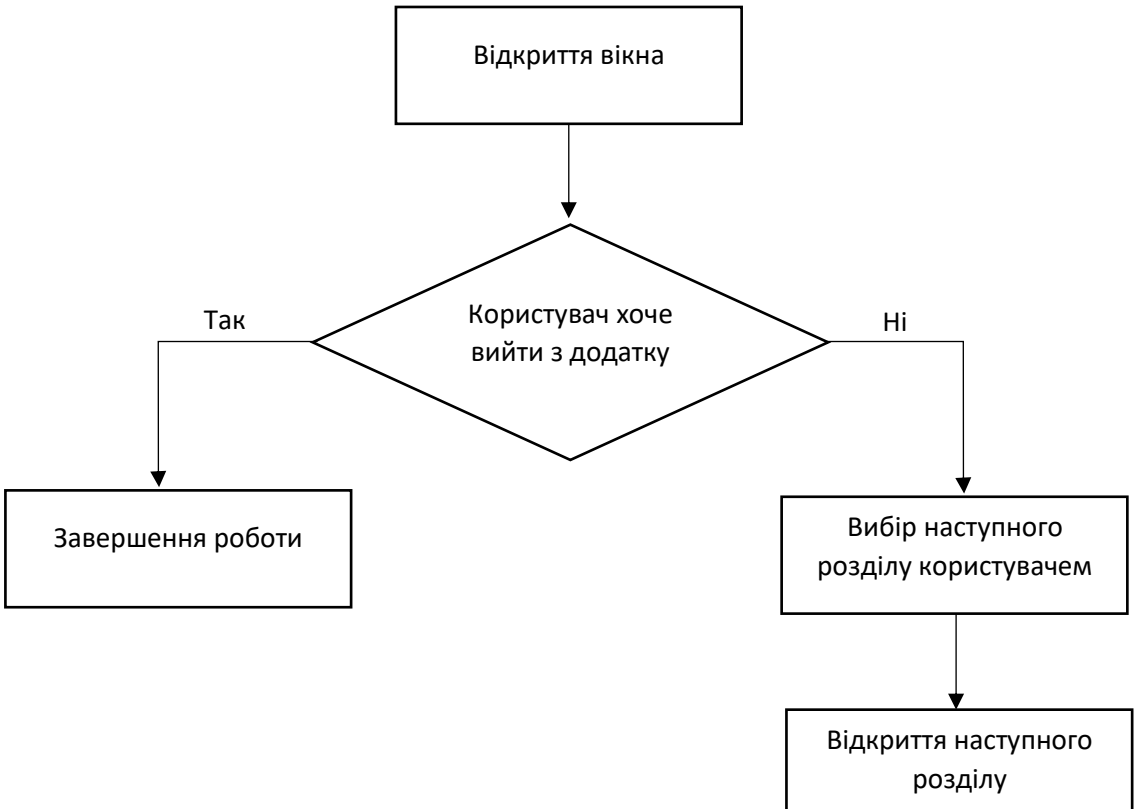


Рис. 11 Алгоритм роботи головного екрану користувача.

Для реалізації роботи даного алгоритму я написав наступний код рис. 12.1-12.4.

```
imageView_acount.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent =  new Intent( packageContext: Customer.this, Customer_acount.class);
        finish();
        startActivity(intent);
    }
});
```

Рис. 12.1 Перехід на вікно додатку з інформацією про обліковий запис.

```
basketButton. setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent( packageContext: Customer.this, Basket.class );
        finish();
        startActivity(intent);
    }
```

Рис. 12.2 Перехід на вікно додатку з інформацією про перелік товарів у кошику.

```
developButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent =  new Intent( packageContext: Customer.this, Develop.class);
        finish();
        startActivity(intent);
    }
});
```

Рис. 12.3 Перехід на вікно додатку з інформацією про доставку товарів.

```
linearLayoutBurger.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent( packageContext: Customer.this, Burger.class);
        finish();
        startActivity(intent);
    }
});

linearLayoutPizza.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent( packageContext: Customer.this, Pizza.class);
        finish();
        startActivity(intent);
    }
});

linearLayoutShavuha.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent( packageContext: Customer.this, Buritto.class);
        finish();
        startActivity(intent);
    }
});
```

Рис. 12.4 Перехід на вікна товарів за категоріями для вибору конкретних страв.

Оскільки дане вікно слугує лише для переходу на інші екрани користувачем, метод що містяться на ньому практично однакові працює за наступним принципом створюється змінна intent, яка буде використовуватися для зберігання наміру переходу(вибір вікна куди користувач потрапить). Викликається метод finish(), щоб закрити поточну активність. Викликається метод startActivity(intent), щоб запустити нову активність, визначену у змінній intent.

Цей метод забезпечує правильне перенаправлення користувача на відповідний екран додатку на основі його вибору.

### 3.4 Клас «Helper».

Окремо варто відзначити кастомний клас «Helper», який я розробив для оптимізації роботи додатку. Даний клас був створений для спрощення роботи з «Firebase». Його мета – забезпечити зручний інтерфейс для виконання таких завдань, як збереження та отримання даних з бази даних, завантаження зображень у хмарне сховище та завантаження зображень з хмарного сховища, а також оновлення інформації користувачів. Додатково, клас надає функції для роботи з елементами інтерфейсу, такі як додавання значка з кількістю товарів до зображення кошика. Цей клас об'єднує типові операції, що часто виконуються в додатку, з метою підвищення ефективності розробки та полегшення роботи з Firebase. Код, який дозволяє реалізувати виконання вищеперерахованих завдань наведено на рис. 13.1 – 13.8.

```
public void getDateFromDatabase(String key,String email, OnDataReceivedListener listener) {
    mDatabase.child(key).orderByChild( path: "email").equalTo(email).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()) {
                DatabaseReference usersRef = mDatabase.child( pathString: "users");
                Query query = usersRef.orderByChild( path: "email").equalTo(email);
                query.addListenerForSingleValueEvent(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot snapshot) {
                        if (snapshot.exists()) {
                            String[] s = new String[6];
                            for (DataSnapshot childSnapshot : snapshot.getChildren()) {
                                // Отримуємо значення DataSnapshot для childSnapshot
                                Object data = childSnapshot.getValue();
                                if (data != null) {
                                    s[0] = ((Map<String, Object>) data).get("name").toString();
                                    s[1] = ((Map<String, Object>) data).get("s_name").toString();
                                    s[2] = ((Map<String, Object>) data).get("home").toString();
                                    s[3] = ((Map<String, Object>) data).get("phon").toString();
                                    s[4] = ((Map<String, Object>) data).get("photo").toString();
                                    s[5] = ((Map<String, Object>) data).get("kilkst").toString();
                                }
                            }
                            listener.onDataReceived(s);
                        }
                    }
```

Рис. 13.1 Метод для отримання даних з БД, та подачі їх у вигляді масиву.

Цей метод getDateFromDatabase отримує дані з бази даних Firebase для користувача з вказаною електронною адресою.

Він використовує об'єкт OnDataReceivedListener для повернення отриманих даних або повідомлення про помилку. Виконується запит до бази даних Firebase для заданого ключа (key), щоб знайти запис з вказаною електронною адресою

(email). Прослуховувач подій addListenerForSingleValueEvent чекає одноразове отримання даних. Якщо дані існують (dataSnapshot.exists()), створюється ще один запит для отримання детальної інформації про користувача з колекції "users". В другому addListenerForSingleValueEvent у методі onDataChange перевіряється, чи існують дані про користувача. Якщо так, витягуються відповідні поля (name, s_name, home, phon, photo, kilkst) та зберігаються в масиві s. Викликається метод listener.onDataReceived(s), який передає масив отриманих даних у слухач. Якщо користувача з вказаною електронною адресою не знайдено, викликається listener.onError("Помилка: Дані не знайдені"). Обробка помилок здійснюється через методи onCancelled у обох ValueEventListener, де викликається listener.onError з повідомленням про помилку.

Цей метод дозволяє отримувати дані користувача з бази даних Firebase за його електронною адресою та передавати їх через слухач OnDataReceivedListener.

```
public void uploadImage(Uri file,String key) {
    if (file!=null){
        ProgressDialog progressDialog =new ProgressDialog(mContext);
        progressDialog.setTitle("Завантаження фото ... ");
        progressDialog.show();
        storageReference.child( pathString: "image").child(key).putFile(file).
                addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {

            @Override
            public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
                progressDialog.dismiss();
                Toast.makeText(mContext, text: "Фото завантажене",Toast.LENGTH_SHORT).show();

            }
        });
    }
}
```

Рис. 13.2 Метод для завантаження фото на хмару.

Цей метод uploadImage завантажує зображення до Firebase Storage. Перевіряє, чи URI файлу не є порожнім (file != null).

Даний метод створює ProgressDialog, щоб показати користувачеві індикатор завантаження з заголовком "Завантаження фото ...". Показує діалогове вікно прогресу завантаження (progressDialog.show()). Використовує storageReference для завантаження файлу в Firebase Storage, у папку "image" з ідентифікатором key. Додає слухач події успішного завантаження за допомогою addOnSuccessListener: Коли завантаження успішне (onSuccess), приховує діалогове вікно прогресу (progressDialog.dismiss()). Показує повідомлення користувачеві про успішне завантаження за допомогою Toast.

Отже, цей метод забезпечує завантаження зображення в Firebase Storage і відображення прогресу користувачеві.

```
public interface OnPhotoDownloadedListener {
    1 usage   2 implementations
    void onPhotoDownloaded(Uri uri);
}

2 usages
public void getPhotoFromDatabase(String s, final OnPhotoDownloadedListener listener) {
    storageReference.child( pathString: "image").child(s).getDownloadUrl().
            addOnSuccessListener(new OnSuccessListener<Uri>() {
        @Override
        public void onSuccess(Uri uri) {
            if (listener != null) {

                listener.onPhotoDownloaded(uri);

            }
        }
    });
}
```

Рис. 13.3 Метод для завантаження фото з хмари.

Цей метод getPhotoFromDatabase отримує URL-адресу завантаженого зображення з Firebase Storage.

Даний метод використовує storageReference для доступу до папки "image" і підпапки з назвою s. Викликає метод getDownloadUrl(), щоб отримати URL-адресу завантаженого файлу. Додає слухача події успішного отримання URL-адреси за допомогою addOnSuccessListener: Коли URL успішно отримано (onSuccess), перевіряє, чи не є слухач listener порожнім. Викликає метод listener.onPhotoDownloaded(uri), щоб передати отриману URL-адресу слухачу.

Цей метод забезпечує отримання URL-адреси завантаженого зображення з Firebase Storage і передачу цієї адреси через слухач OnPhotoDownloadedListener.

