Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем та комп'ютерного моделювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення мобільного застосунку "Розумний офіціант"»

Виконав студент 4 курсу, групи ІСТ-41 спеціальності:
126 „Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Корнелюк Ю. О.
(прізвище, ініціали)

Керівник:
Флуд Л.О.
(прізвище, ініціали)

Рецензент: Дрощик Ю.С.
(прізвище, ініціали)

Львів-2024

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший(бакалаврський)
Спеціальність 126 " Інформаційні системи та технології "

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
Сторожук О.Л.
„ 06 " 02 2024року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Корнелюк Юрій Олегович
(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: «Розроблення мобільного застосунку "Розумний офіціант"»

керівник роботи Флуд Любомир Олегович, кандидат технічних наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "6" лютого 2024 року, №С-87
2.Термін подання студентом проекту(роботи) 10 червня 2024 р.
3. Вихідні дані до проекту (роботи) Ознайомитися з проблемною областю, розробити та реалізувати мобільний застосунок для автоматизації процесів прийому та обробки замовлень у закладах ресторанного господарства з використанням сучасних засобів і технологій розробки Android-застосунків.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Стан проблемної області
Інформаційне та математичне забезпечення
Програмне та технічне забезпечення
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Підготовка матеріалу до доповіді .
6. Дата видачі завдання "7" лютого 2024 року

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	07.02-20.02	Виконано
2.	Постановка задачі і її формалізація	20.02-05.03	Виконано
3.	Виконання вхідного етапу технології	05.03-25.03	Виконано
4.	Реалізація головних класів проекту	25.03-14.04	Виконано
5.	Виконання етапу відлагодження проєкту	14.04.-10.05	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	10.05.-02.06.	Виконано
7.	Оформлення записки до дипломного проекту.	02.06.-10.06.	Виконано

Студент _____ Корнелюк Ю. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Флуд Л.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 58 сторінок пояснювальної записки, 38 рисунків, 3 додатки, 15 джерел.

Бакалаврська дипломна робота присвячена розробці мобільного застосунку "Розумний офіціант" для оптимізації роботи офіціантів у ресторанах. Основна мета роботи - створення ефективного інструменту для прийому та обробки замовлень, управління рахунками клієнтів та покращення комунікації між клієнтами та кухнею.

Ключові слова: Мобільний застосунок**,** Офіціант**,** Автоматизація замовлень**,** Управління рахунками**,** Firestore Cloud**,** Android Studio**,** Java**,** Інтеграція.

ABSTRACT

The diploma thesis contains 58 pages of explanatory notes, 38 figures, 3 appendices, 15 sources.

Bachelor's Thesis is dedicated to the development of a mobile application "Розумний Офіціант" for optimizing the work of waiters in restaurants. The main goal of the work is to create an efficient tool for receiving and processing orders, managing customer bills, and improving communication between customers and the kitchen.

Keywords: Mobile application, Waiter, Order automation, Bill management, Firestore Cloud, Android Studio, Java, Integration.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити мобільний додаток для Android - "Розумний офіціант”. Додаток має давати можливість користувачам переглянути меню, ціни страв та напоїв, зробити замовлення та оплатити послуги закладу з можливістю додавання чайових для офіціанта. Реалізувати проєкт з допомогою засобів Java, бібліотеки Gradle, android jetpack та ін. Розроблений застосунок повинен бути лаконічним, інтуїтивно зрозумілим та зручним у користуванні. Дизайн додатку повинен відображати сучасні тенденції, бути привабливим та відповідати потребам користувачів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

1. UI - User Interface (Користувацький інтерфейс)

2. UX - User Experience (Користувацький досвід)

3. API - Application Programming Interface (Інтерфейс програмування додатків)

4. DB - Database (База даних)

5. CRUD - Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення)

6. SDK - Software Development Kit (Набір засобів розробки програмного забезпечення)

7. JSON - JavaScript Object Notation (Нотація об'єктів JavaScript)

8. XML - eXtensible Markup Language (Розширювана мова розмітки)

9. HTTP - HyperText Transfer Protocol (Протокол передачі гіпертексту)

10. SSL/TLS - Secure Sockets Layer / Transport Layer Security (Рівень захищених сокетів / Транспортний рівень безпеки)

11. IDE - Integrated Development Environment (Інтегроване середовище розробки)

12. MVC - Model-View-Controller (Модель-Вид-Контроллер)

13. Firebase - Платформа для розробки мобільних та веб-додатків від Google

14. Firestore - Cloud Firestore (Хмарне сховище від Firebase)

15. CI/CD - Continuous Integration / Continuous Deployment (Безперервна інтеграція / Безперервне розгортання)

16. MVVM - Model-View-ViewModel (Модель-Вид-Модель подання)

17.KPI - Key Performance Indicator (Ключовий показник ефективності)

18. RAM - Random Access Memory (Оперативна пам'ять)

19. GPS - Global Positioning System (Глобальна система позиціонування)

20. CRUD - Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення)

ВСТУП

Актуальність теми. У сучасному світі індустрія ресторанного бізнесу постійно розвивається, вимагаючи впровадження новітніх технологій для підвищення ефективності обслуговування клієнтів. Традиційні методи управління замовленнями часто є недостатньо ефективними, призводять до затримок та помилок, що негативно впливає на рівень задоволеності клієнтів.