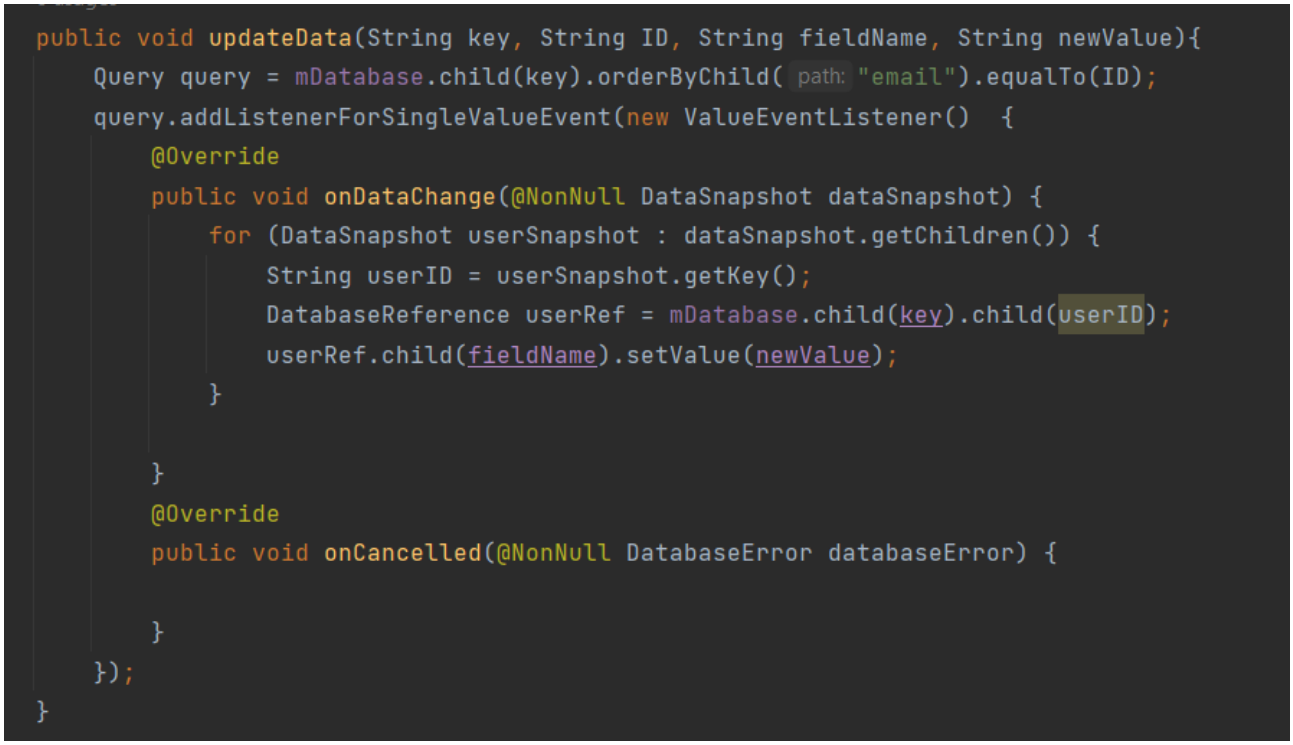



Рис. 13.4 Метод для оновлення даних у БД.

Цей метод updateData оновлює конкретне поле в базі даних Firebase для користувача з вказаною електронною адресою.

Даний метод відправляє запит до бази даних Firebase для заданого ключа (key), щоб знайти запис з вказаною електронною адресою (ID). Використовує addListenerForSingleValueEvent, щоб одноразово прослухати отримання даних. В методі onDataChange: Перебирає всі знайдені записи користувачів (for (DataSnapshot userSnapshot : dataSnapshot.getChildren())). Отримує ключ користувача (userID = userSnapshot.getKey()). Створює посилання на запис користувача (userRef = mDatabase.child(key).child(userID)). Оновлює значення конкретного поля (fieldName) на нове значення (newValue) за допомогою userRef.child(fieldName).setValue(newValue). Метод onCancelled обробляє можливі помилки.

Отже, цей метод дозволяє оновлювати конкретне поле в базі даних Firebase для користувача, визначеного його електронною адресою.

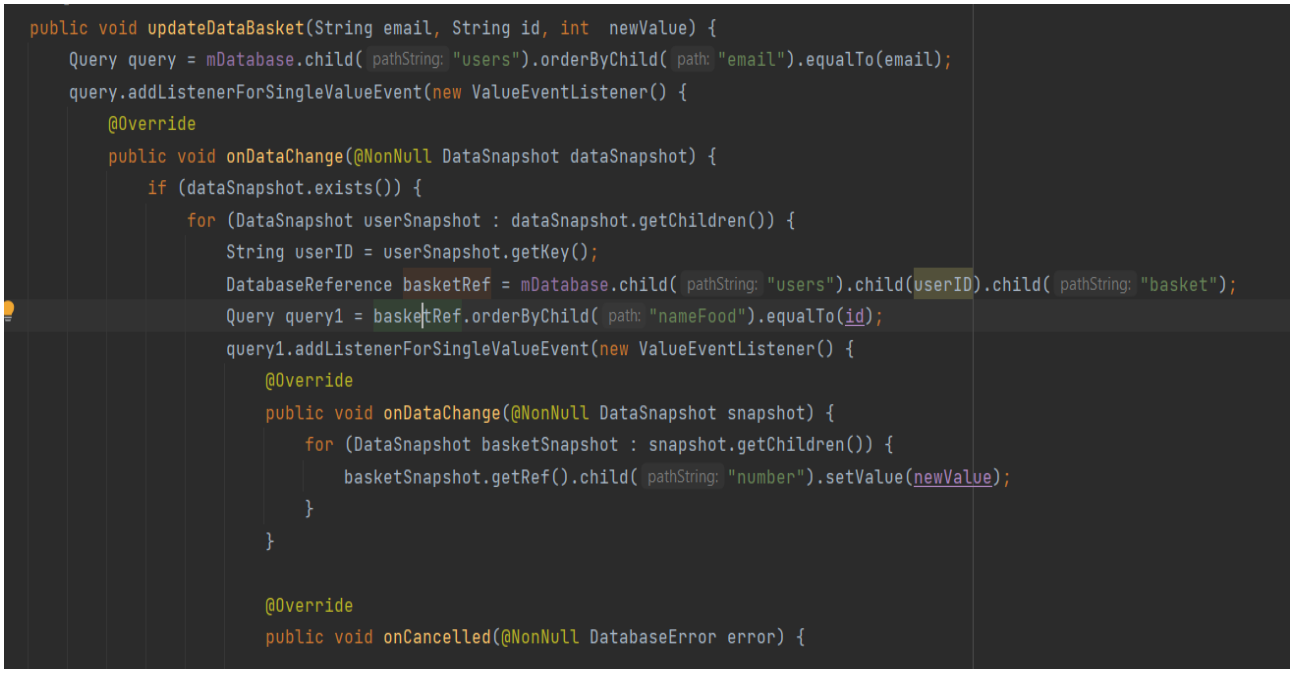



Рис. 13.5 Метод для оновлення даних про кількість товару у корзині користувача.

Цей метод updateDataBasket оновлює кількість товарів у кошику для користувача з вказаною електронною адресою в базі даних Firebase.

Даний метод надсилає запит до бази даних Firebase для колекції "users", щоб знайти користувача з вказаною електронною адресою (email). Використовує addListenerForSingleValueEvent, щоб одноразово прослухати отримання даних. В методі onDataChange перевіряє, чи існують дані користувача (if (dataSnapshot.exists())). Перебирає всі знайдені записи користувачів (for (DataSnapshot userSnapshot : dataSnapshot.getChildren())). Отримує ключ користувача (userID = userSnapshot.getKey()). Створює посилання на запис кошика користувача (basketRef = mDatabase.child("users").child(userID).child("basket")). Виконує запит до кошика користувача для знаходження товару за його ідентифікатором (id) через orderByChild("nameFood").equalTo(id).

Використовує addListenerForSingleValueEvent для одноразового прослуховування отримання даних кошика. В методі onDataChange перебирає всі знайдені записи товарів у кошику (for (DataSnapshot basketSnapshot : snapshot.getChildren())). Оновлює значення кількості товару (number) на нове значення (newValue) за допомогою basketSnapshot.getRef().child("number").setValue(newValue). Метод onCancelled в обох ValueEventListener обробляє можливі помилки.

Отже, цей метод дозволяє оновлювати кількість конкретного товару в кошику користувача в базі даних Firebase, використовуючи електронну адресу користувача та ідентифікатор товару.

```
public static void addBadgeToImageView(ImageView imageView, int badgeNumber) {
    BitmapDrawable bitmapDrawable = (BitmapDrawable) imageView.getDrawable();
    Bitmap bitmap = bitmapDrawable.getBitmap().copy(Bitmap.Config.ARGB_8888, isMutable: true);

    Canvas canvas = new Canvas(bitmap);

    Paint paint = new Paint();
    paint.setColor(Color.RED);
    paint.setStyle(Paint.Style.FILL);
    int circleRadius = 16;
    int circleMargin = 8;
    int circleX = 40;
    int circleY = 45;
    canvas.drawCircle(circleX, circleY, circleRadius, paint);

    Paint textPaint = new Paint();
    textPaint.setColor(Color.WHITE);

    if(badgeNumber>9){
        if(badgeNumber>99) badgeNumber=99;
        textPaint.setTextSize(17);
        canvas.drawText(String.valueOf(badgeNumber), x: 30, y: 50, textPaint);}
   else  {
        textPaint.setTextSize(25);
       canvas.drawText(String.valueOf(badgeNumber), x: 33, y: 54, textPaint);}

    imageView.setImageBitmap(bitmap);
}
```

Рис. 13.6 Метод для додавання кількості товару на іконку корзини в меню-барі.

Цей метод addBadgeToImageView додає значок з числом до зображення в ImageView. Даний метод отримує зображення з ImageView як BitmapDrawable і створює копію його Bitmap, яка підтримує редагування (bitmapDrawable.getBitmap().copy(Bitmap.Config.ARGB_8888, true)). Створює Canvas для малювання на скопійованому Bitmap. Налаштовує Paint для малювання червоного кола: Встановлює колір RED. Встановлює стиль FILL. Малює коло з центром у точці (circleX, circleY) і радіусом circleRadius. Налаштовує Paint для малювання білого тексту: Встановлює колір WHITE. В залежності від значення badgeNumber: Якщо badgeNumber більше 99, обмежує значення до 99 і встановлює розмір тексту 17. Якщо badgeNumber більше 9, встановлює розмір тексту 17. В іншому випадку встановлює розмір тексту 25. Малює текст на Canvas у відповідній позиції. Оновлює ImageView з новим Bitmap, який тепер містить значок з числом.

Цей метод додає значок з числом до зображення в ImageView, відображаючи кількість у червоному колі на зображенні.

```
public void addObjectToBasket(String name_food, String email,ImageView imageView) {
    DatabaseReference usersRef = mDatabase.child( pathString: "users");
    Query query = usersRef.orderByChild( path: "email").equalTo(email);
    query.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for (DataSnapshot userSnapshot : dataSnapshot.getChildren()) {
                String userKey = userSnapshot.getKey();
                User user = userSnapshot.getValue(User.class);
                if (user != null) {
                    DatabaseReference basketRef = usersRef.child(userKey).child( pathString: "basket");
```

```
  // Перевіряємо, чи "basket" вже існує для користувача
  if (user.getBasket() == null) {
      // Якщо "basketClass" відсутнє, створюємо новий об'єкт Basket_Class
      Basket_Class basketClass = new Basket_Class();
      basketClass.addFood(name_food,  quantity: 1); // Додаємо новий запис до "basketClass"
      basketRef.push().setValue(basketClass.toMap());
  } else {
      // Якщо "basket" вже присутнє, оновлюємо його значення
      boolean foodExists = false;

      for (DataSnapshot basketSnapshot : userSnapshot.child( path: "basket").getChildren()) {
          Basket_Class basketClassItem = basketSnapshot.getValue(Basket_Class.class);
          if (basketClassItem != null && basketClassItem.getNameFood().equals(name_food)) {
              // Якщо "name_food" вже існує в "basket", оновлюємо "number"
              basketSnapshot.child( path: "number").getRef().setValue(basketClassItem.getNumber() + 1);
              foodExists = true;
              break;
          }
      }
      if (!foodExists) {
          Basket_Class newBasketItemClass = new Basket_Class();
          newBasketItemClass.addFood(name_food,  quantity: 1);
          basketRef.push().setValue(newBasketItemClass.toMap());
      }
  }
  updateNumber(email,imageView);
```

Рис. 13.7 Метод для додавання конкретного товару в корзину користувача.

Цей метод addObjectToBasket додає об'єкт до кошика користувача в базі даних Firebase і оновлює відповідний значок на зображенні. Даний метод отримує посилання на колекцію "users" в базі даних (usersRef = mDatabase.child("users")). Виконує запит до бази даних для знаходження користувача з вказаною електронною адресою (email) через orderByChild("email").equalTo(email). Використовує addListenerForSingleValueEvent для одноразового прослуховування отримання даних. В методі onDataChange: Перебирає всі знайдені записи користувачів (for (DataSnapshot userSnapshot : dataSnapshot.getChildren())). Отримує ключ користувача (userKey = userSnapshot.getKey()). Перетворює запис користувача в об'єкт User (user = userSnapshot.getValue(User.class)). Якщо об'єкт User не є порожнім: Отримує посилання на кошик користувача (basketRef = usersRef.child(userKey).child("basket")). Перевіряє, чи кошик існує для

користувача: Якщо кошик відсутній (user.getBasket() == null), створює новий об'єкт Basket_Class, додає товар з кількістю 1 та зберігає його в базі даних. Якщо кошик вже існує, перевіряє, чи товар з вказаною назвою (name_food) вже є в кошику: Якщо товар існує, оновлює його кількість, збільшуючи на 1. Якщо товар не існує, створює новий запис товару з кількістю 1 та додає його до кошика. Викликає метод updateNumber, щоб оновити значок на зображенні з врахуванням змін у кошику. Метод onCancelled обробляє можливі помилки.

Цей метод дозволяє додавати товари до кошика користувача в базі даних Firebase, оновлювати кількість існуючих товарів та відображати зміни на зображені.

```
public void updateNumber(String email, ImageView imageView){
    int[] numberBasket = {0};
    int[] numberData = new int[1];
    getDateFromDatabase( key: "users", email, new OnDataReceivedListener() {
        1 usage
        @Override
        public void onDataReceived(String[] data) {
            if (data[5]!=null){
            numberData[0] = Integer.parseInt(data[5]);}
        }

        3 usages
        @Override
        public void onError(String errorMessage) {

        }
    });
    DatabaseReference usersRef = mDatabase.child( pathString: "users");
    Query query = usersRef.orderByChild( path: "email").equalTo(email);
    query.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            for (DataSnapshot userSnapshot : dataSnapshot.getChildren()) {
                User user = userSnapshot.getValue(User.class);
                if (user != null && user.getBasket() != null) {
                    for (DataSnapshot basketSnapshot : userSnapshot.child( path: "basket").getChildren()) {
                        Basket_Class basketClassItem = basketSnapshot.getValue(Basket_Class.class);
                        if (basketClassItem != null) {
                            numberBasket[0] += basketClassItem.getNumber();
                        }
                    }
                }
            }
        addBadgeToImageView(imageView,numberBasket[0]);
        if(numberData[0]!=numberBasket[0]){
            updateData( key: "users",email, fieldName: "kilkst",String.valueOf(numberBasket[0]));
        }
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {
    }
```

Рис. 13.8 Метод для перевірки коректності кількості товару в кошику та оновлені кількості товару якщо товар видалений користувачем.

Цей метод updateNumber оновлює кількість товарів у кошику користувача та відображає значок з кількістю на зображенні.

Наведений код ініціалізує масиви для збереження кількості товарів у кошику (numberBasket) та кількості товарів, отриманих з бази даних (numberData). Викликає метод getDateFromDatabase для отримання даних користувача з бази даних: В методі onDataReceived зберігає кількість товарів з поля kilkst у масиві numberData. В методі onError обробляє можливі помилки. Виконує запит до бази даних для знаходження користувача з вказаною електронною адресою (email) через orderByChild("email").equalTo(email). Використовує addListenerForSingleValueEvent для одноразового прослуховування отримання даних. В методі onDataChange: Перебирає всі знайдені записи користувачів (for (DataSnapshot userSnapshot : dataSnapshot.getChildren())). Перетворює запис користувача в об'єкт User (user = userSnapshot.getValue(User.class)). Якщо об'єкт User не є порожнім та має кошик (user.getBasket() != null), перебирає всі товари у кошику. Збільшує кількість товарів у кошику (numberBasket[0] += basketClassItem.getNumber()). Викликає метод addBadgeToImageView для додавання значка з кількістю товарів на зображення. Якщо кількість товарів у кошику (numberBasket[0]) відрізняється від кількості товарів у полі kilkst (numberData[0]), оновлює поле kilkst у базі даних на нове значення (updateData("users", email, "kilkst", String.valueOf(numberBasket[0]))). Метод onCancelled обробляє можливі помилки.

Цей метод оновлює кількість товарів у кошику користувача в базі даних Firebase та відображає її на зображенні у вигляді значка.

### 3.5 Замовлення їжі.

На головному екрані, для зручності пошуку, розміщені розділи з різними стравами. Там є фото та короткий опис кожної категорії товару. Після переходу на потрібну категорію, ми бачимо перелік товарів з описом, зображеннями та ціною товару, біля кожного товару є кнопка «додати до кошика», також в кожному вікні є меню-бар з кнопками які дозволяють, переглянути кошик, відомості про доставку, особистий кабінет та повернутися на головну.

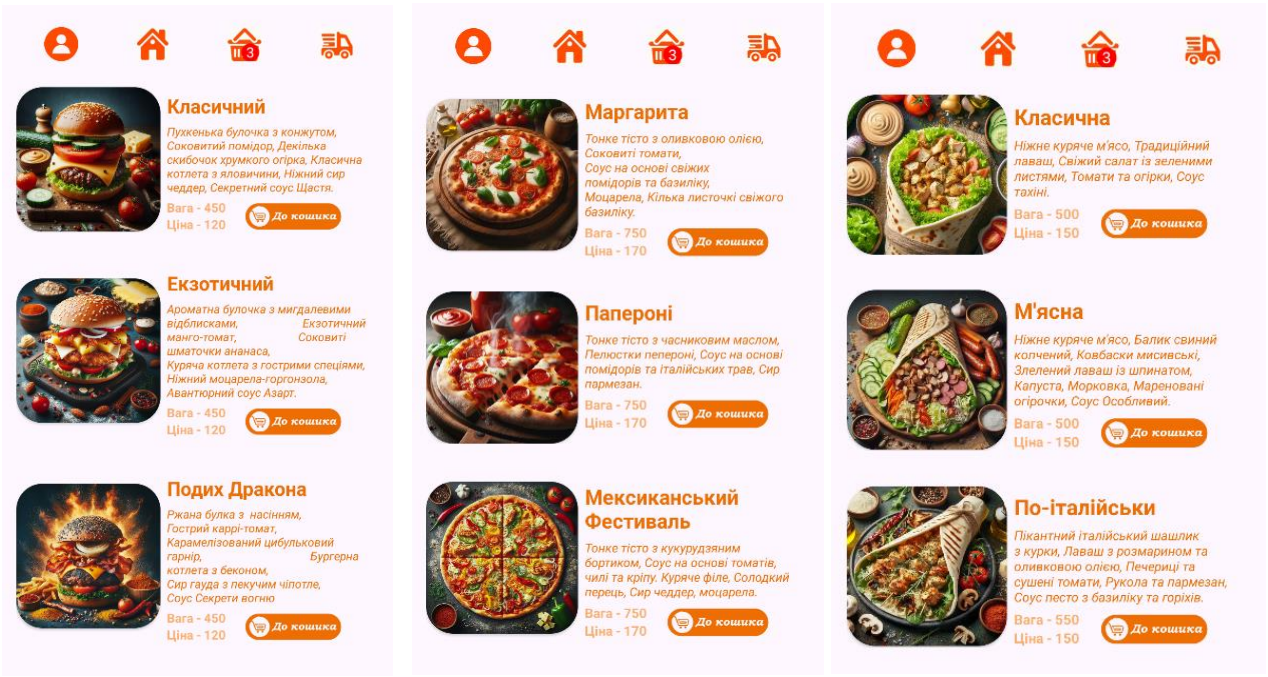



Рис. 14 Вікна для замовлення товару.

Наведені вікна працюють за однаковим алгоритмом, що наведений на рис.15.

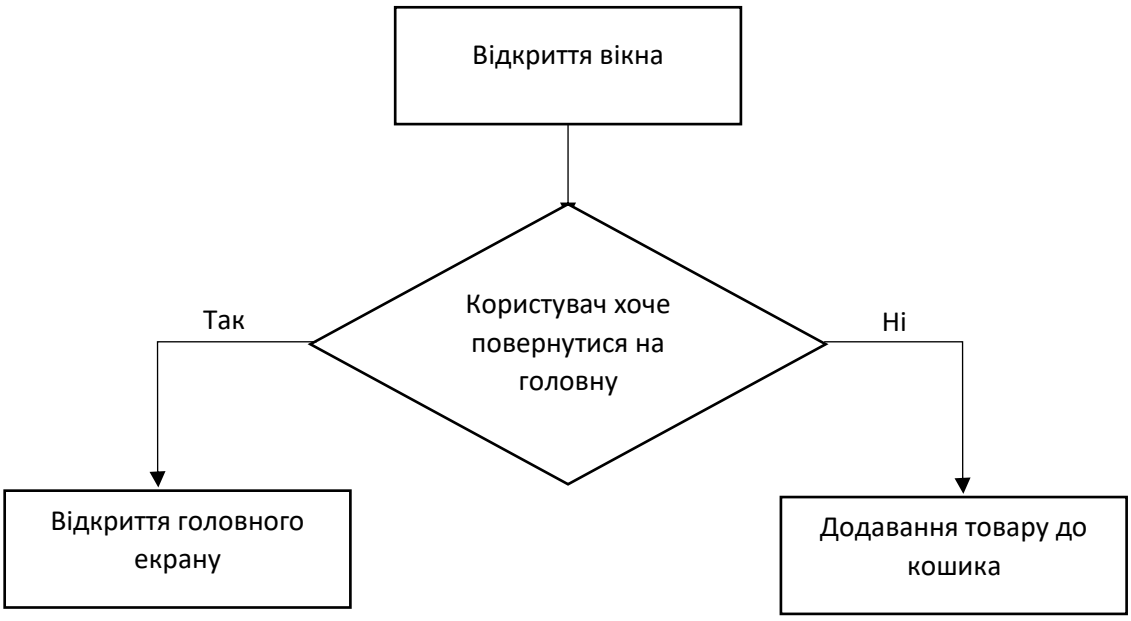



Рис. 15 Алгоритм роботи вікна для додавання товару.