Розробка мобільного застосунку "Розумний офіціант" є актуальною, оскільки він допоможе автоматизувати процес прийому та обробки замовлень, покращити комунікацію між офіціантами, клієнтами та кухнею, а також оптимізувати управління рахунками клієнтів.

Об’єкт дослідження. Процес автоматизації роботи персоналу в ресторанах за допомогою мобільного застосунку.

Предмет дослідження. Мобільний застосунок "Розумний офіціант" як інструмент для прийому та обробки замовлень, управління рахунками клієнтів та покращення комунікації між офіціантами, клієнтами та кухнею.

Мета роботи. Розробка та впровадження мобільного застосунку "Розумний офіціант" для підвищення ефективності роботи офіціантів у ресторанах, зниження кількості помилок при обробці замовлень та підвищення продуктивності між клієнтом та рестораном. Для реалізації поставленої задачі необхідно провести аналіз існуючих рішень для автоматизації роботи офіціантів; визначити вимоги до функціональності мобільного застосунку; розробити архітектуру додатку та вибрати відповідні технології та інструменти; реалізувати основні функціональні можливості застосунку (аутентифікація користувачів, прийом та обробка замовлень, управління рахунками); інтегрувати додаток з Firestore Cloud для зберігання та синхронізації даних; провести модульне та інтеграційне тестування додатку; оцінити продуктивність та зручність використання додатку; підготувати рекомендації щодо подальшого розвитку та вдосконалення системи.

Практичне значення. Розроблений мобільний застосунок "Розумний офіціант" може бути впроваджений у ресторанах для автоматизації процесу прийому та обробки замовлень, що сприятиме зниженню кількості помилок,

підвищенню швидкості обслуговування клієнтів та загальному покращенню якості сервісу. Це допоможе ресторанам ефективніше використовувати ресурси та підвищити рівень задоволеності клієнтів, що в свою чергу позитивно вплине на їхню конкурентоспроможність.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд галузі обслуговування

Галузь обслуговування, зокрема ресторанний бізнес, займає важливе місце в сучасному суспільстві, пропонуючи різноманітні послуги, що сприяють задоволенню потреб споживачів у харчуванні, відпочинку та соціалізації. Відвідування ресторанів і кафе стало невід'ємною частиною повсякденного життя багатьох людей, що створює високий попит на якісне обслуговування та інноваційні рішення в цій сфері.

Одним із ключових аспектів ресторанного обслуговування є роль офіціантів, які виступають безпосередніми посередниками між кухнею та клієнтами. Офіціанти не лише приймають та подають замовлення, а й створюють приємну атмосферу, підтримуючи високий рівень сервісу та забезпечуючи задоволення клієнтів. Їхня робота включає в себе не лише технічні аспекти, але й комунікаційні навички, що дозволяють будувати довірчі відносини з гостями закладу [13].

Проте сучасні офіціанти стикаються з численними викликами, які ускладнюють їхню роботу та впливають на якість обслуговування. Велике навантаження, що виникає у пікові години, вимагає від офіціантів високої швидкості та ефективності. Недостатня кількість персоналу призводить до перевантаження, стресу та втоми, що збільшує ризик помилок. Людський фактор, включаючи емоційне напруження та фізичну втому, також впливає на продуктивність і якість обслуговування.

Затримки в обслуговуванні можуть бути викликані як технічними проблемами (наприклад, несправністю POS-систем), так і недостатньою координацією між працівниками ресторану. Це призводить до зниження рівня задоволеності клієнтів і, як наслідок, до втрати лояльних гостей. Крім того, помилки в замовленнях, спричинені неправильним прийомом або передачею інформації на кухню, створюють негативний досвід для клієнтів та знижують репутацію закладу.

Враховуючи ці виклики, стає очевидною необхідність впровадження нових технологій та методів автоматизації процесів у ресторанній сфері. Мобільні додатки, як-от "розумний офіціант", можуть значно підвищити ефективність роботи персоналу, знизити навантаження та покращити якість обслуговування. Це, у свою чергу, сприяє задоволенню потреб клієнтів і підвищенню конкурентоспроможності ресторанного бізнесу.