Для реалізації роботи даного алгоритму я написав наступний код рис. 16.1-16.3.

```
@Override
protected void onStart() {
    super.onStart();

    button_addKlasiksavuha.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { helper.addObjectToBasket( name_food: "31",email,basketButton); }
    });
    button_addMeat.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            helper.addObjectToBasket( name_food: "32",email,basketButton);

        }
    });
    button_addItaliano.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            helper.addObjectToBasket( name_food: "33",email,basketButton);

        }
    });
```

Рис. 16.1 Метод для додавання товару в кошик категорії «Шаурма».

```
protected void onStart() {
    super.onStart();
    button_addBurgerPicant.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { helper.addObjectToBasket( name_food: "13", email,basketButton); }
    });
    button_addBurgerClassik.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            helper.addObjectToBasket( name_food: "11", email,basketButton);

        }
    });
    button_addBurgerExotick.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            helper.addObjectToBasket( name_food: "12", email,basketButton);


        }
    });
}
```

Рис. 16.2 Метод для додавання товару в кошик категорії «Бургер».

```
@Override
protected void onStart() {
    super.onStart();

    button_addMargarita.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { helper.addObjectToBasket( name_food: "21",email,basketButton); }
    });
    button_addPaperoni.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { helper.addObjectToBasket( name_food: "22",email,basketButton); }
    });
    button_addMexikan.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) { helper.addObjectToBasket( name_food: "23",email,basketButton); }
    });


}
```

Рис. 16.3 Метод для додавання товару в кошик категорії «Шаурма».

Як видно на скріншоті для виконання операції додавання товару до кошика я використав створений мною клас Helper, та метод цього класу роботу якого було описано вище.

### 3.6 Особистий кабінет користувача.

Після переходу у особистий кабінет через кнопку, що знаходиться в меню-барі, користувачеві відкривається вікно рис. 17. В даному вікні розміщено коротка інформація користувача:

- Прізвище.
- Ім’я.
- Домашня адреса (користувач сам її додає за бажанням, вона буде автоматично вписуватись при новому замовлені, якщо користувач не захоче змінити її в якомусь конкретному замовлені).
- Номер телефону.
- Фото.

Користувач зможе змінювати всю наявну інформацію а також редагувати фото. Для переходу на інший обліковий запис в даному вікні є кнопка виходу з поточного облікового запису.


Рис. 17 Особистий кабінет користувача.

Алгоритм роботи даного вікна наведений у схемі на рис. 18.

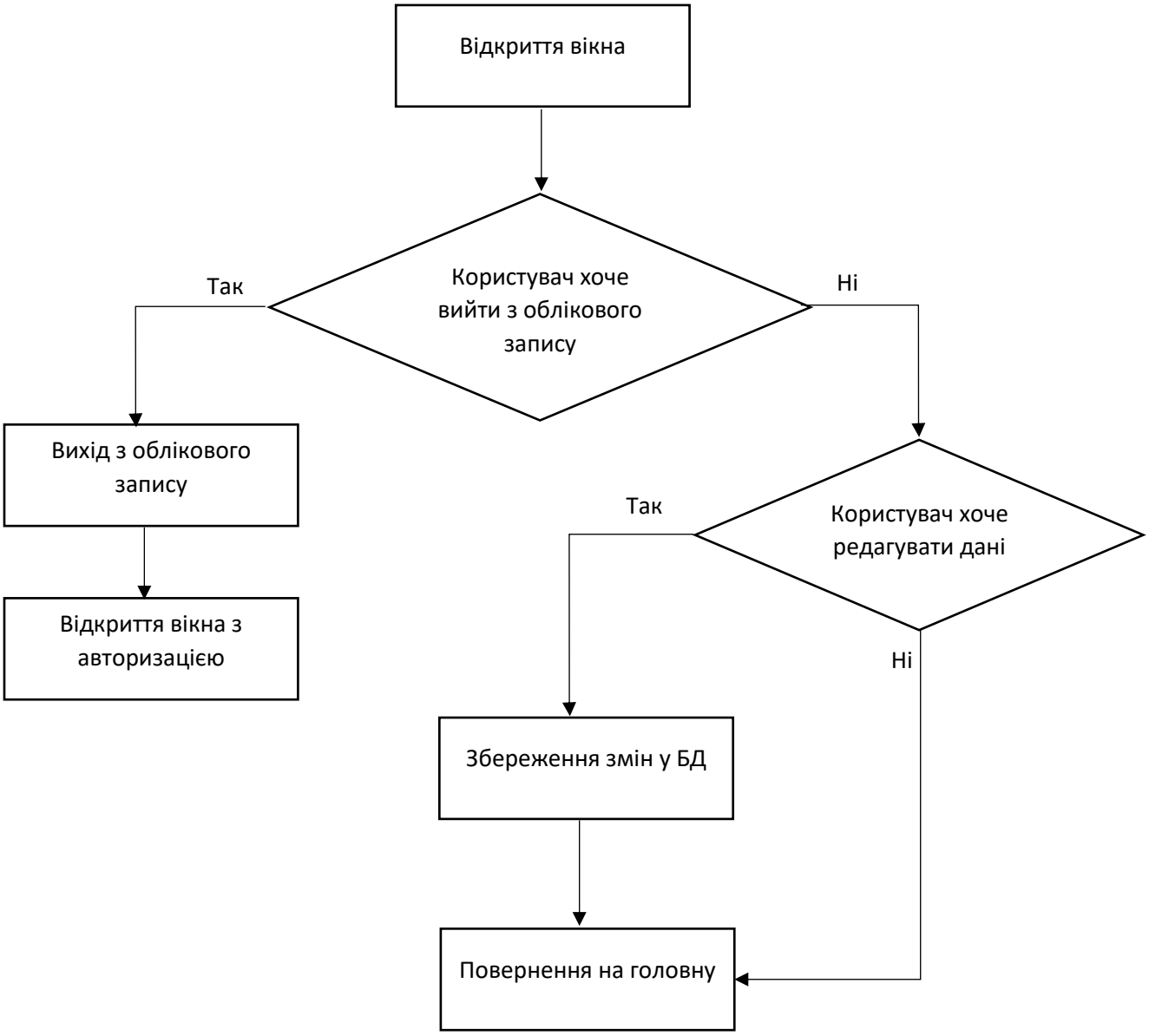



Рис. 18 Алгоритм роботи особистого кабінету користувача.

Для реалізації роботи даного алгоритму я написав наступний код Рис. 19.1 - 19.6.

```
singout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        gsc.signOut().addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                finish();
                startActivity(new Intent( packageContext: Customer_acount.this, MainActivity.class ));
            }
        });
    }
});
```

Рис. 19.1 Метод для виходу з облікового запису.

Цей код встановлює обробник для кнопки singout, яка виконує вихід користувача з облікового запису Google та перехід до головного екрану програми (MainActivity). Метод встановлює обробник кліків для кнопки singout. В обробнику викликає метод signOut() для об'єкта gsc (GoogleSignInClient), що відповідає за вихід користувача з облікового запису Google. Додає слухача (OnCompleteListener), який виконується після завершення виходу з облікового запису. В методі onComplete, який викликається після успішного виходу: Викликає finish() для закриття поточної активності. Запускає новий (Intent) для переходу до MainActivity.

Цей метод забезпечує коректний вихід користувача з облікового запису Google та перехід до головного екрану програми після виходу.

```
1 usage
private void setText(String email){
    String [] strings = new String[5];
    helper.getDateFromDatabase( key: "users",email, new Helper.OnDataReceivedListener() {
        1 usage
        @Override
        public void onDataReceived(String[] data) {
            textView_name.setText(data[0]);
            textView_s_name.setText(data[1]);
            textView_home.setText(data[2]);
            textView_phon.setText(data[3]);
            type_f= Integer.parseInt(data[4]);

        }

        3 usages
        @Override
        public void onError(String errorMessage) {
            Toast.makeText( context: Customer_acount.this,errorMessage,Toast.LENGTH_SHORT).show();
        }
    });

}
```

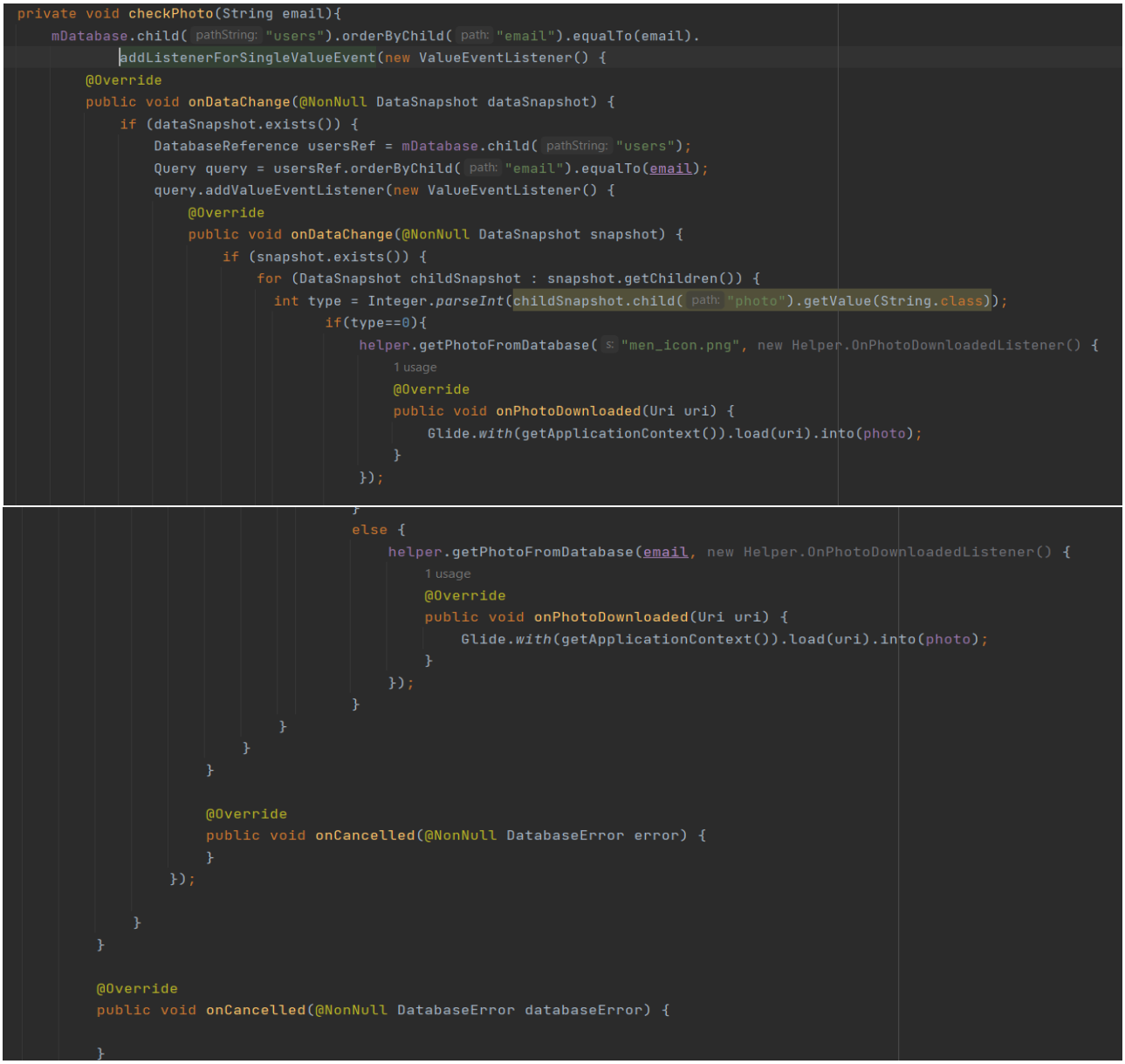
Рис. 19.2 Метод для отримання даних з БД та виведення їх у вікні.

Цей метод setText встановлює текст для різних TextView на основі даних, отриманих з БД. Поданий код створює масив strings розміром 5 для зберігання даних з БД. Викликається метод getDateFromDatabase з об'єкта helper для отримання даних користувача за допомогою електронної адреси email. В методі onDataReceived обробника подій. Далі метод встановлює текст для textView_name з отриманого data[0]. Встановлює текст для textView_s_name з отриманого data[1]. Встановлює текст для textView_home з отриманого data[2]. Встановлює текст для textView_phon з отриманого data[3]. Конвертує значення data[4] в ціле число (type_f). У разі помилки викликається метод onError, який відображає коротке повідомлення про помилку за допомогою Toast.

Цей метод дозволяє отримувати дані з бази даних і автоматично встановлювати їх у відповідні TextView на екрані користувача.

```
    1 usage
    private void editPhoto(){
        Intent intent= new Intent();
        intent.setType("image/+");
        intent.setAction(Intent.ACTION_GET_CONTENT);
        startActivityForResult(Intent.createChooser(intent, title: "Виберіть фото для завантаження ..."),PICK_IMAGE_REQUEST);
    }

@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode==PICK_IMAGE_REQUEST&&resultCode== Activity.RESULT_OK&&data!=null){
        uri_image =data.getData();
        try {
            Bitmap bitmap= MediaStore.Images.Media.getBitmap(getApplicationContext().getContentResolver(),uri_image);
            photo.setImageBitmap(bitmap);

        }catch (IOException e){
                e.printStackTrace();
        }
    }
}
```

Рис. 19.3 Метод для завантаження фото на хмару.



```
private void checkPhoto(String email){
    mDatabase.child( pathString: "users").orderByChild( path: "email").equalTo(email).
            addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()) {
                DatabaseReference usersRef = mDatabase.child( pathString: "users");
                Query query = usersRef.orderByChild( path: "email").equalTo(email);
                query.addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot snapshot) {
                        if (snapshot.exists()) {
                            for (DataSnapshot childSnapshot : snapshot.getChildren()) {
                              int type = Integer.parseInt(childSnapshot.child( path: "photo").getValue(String.class));
                                    if(type==0){
                                        helper.getPhotoFromDatabase( s: "men_icon.png", new Helper.OnPhotoDownloadedListener() {
                                            1 usage
                                            @Override
                                            public void onPhotoDownloaded(Uri uri) {
                                                Glide.with(getApplicationContext()).load(uri).into(photo);
                                            }
                                        });
                                    }
                                    else {
                                        helper.getPhotoFromDatabase(email, new Helper.OnPhotoDownloadedListener() {
                                            1 usage
                                            @Override
                                            public void onPhotoDownloaded(Uri uri) {
                                                Glide.with(getApplicationContext()).load(uri).into(photo);
                                            }
                                        });
                                    }
                            }
                        }
                    }

                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {
                    }
                });

            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError databaseError) {

        }
```

Рис. 19.4 Метод для перевірки наявності фото користувача та завантажені його з хмари.

Цей метод checkPhoto перевіряє, чи є фотографія користувача в базі даних і встановлює її на відповідному інтерфейсі за допомогою бібліотеки Glide. Ось як він працює - виконує запит до бази даних Firebase для знаходження користувача з вказаною електронною адресою (email) через orderByChild("email").equalTo(email). Викликає addListenerForSingleValueEvent, щоб одноразово прослухати результат запиту. У методі onDataChange: Перевіряє, чи існує хоча б один запис користувача за допомогою dataSnapshot.exists(). Якщо запис існує, створює посилання usersRef на розділ "users" у базі даних. Виконує додатковий запит (query.addValueEventListener), щоб отримати детальнішу інформацію про користувача. У методі onDataChange внутрішнього ValueEventListener: Перевіряє, чи існує хоча б один дочірній елемент (snapshot.exists()). Перебирає всі дочірні елементи користувача для отримання значення photo. Конвертує значення photo в ціле число type. Якщо type дорівнює 0, викликає метод getPhotoFromDatabase зображення

"men_icon.png"(дане фото є стандартним для всіх нових користувачів) через helper і встановлює його за допомогою Glide у photo. Якщо type не дорівнює 0, викликає метод getPhotoFromDatabase через helper, використовуючи email як ключ, і встановлює отримане зображення у photo.

Цей метод дозволяє динамічно завантажувати та встановлювати фотографію користувача з бази даних у відповідному інтерфейсі за допомогою Glide, в залежності від наявності і типу фотографії .

```
1 usage
private void save(){
    if(uri_image!=null){
        helper.uploadImage(uri_image,email);
        helper.updateData( key: "users",email, fieldName: "photo", newValue: "1");
    }
    helper.updateData( key: "users",email, fieldName: "home",String.valueOf(textView_home.getText()));
    helper.updateData( key: "users",email, fieldName: "phon",String.valueOf(textView_phon.getText()));
    helper.updateData( key: "users",email, fieldName: "name",String.valueOf(textView_name.getText()));
    helper.updateData( key: "users",email, fieldName: "s_name",String.valueOf(textView_s_name.getText()));

    cancel();
}
2 usages
```

Рис. 19.5 Метод для збереження змін в даних користувача.

Цей метод save зберігає зміни в базі даних користувача. Ось як він працює- Перевіряє, чи uri_image не є порожнім. Якщо uri_image не є порожнім, викликає метод uploadImage об'єкта helper для завантаження зображення за вказаною uri_image та email. Оновлює дані користувача в базі даних через метод updateData об'єкта helper для таких полів: "home" зі значенням, отриманим з textView_home. "phon" зі значенням, отриманим з textView_phon. "name" зі значенням, отриманим з textView_name. "s_name" зі значенням, отриманим з textView_s_name. "photo" оновлюється на значення "1", щоб позначити, що фотографія користувача тепер існує в базі даних. Викликає метод cancel, який слугує для виходу з поточного вікна.

Отже, метод save використовується для збереження змін у профілі користувача, включаючи завантаження фотографії (якщо вона вказана) та оновлення інших особистих даних.

```
2 usages
private void cancel(){
    Intent intent = new Intent( packageContext: Customer_acount.this,Customer.class);
    finish();
    startActivity(intent);
}
```

Рис. 19.6 Метод для виходу з даного вікна без змін.

Цей метод cancel створює новий об'єкт Intent для переходу до активності Customer з поточного контексту Customer_acount.this. Викликає метод finish(), що завершує поточну активність Customer_acount. Запускає нову активність за допомогою методу startActivity(intent), яка відображає створений Intent, переходячи до активності Customer. Отже, метод cancel використовується для завершення поточної активності та переходу до іншої активності у додатку.

### 3.7 Кощик.

Дане вікно призначене для відображення кількості товарів що обрав користувач рис. 20. Перелік продуктів, доданих до корзини, кожен з яких відображається з назвою, кількістю, ціною та зображенням. Також в даному вікні передбачена можливість збільшення або зменшення кількості кожного товару. Для оформлення замовлення варто в даному вікні вказати адресу доставки та натиснути на кнопку «Замовити». Для переходу на інші вікна додатку вгорі розміщується меню-бар з усіма необхідними кнопками.

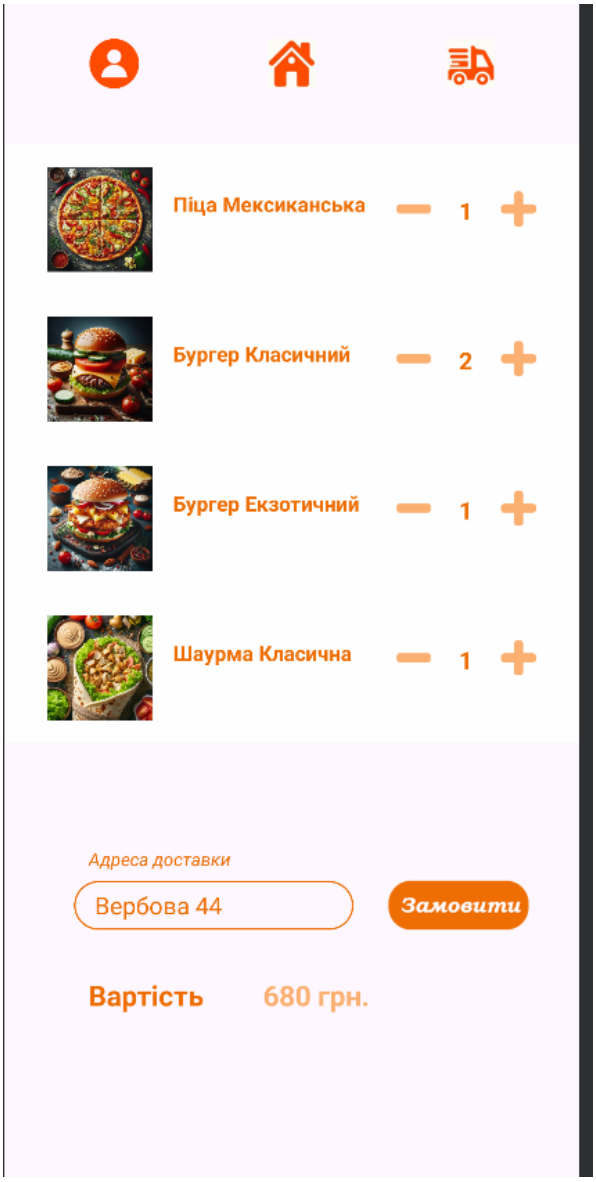



Рис. 20 Кошик.

Алгоритм роботи  даного вікна наведений на рис. 21.