1.2. Технології в сфері ресторанного бізнесу

Індустрія гостинності, зокрема ресторанний бізнес, протягом останніх десятиліть зазнала значних змін завдяки впровадженню різноманітних технологічних рішень. Технології стали невід'ємною частиною сучасних ресторанів, допомагаючи автоматизувати процеси, підвищувати ефективність обслуговування та покращувати досвід клієнтів [12].

Історія впровадження технологій у ресторанах

Впровадження технологій у ресторанний бізнес розпочалося ще з появою перших POS-систем (Point of Sale), які дозволяли автоматизувати процес обробки замовлень та управління продажами. З часом, ці системи стали більш інтегрованими, додаючи функціональність для управління запасами, планування робочого часу персоналу та аналізу бізнес-даних [15].

Сучасні технології автоматизації обслуговування

1. POS-системи:
    - Сучасні POS-системи є центральним елементом більшості ресторанів, забезпечуючи швидку та точну обробку замовлень, інтеграцію з кухонним обладнанням та управління фінансами.
    - Вони також дозволяють збирати дані про продажі та поведінку клієнтів, що допомагає в прийнятті обґрунтованих бізнес-рішень.
2. Електронні меню та планшети:
    - Використання електронних меню та планшетів дозволяє клієнтам самостійно переглядати меню, робити замовлення та оплачувати рахунки.

- Це знижує навантаження на офіціантів та мінімізує людський фактор у процесі прийому замовлень.

3. Мобільні додатки для замовлення:
   - Мобільні додатки стають все більш популярними серед ресторанів, дозволяючи клієнтам робити замовлення заздалегідь, бронювати столики та отримувати персоналізовані пропозиції.
   - Вони також можуть інтегруватися з програмами лояльності, надаючи додаткові вигоди постійним клієнтам.
4. Автоматизовані системи управління запасами:
   - Ці системи допомагають ресторанам ефективно управляти запасами продуктів, зменшуючи кількість відходів та оптимізуючи закупівлі.
   - Вони автоматично відстежують рівні запасів та можуть формувати замовлення на постачання у разі необхідності.
5. CRM-системи (Customer Relationship Management):
   - CRM-системи допомагають ресторанам будувати та підтримувати відносини з клієнтами, зберігаючи інформацію про їхні уподобання, історію замовлень та контактні дані.
   - Це дозволяє створювати персоналізовані маркетингові кампанії та покращувати загальний досвід клієнтів.
6. Аналітичні платформи:
   - Використання аналітичних платформ дозволяє ресторанам аналізувати великі обсяги даних, виявляти тренди та прогнозувати майбутні потреби.
   - Це сприяє оптимізації операційних процесів та прийняттю обґрунтованих рішень щодо розвитку бізнесу.

Приклади успішного застосування технологій в ресторанах

1. McDonald's:
    - Використання кіосків для самообслуговування дозволило зменшити черги та підвищити швидкість обслуговування клієнтів.
    - Інтеграція з мобільними додатками для замовлення їжі дозволяє клієнтам робити замовлення заздалегідь та отримувати спеціальні пропозиції.
2. Starbucks:
    - Мобільний додаток Starbucks Rewards дозволяє клієнтам накопичувати бали за кожну покупку та використовувати їх для отримання безкоштовних напоїв та їжі.

- Впровадження мобільного замовлення та оплати дозволило зменшити час очікування та підвищити зручність для клієнтів.

3. Domino's Pizza:

- Інноваційна платформа для онлайн-замовлень дозволяє клієнтам створювати персоналізовані піци та відстежувати статус їхнього замовлення в режимі реального часу.
- Використання дронів та автономних транспортних засобів для доставки їжі демонструє прагнення компанії до інновацій та підвищення ефективності.

Таким чином, сучасні технології грають вирішальну роль у розвитку ресторанного бізнесу, допомагаючи автоматизувати процеси, підвищувати ефективність обслуговування та покращувати досвід клієнтів. Впровадження інноваційних рішень, таких як мобільні додатки та автоматизовані системи, дозволяє ресторанам залишатися конкурентоспроможними та відповідати зростаючим вимогам сучасних споживачів.

1.3. Аналіз існуючих рішень

Розвиток цифрових рішень та автоматизація процесів стали ключовими аспектами для багатьох закладів, які прагнуть покращити якість обслуговування, оптимізувати операційні процеси та підвищити задоволеність клієнтів. Нижче наведено аналіз існуючих рішень у цій сфері в Україні.