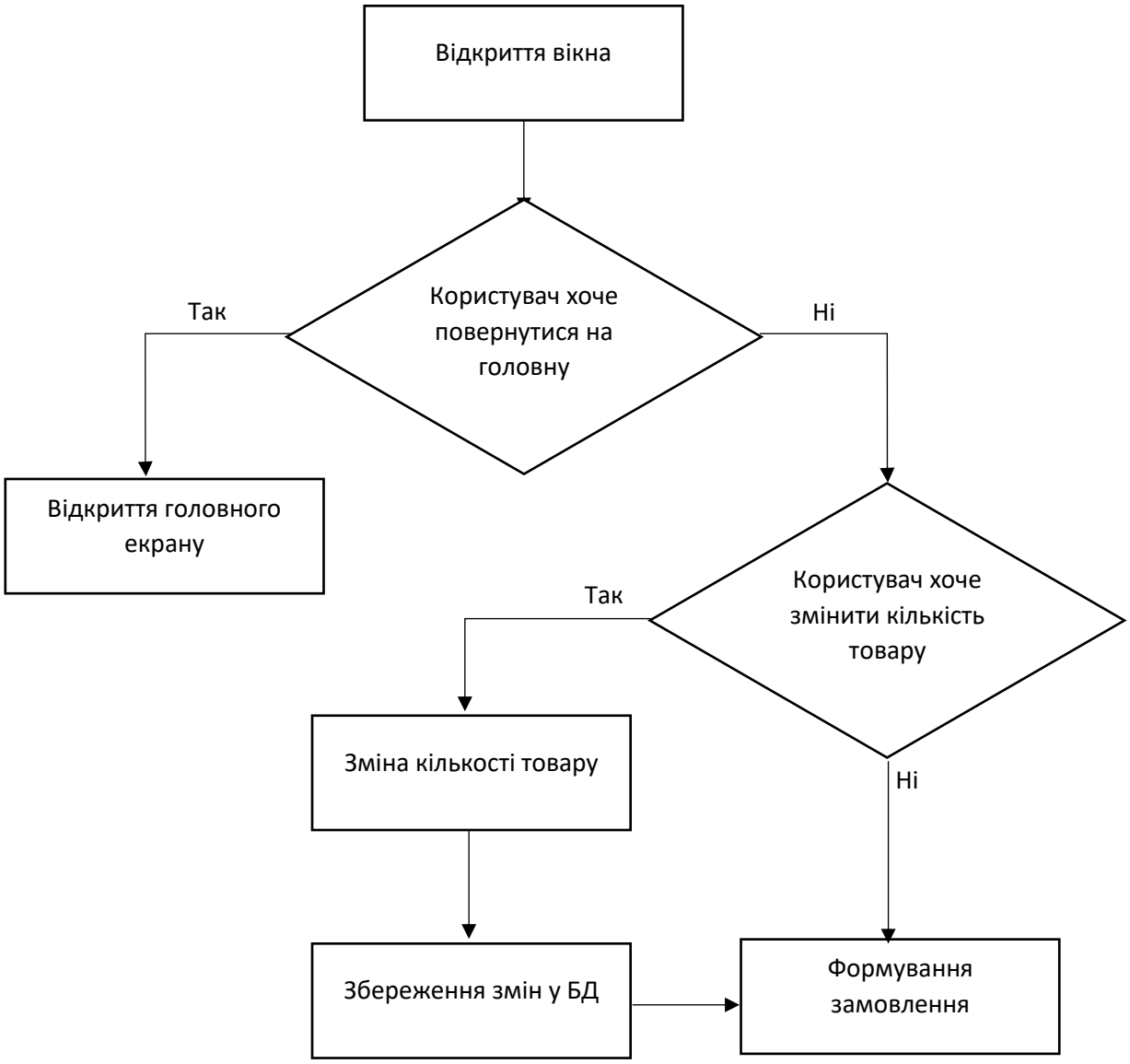



Рис. 21 Алгоритм роботи кошика.

Для реалізації роботи даного алгоритму я написав наступний код рис. 22.1-22.4

```
private void listenerData(String email) {
    DatabaseReference usersRef = mDatabase.child( pathString: "users");
    Query query = usersRef.orderByChild( path: "email").equalTo(email);
    query.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            bascetDataList.clear();
            coust=0;
            if (snapshot.exists()) {
                DataSnapshot userSnapshot = snapshot.getChildren().iterator().next();
                if (userSnapshot.hasChild( path: "basket")) {
                    for (DataSnapshot basket : userSnapshot.child( path: "basket").getChildren()) {
                        String text1 = basket.child( path: "nameFood").getValue(String.class);
                        Long number = basket.child( path: "number").getValue(Long.class);

                        if(Integer.parseInt(text1)>30){
                            coust= (int) (coust+150*number);

                        }
                        else if(Integer.parseInt(text1)<20){
                            coust= (int) (coust+120*number);
                        }
                        else coust= (int) (coust+170*number);
```

```
                if (text1 != null && number != null && number > 0) {
                    BascetData bascetData = new BascetData(text1, String.valueOf(number));
                    bascetDataList.add(bascetData);
                    String itemData = text1 + ", " + number.toString(); // Створюємо рядок з ім'ям товару та його кількістю
                    newStrainList.add(itemData);
                } else {
                    // Видаляємо запис про товар з кількістю, меншою або рівною 0
                    basket.getRef().removeValue();
                }
            }
        }
        basketAdapter.notifyDataSetChanged();
        if (!bascetDataList.isEmpty()) {
            recyclerView.scrollToPosition(bascetDataList.size() - 1);
            textViewcoust.setText(coust+" грн.");
        }
    }
}

@Override
public void onCancelled(@NonNull DatabaseError error) {
}
});
```

Рис. 22.1 Метод, який переглядає БД та виводить всі товари що додав користувач.

Цей метод listenerData слугує для отримання даних про товари, що обрав клієнт. Він встановлює посилання на вузол користувачів у базі даних і створює запит query, який шукає користувача з вказаною електронною адресою (email). Додає ValueEventListener, який відслідковує зміни даних у вузлі, відповідному

зазначеному query. У методі onDataChange: Очищує список bascetDataList, який зберігає дані корзини покупок. Ініціалізує змінну coust (зміна для підрахунку загальної вартості всього товару) і обнуляє її. Перевіряє, чи існує користувач з такою електронною адресою в базі даних. Отримує перший дочірній елемент DataSnapshot, щоб отримати доступ до даних користувача. Перевіряє, чи існує у користувача поле "basket" (кошик з товарами). Перебирає всі елементи "basket": Отримує назву товару (text1) і його кількість (number). Розраховує вартість кожного товару залежно від його категорії. Додає дані товару до списку bascetDataList і створює рядок для нового списку newStrainList. Якщо кількість товару менша або дорівнює 0, видаляє запис про цей товар з бази даних. Оновлює адаптер basketAdapter для відображення оновлених даних у списку RecyclerView. Якщо список bascetDataList не порожній, прокручує список до останнього елемента і встановлює відповідну вартість товарів (coust) у текстовому полі textViewcoust.

Отже, метод listenerData відстежує та оновлює дані користувача про його кошик покупок у реальному часі, використовуючи базу даних Firebase.

```
public void createOrder(LatLng address, List<String> basketDataList) {
    DatabaseReference ordersRef = mDatabase.child( pathString: "orders").push();

    Map<String, Object> orderData = new HashMap<>();
    orderData.put( k: "emailcustomer", email);
    orderData.put( k: "phonCur",  v: 0);
    orderData.put( k: "adress", textViewAdress.getText().toString());
    orderData.put( k: "coust", coust);
    orderData.put( k: "type",  v: 1);
    orderData.put( k: "latlngAddress", address);
    orderData.put( k: "customerData", basketDataList); // Додаємо список у замовлення

    ordersRef.updateChildren(orderData);
}
```

Рис. 22.2 Метод, який створює новий запис у БД з замовленням користувача.

Цей метод createOrder призначений для створення нового замовлення у базі даних Firebase. Наведений код спочатку отримує посилання на вузол "orders": Створюється посилання ordersRef, яке вказує на новий унікальний ідентифікатор у вузлі "orders" у базі даних Firebase.

Створюється об'єкт orderData, який містить всю необхідну інформацію для замовлення: "emailcustomer" - Електронна адреса користувача, яка вже збережена в змінній email. "adress" - Адреса доставки, яка береться з текстового поля textViewAdress. "coust" - Загальна вартість замовлення (coust), яка також передається. "type" - Тип замовлення (це слугує для відслідковування актуальності замовлення, адже якщо користувач скасує замовлення дані про це замовлення мають бути збережені а тип замінений на 0). "latlngAddress": Географічні координати адреси доставки (об'єкт типу LatLng). "customerData": Список товарів у замовленні (basketDataList), який передається у вигляді списку рядків. Оновлення даних в базі даних: Здійснюється оновлення у вузлі orders за допомогою методу updateChildren, який додає новий запис з усією інформацією про замовлення.

Виводиться повідомлення за допомогою Toast про успішне створення замовлення.

Отже, метод createOrder є ключовим для додавання нових замовлень у базу даних із збереженням всієї необхідної інформації.

```
private void setText(String email){
    String [] strings = new String[5];
    helper.getDateFromDatabase( key: "users",email, new Helper.OnDataReceivedListener() {
        1 usage
        @Override
        public void onDataReceived(String[] data) { textViewAdress.setText(data[2]); }

        3 usages
        @Override
        public void onError(String errorMessage) {
            Toast.makeText( context: Basket.this,errorMessage,Toast.LENGTH_SHORT).show();
        }
    });

}
```

Рис. 22.3 Метод для отримання адресу доставки з БД.

Цей метод setText встановлює текстове значення для textViewAdress на основі даних, отриманих з бази даних. Спочатку відбувається ініціалізація масиву і виклик методу getDateFromDatabase, далі створюється масив strings з п'ятьма елементами для збереження даних. Викликається метод

getDateFromDatabase з об'єкта helper, щоб отримати дані з бази даних для користувача з вказаною електронною адресою (email).

У методі onDataReceived інтерфейсу Helper.OnDataReceivedListener встановлюється текст для textViewAdress з отриманих даних під індексом 2 (це відповідає третьому елементу у масиві data).

У методі onError інтерфейсу Helper.OnDataReceivedListener виводиться коротке сповіщення про помилку через Toast, що містить текст помилки.

Отже, цей метод забезпечує завантаження адреси користувача з бази даних і встановлює її в текстове поле textViewAdress.

```
private  LatLng getLatLng(String adress){
    LatLng latLngAdress=null;
    String cityName = "Рівне";
    String streetName = adress;
    Geocoder geocoder = new Geocoder( context: Basket.this, Locale.getDefault());
    if(!streetName.isEmpty()){
        try {
            List<Address> cityAddresses = geocoder.getFromLocationName(cityName,  maxResults: 1);
            if (!cityAddresses.isEmpty()) {
                Address cityAddress = cityAddresses.get(0);
                double cityLatitude = cityAddress.getLatitude();
                double cityLongitude = cityAddress.getLongitude();
                double searchRadiusKm = 50.0;
                List<Address> addresses = geocoder.getFromLocationName(streetName,  maxResults: 1,
                        lowerLeftLatitude: cityLatitude - (searchRadiusKm / 111.32),  lowerLeftLongitude: cityLongitude - (searchRadiusKm / (111.32 * Math.cos(cityLatitude))),
                        upperRightLatitude: cityLatitude + (searchRadiusKm / 111.32),  upperRightLongitude: cityLongitude + (searchRadiusKm / (111.32 * Math.cos(cityLatitude))));
                if (!addresses.isEmpty()) {
                    Address address = addresses.get(0);
                    double latitude = address.getLatitude();
                    double longitude = address.getLongitude();
                    LatLng latLng = new LatLng(latitude, longitude);
                    latLngAdress=latLng;
                } else {
                    Toast.makeText( context: Basket.this,  text: "Невірна адреса ",Toast.LENGTH_LONG).show();
                }
            } else {
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Рис. 22.4 Метод для перевірки коректності вказаного адресу та переведення його у координати на карті.

Цей метод приймає адресу (address) і повертає координати цієї адреси (об'єкт LatLng). Він використовує Geocoder для пошуку координат міста "Рівне" і введеної адреси в межах 50 км від цього міста. Якщо адресу знайдено, метод повертає координати, інакше виводиться повідомлення про помилку або про необхідність вказати початкову точку маршруту.

### 3.8Вікно доставки товару.

Дане вікно призначене для відображення всіх замовлень, що на разі є актуальними для даного користувача рис. 23. Для інформативності кожне замовлення містить інформацію про вартість замовлення та адресу доставки. Користувач може зв’язатися з кур’єром для цього біля кожного замовлення розміщені кнопки з телефоном та можливістю почати чат з кур’єром. Також передбачена можливість скасувати замовлення клієнтом. На даному вікні вгорі також розміщений меню-бар для можливості переходу на інші розділи додатку.


Рис. 23 Вікно доставки товару.

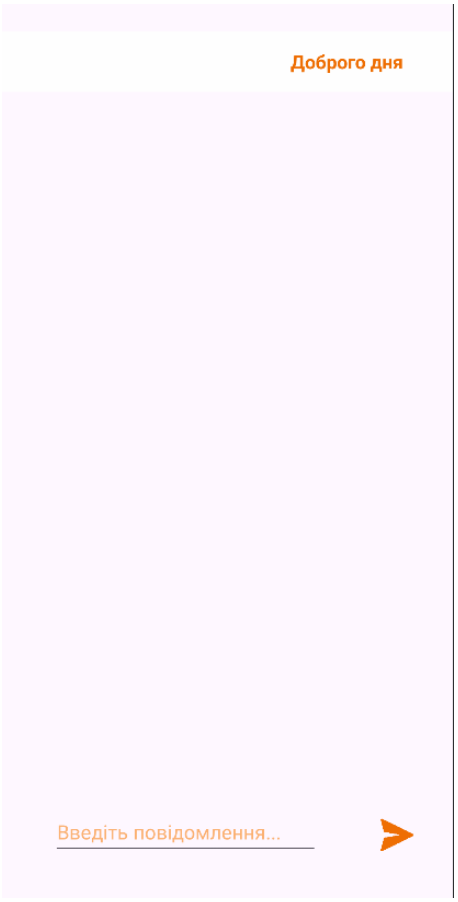



Рис. 24 Екран чату.

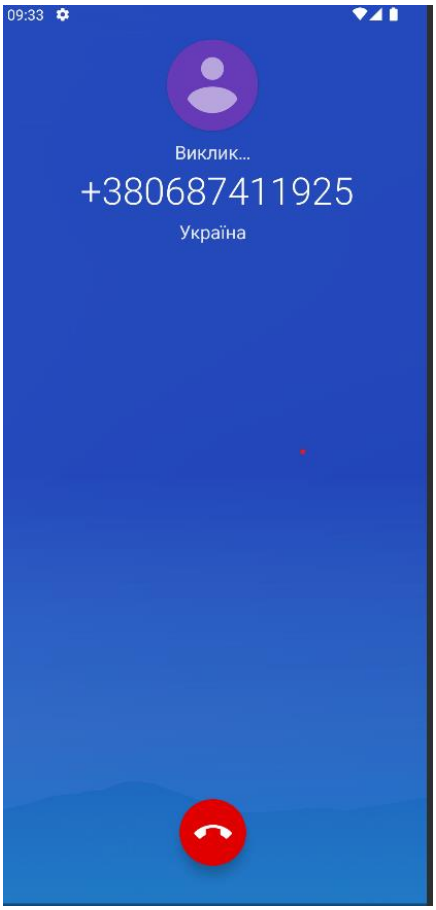



Рис. 25 Екран виклику кур’єра.

Алгоритм роботи даного вікна наведений на рис. 26.

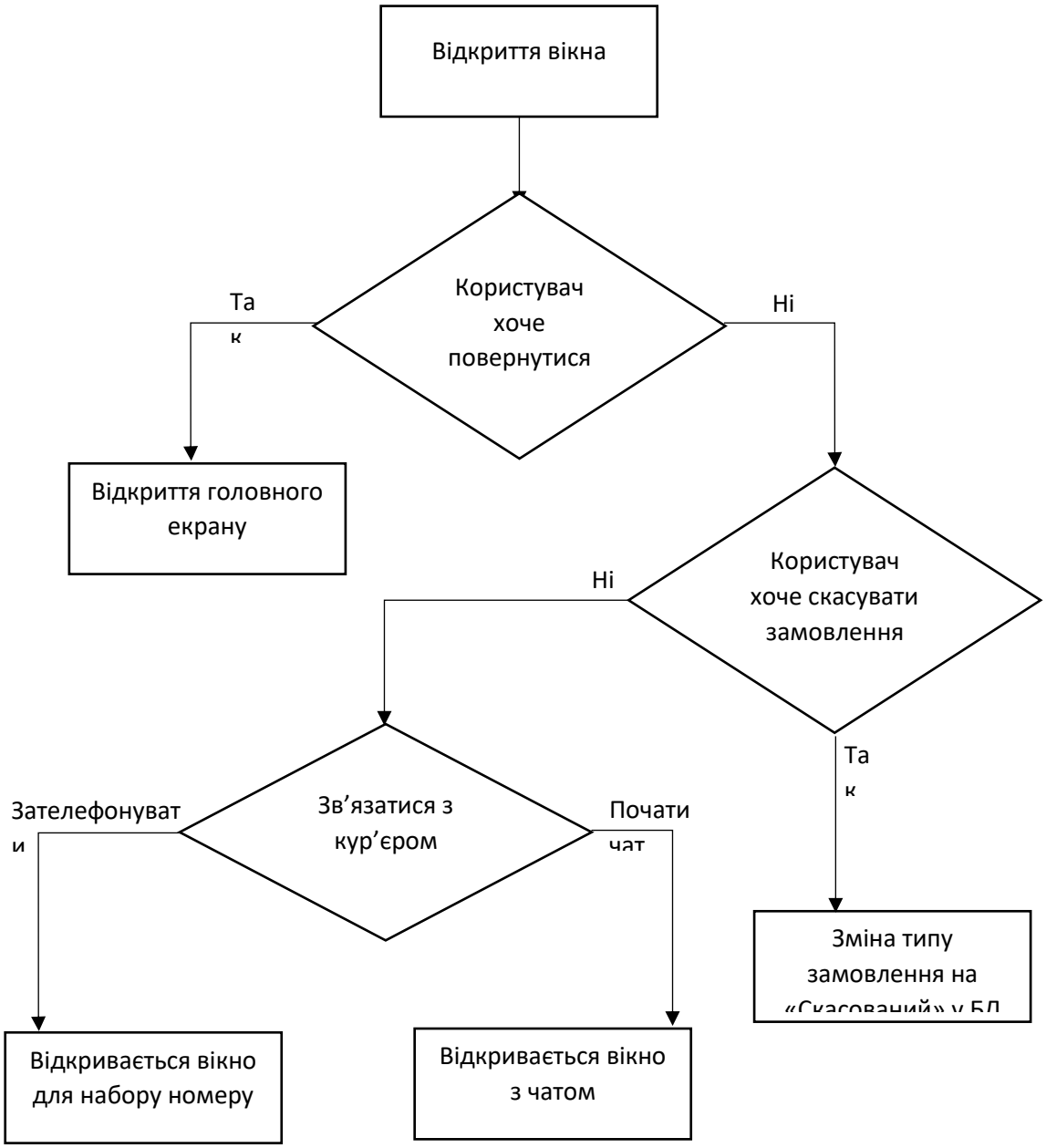



Рис. 26 Алгоритм роботи вікна доставки.

Для реалізації роботи даного алгоритму я написав наступний код рис. 27.1-27.4

```
public  void checkDevelop(String email){
    DatabaseReference usersRef = mDatabase.child( pathString: "orders");
    Query query =usersRef.orderByChild( path: "emailcustomer").equalTo(email);
    query.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            develop_classes.clear();
            if (snapshot.exists()) {
                    for (DataSnapshot basket : snapshot.getChildren()) {
                        Long type = basket.child( path: "type").getValue(Long.class);
                        if(type==1) {
                            String text1 = basket.child( path: "adress").getValue(String.class);
                            Long text2 = basket.child( path: "coust").getValue(Long.class);
                            String phon = String.valueOf(basket.child( path: "phonCur").getValue(Long.class));
                            if (text1 != null && text2 != null) {
                                Develop_Class data = new Develop_Class(text1, (String.valueOf(text2) + " грн. "),
                                        basket.getKey(), phon);
                                develop_classes.add(data);
                            }
                        }
                    }

                }

                adapter.notifyDataSetChanged();
            }
```

Рис. 27.1 Метод для перевірки наявності актуальних замовлень конкретного користувача.

Цей метод checkDevelop перевіряє базу даних на наявність замовлень, які відповідають заданій електронній адресі користувача (email). Метод створює запит до бази даних Firebase, щоб отримати всі замовлення, де поле "emailcustomer" дорівнює вказаному email. Після отримання результатів метод очищує список develop_classes і додає до нього об'єкти типу Develop_Class для замовлень, що задовільняють умову (if type==1) , які включають адресу, вартість, індентифікатор та телефонні дані. Зміни у списку спричиняють оновлення адаптера, що відображає дані у інтерфейсі.

Цей код забезпечує динамічне завантаження даних з бази даних Firebase та їх відображення у списку для користувача в додатку.

```
holder.imageViewPhon.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        int permicssionCheck = ContextCompat.checkSelfPermission(view.getContext(), Manifest.permission.CALL_PHONE);

        if (permicssionCheck!= PackageManager.PERMISSION_GRANTED){
            ActivityCompat.requestPermissions(
                    (Activity) view.getContext(),new String[]{Manifest.permission.CALL_PHONE}, requestCode: 123
            );

        }
        else {
          Intent intent= new Intent(Intent.ACTION_CALL, Uri.parse( uriString: "tel: "+phon));
          view.getContext().startActivity(intent);
        }
    }
});

}
```

Рис. 27.2 Метод для набору номера кур'єра.

Цей код перевіряє наявність дозволу CALL_PHONE у додатку. Він використовує ContextCompat.checkSelfPermission для перевірки дозволу. Якщо дозвіл не наданий (PERMISSION_GRANTED), викликається ActivityCompat.requestPermissions, щоб запитати користувача про дозвіл на виклик телефону. Якщо дозвіл вже наданий, створюється інтент для виклику телефону з вказаним номером (phon) і запускається активність для здійснення дзвінка. Цей код забезпечує безпечний доступ до функціоналу дзвінків на Android, дозволяючи користувачеві надати необхідні дозволи до використання функціоналу.

```
holder.imageViewCancelDevelop.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        mDatabase = FirebaseDatabase.getInstance().getReference();
        DatabaseReference databaseReference = mDatabase.child( pathString: "orders").child(id);
        databaseReference.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(snapshot.exists()){
                    databaseReference.child( pathString: "type").setValue(0);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });
    }
});
```

Рис. 27.3 Метод для скасування замовлення.

Цей код отримує посилання на базу даних Firebase (mDatabase), після чого визначає посилання на конкретний об'єкт у вузлі "orders" за допомогою унікального ідентифікатора id. Потім він встановлює прослуховувач для одноразового отримання значення (addListenerForSingleValueEvent). У методі onDataChange перевіряється наявність даних у зазначеному вузлі. Якщо дані існують, встановлюється значення поля "type" у 0 для об'єкта з вказаним id.

Цей код призначений для оновлення конкретного поля в базі даних Firebase після одноразового отримання даних, і викликається лише один раз для заданого вузла.

```
holder.imageViewChat.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(view.getContext(), Chat_Customer.class);
        intent.putExtra( name: "chatKey",id);
        view.getContext().startActivity(intent);
    }
});
```

Рис. 27.4 Метод для відкриття вікна чату.

Цей код виконує дії при кліку на елемент інтерфейсу. Коли користувач клікає на елемент, створюється новий інтент для переходу до активності Chat_Customer. У цьому інтенті передається додаткова інформація (ключ "chatKey" зі значенням id). Після цього стартується нова активність для відображення чату з клієнтом чи користувачем зі вказаним id.

Цей код дозволяє запускати нову активність при кліку на елемент і передавати додаткові дані для відображення чату.

# ЗАГАЛЬНІ ВИСНОВКИ

У дипломній роботі я розробив мобільний застосунок служби кур’єрської доставки їжі.

Для кращого розуміння сфери громадського харчування я провів дослідження в ході якого порівнював традиційні заклади і служби доставки. В результаті цього дослідження я зміг чітко встановити актуальність даного ПЗ вимоги до нього та можливі характеристики. Дана інформація допомогла мені чітко спроектувати та розробити додаток «FoodFlyer».

Основними функція даного додатку є :

- **Реєстрація та вхід.** Створення облікового запису. Вхід до системи.
- **Профіль користувача.** Перегляд і редагування особистої інформації. Збереження адрес доставки.
- **Меню та каталог.** Перегляд меню з фотографіями, описами та цінами страв. Фільтрація страв за категоріями.
- **Замовлення.** Додавання страв до кошика. Перегляд та редагування замовлення перед підтвердженням.

Таким чином, розроблений додаток «FoodFlyer» враховує всі необхідні аспекти для забезпечення ефективної роботи служби кур'єрської доставки їжі. Він полегшує процес замовлення для клієнтів, оптимізує роботу кур'єрів та надає адміністрації кафе необхідні інструменти для управління замовленнями та аналітики. Впровадження цього додатку сприятиме підвищенню рівня обслуговування, збільшенню клієнтської бази та покращенню загальної ефективності бізнесу в сфері громадського харчування.

# ПЕРЕЛІК ПОСИЛАНЬ


1. Android Developer Documentation – [Електронний ресурс] – Режим доступу: https://developer.android.com/docs (Дата звернення 01.01.2024)
2. Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs (Дата звернення 01.01.2024)
3. "Android Studio User Guide", Google Inc. – [Електронний ресурс] – Режим доступу: https://developer.android.com/studio/intro (Дата звернення 01.01.2024)
4. "Firebase Realtime Database", Google Inc. – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/database (Дата звернення 01.01.2024)
5. "Authentication", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/auth (Дата звернення 01.01.2024)
6. "Cloud Messaging", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/cloud-messaging (Дата звернення 01.01.2024)
7. "Cloud Firestore", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/firestore (Дата звернення 01.01.2024)
8. "Crashlytics", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/crashlytics (Дата звернення 01.01.2024)
9. "Analytics", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/analytics (Дата звернення 01.01.2024)
10. "App Distribution", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/app-distribution (Дата звернення 01.01.2024)
11. "Implementing Push Notifications in Android", [Електронний ресурс] – Режим доступу: https://www.digitalocean.com/community/tutorials/android-push-notifications-with-firebase-cloud-messaging (Дата звернення 01.01.2024)

12. "Guide to Android App Architecture", Google Developers – [Електронний ресурс] – Режим доступу: https://developer.android.com/jetpack/guide (Дата звернення 01.01.2024)

13. "Build a Real-time Chat Application using Firebase", [Електронний ресурс] – Режим доступу: https://www.freecodecamp.org/news/build-a-real-time-chat-app-with-reactjs-and-firebase (Дата звернення 01.01.2024)

14. "Firebase Cloud Functions", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/functions

15. (Дата звернення 08.06.2024) "Data and file storage in Firebase", [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/storage (Дата звернення 01.01.2024)

16. "Getting Started with Firebase Authentication on Android", [Електронний ресурс] – Режим доступу: https://auth0.com/blog/firebase-authentication-on-android (Дата звернення 08.06.2024)

17. "Managing Firebase Projects", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/projects/overview (Дата звернення 01.01.2024)

18. "Firebase Hosting", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/hosting (Дата звернення 01.01.2024)

19. "Using Firebase for Android App Development: A Beginner's Guide", [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/using-firebase-for-android-app-development-a-beginners-guide/ (Дата звернення 01.01.2024)

20. "Firebase Test Lab", Firebase Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/test-lab (Дата звернення 01.01.2024)

21."Мінфін", [Електронний ресурс] – Режим доступу: https://minfin.com.ua/ (Дата звернення 02.05.2024)

22. «Епіцентр», [Електронний ресурс] – Режим доступу: https://epicentrk.ua/ (Дата звернення 02.05.2024).
23. «DIM.RIA», [Електронний ресурс] – Режим доступу: https://dom.ria.com/uk/arenda-ofisov/rovno/ (Дата звернення 02.05.2024).
24. «Prom», [Електронний ресурс] – Режим доступу: https://prom.ua/ua/?utm_source=google_brend&utm_medium=cpc&utm_content=kw&utm_campaign=Search_Prom_Brand&gad_source=1&gclid=Cj0KCQjwsaqzBhDdARIsAK2gqnfP22FE7fOAjVLfUtdnCq1ViMf05BsCSEy2uRKRpBXUCTclN3wE1sIaAmNQEALw_wcB (Дата звернення 02.05.2024).
25. «YC MARKET», [Електронний ресурс] – Режим доступу: https://catalog.youcontrol.market/hromadske-kharchuvannia/rivnenska-oblast/rivne (Дата звернення 02.05.2024).
26. «Рівненське обласне управління статистики» », [Електронний ресурс] – Режим доступу: https://gusrv.gov.ua/ (Дата звернення 02.05.2024).

# ПЕРЕЛІК ДОДАТКІВ

1. **Додаток А.** Технічне завдання до програмного продукту.

**Додаток А.**

Технічне завдання

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ**

# ТЕХНІЧНЕ ЗАВДАННЯ

на розробку програмного продукту «Мобільний додаток кур’єрської доставки»

спеціальність 126 «Інформаційні системи і технології»

освітньо – кваліфікаційний рівень «Бакалавр»

Виконав здобувач освіти ІСТ-21інг групи Антонюк Б.С.

Рівне - 2024

# ВСТУП

Сьогодні ми живемо в динамічному світі, де час є найціннішим ресурсом. У зв'язку з цим, швидкість і зручність доставки їжі стали невід'ємною частиною нашого повсякденного життя.

Мета створення мобільного додатку для служби кур'єрської доставки їжі полягає в тому, щоб забезпечити користувачам максимально комфортний, швидкий і надійний сервіс доставки різноманітних страв.

Додаток покликаний оптимізувати процес замовлення та відстеження доставки, надаючи можливість користувачам легко та зручно замовляти їжу приготовану за особливими рецептами наших кухарів.

Використовуючи сучасні технології та інноваційні підходи, ми прагнемо зробити процес доставки ще більш ефективним та прозорим, задовольняючи потреби як приватних клієнтів, так і бізнесу.

## РОЗДІЛ І. НАЙМЕНУВАННЯ Й ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування програмного продукту «FoodFlyer». Додаток розробляється на платформі Android, застосовуватись буде в першу чергу на мобільних пристроях.

## РОЗДІЛ ІІ. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки даного додатку є зростаючий попит на швидку та зручну доставку їжі, що обумовлено збільшенням темпу життя сучасних людей.

## РОЗДІЛ ІІІ. ПРИЗНАЧЕННЯ РОЗРОБКИ

ПЗ призначене для зручного замовлення товарів, відстеження доставки та для можливості зв’язку з кур’єром.

## РОЗДІЛ IV. ТЕХНІЧНІ ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

### *4.1 Вимоги до функціональних характеристик:*

Програма забезпечує можливість ознайомлення з продукцією, додавання товарів у кошик користувача, можливості налаштування профілю користувача, для зручності зберігання даних про користувачів та їх замовлення програма відправляє та обробляє запити з БД що знаходиться на сервері.

### *4.2 Вимоги до надійності:*

– Передбачено необхідність авторизації для подальшої роботи.

– Передбачено блокування некоректних дій користувача при роботі.

– При виникненні збою в роботі продукт повертає користувача в попередній стан роботи.

### *4.3 Умови експлуатації:*

Має здійснюватися періодичне оновлення інформації відповідно до оновлення вхідних даних компанії.

### *4.4 Вимоги до складу й параметрів технічних засобів:*

– об’єм ОП від 1 ГБ;

– наявність 128 Мб вільного місця на телефоні;

### *4.5 Вимоги до інформаційної й програмної сумісності:*

Для роботи програмного продукту на телефоні має бути встановлена ОС Android версії 8.0 або вище .

## РОЗДІЛ V. СТАДІЇ Й ЕТАПИ РОЗРОБКИ

Етапами розробки програмної системи є:

– Специфікація вимог – буде проведено збір та аналіз інформації за досліджуваною темою.

– Постановка задачі – буде проведено систематизацію інформації ПО та розроблено ТЗ.

– Проєктування – створення алгоритму, за яким буде створене саме програмне забезпечення.

– Розробка основи проєкту – саме написання програмного продукту на вибраній мові програмування.

– Тестування і запуск проєкту – буде проведено розміщення системи на емуляторі пристрою та тестування готового програмного продукту.

## РОЗДІЛ VI. ПОРЯДОК КОНТРОЛЮ Й ПРИЙМАННЯ

Контроль програмного забезпечення проводиться після кожного етапу розробки ПЗ, для подальшого уникнення помилок на наступних етапах, і загалом у всьому програмному забезпеченні. Також буде проводитись контроль на перевірку відповідності програмного продукту поставленим вимогам.

Приймання програмного забезпечення замовником відбудеться тільки тоді, коли програма буде відповідати поставленим початковим вимогам, без помилок під час роботи.

Перед введенням в експлуатацію будуть проведені тестування на стабільність, безпеку та функціонування програмного забезпечення.