POS-системи

Poster - одна з найбільш популярних POS-систем в Україні. Пропонує широкий спектр функцій, включаючи управління замовленнями, облік запасів, аналітику продажів та інтеграцію з програмами лояльності. Відома своєю простотою у використанні та гнучкістю налаштувань для різних типів закладів.

Iiko - Російська POS-система, яка також широко використовується в Україні. Забезпечує повну автоматизацію бізнес-процесів, включаючи управління персоналом, фінансовий облік та контроль запасів. Інтегрується з онлайн-системами замовлень і доставки [14].

Електронні меню та мобільні додатки

Menu.ua - Онлайн-сервіс для замовлення їжі з різних ресторанів, що дозволяє користувачам переглядати меню, робити замовлення та оплачувати їх онлайн. Спрощує процес замовлення як для клієнтів, так і для ресторанів.

Glovo, Uber Eats, Raketa - Мобільні додатки для доставки їжі, які стали дуже популярними в українських містах. Пропонують зручний інтерфейс для замовлення їжі з різних ресторанів, відстеження замовлення та здійснення оплати. Інтеграція з ресторанними POS-системами для автоматизації процесу прийому замовлень.

CRM-системи

Poster Loyalty - інтегрована з Poster POS система лояльності, що дозволяє ресторанам будувати базу постійних клієнтів, накопичувати бали та надавати персоналізовані пропозиції. Допомагає у підвищенні лояльності клієнтів та збільшенні повторних продажів.

KeepinCRM - українська CRM-система, що підходить для малого та середнього бізнесу, включаючи ресторани. Пропонує функціонал для управління контактами, ведення історії замовлень, аналізу поведінки клієнтів та створення маркетингових кампаній.

Автоматизовані системи управління запасами

SmartyPos - пропонує функціонал для автоматизованого управління запасами, аналізу продажів та контролю витрат. Забезпечує інтеграцію з POS-системами для оптимізації процесів закупівель та управління запасами.

Аналітичні платформи

Аналізатор бізнесу - українська платформа для аналізу даних продажів, витрат та прибутків. Допомагає ресторанам отримувати детальні звіти та прогнозувати майбутні потреби на основі даних.

Відгуки та рейтинг

Google My Business, TripAdvisor - платформи, що дозволяють клієнтам залишати відгуки та оцінки ресторанів. Важливі інструменти для збору зворотного зв'язку та покращення якості обслуговування.

Переваги та недоліки існуючих рішень

Переваги:

1. Автоматизація процесів: Сучасні технології допомагають автоматизувати багато рутинних процесів, зменшуючи навантаження на персонал та знижуючи кількість помилок.

2. Покращення обслуговування клієнтів: Використання електронних меню, мобільних додатків та CRM-систем дозволяє покращити досвід клієнтів, забезпечуючи швидке та якісне обслуговування.

3. Ефективне управління запасами: Автоматизовані системи управління запасами допомагають оптимізувати процеси закупівель та знижувати витрати.

4. Аналітика та звітність: Аналітичні платформи забезпечують детальну звітність та аналіз даних, що допомагає приймати обґрунтовані бізнес-рішення.

Недоліки:

1. Висока вартість впровадження: Впровадження сучасних технологій може вимагати значних початкових інвестицій, що може бути складним для малого бізнесу.

2. Необхідність навчання персоналу: Нові системи потребують навчання персоналу, що може зайняти час та вимагати додаткових ресурсів.

3. Інтеграційні проблеми: Інтеграція різних систем може бути складною та вимагати додаткових технічних знань.

Український ресторанний бізнес активно впроваджує сучасні технології для автоматизації процесів та покращення якості обслуговування. Існуючі рішення, такі як POS-системи, мобільні додатки для замовлення, CRM-системи та аналітичні платформи, допомагають ресторанам залишатися конкурентоспроможними та задовольняти зростаючі вимоги споживачів. Однак, попри численні переваги, впровадження нових технологій вимагає значних ресурсів та зусиль, що підкреслює необхідність ретельного планування та навчання персоналу.

1.4. Проблеми та потреби користувачів

Галузь ресторанного обслуговування стикається з численними викликами, пов'язаними з задоволенням потреб клієнтів. Сучасні споживачі очікують високої якості обслуговування, швидкості та точності у виконанні замовлень, а також персоналізованого підходу. У той же час, ресторани зіштовхуються з проблемами, які можуть перешкоджати досягненню цих очікувань. Нижче наведено основні проблеми та потреби користувачів у ресторанному бізнесі [15].

Проблеми користувачів

1. Довге очікування
   - Очікування на вільний столик: В популярних закладах часто виникають ситуації, коли клієнтам доводиться довго чекати на вільний столик, особливо у пікові години. Це створює незручності та негативно впливає на загальне враження від відвідування ресторану.
   - Час очікування на замовлення: Клієнти можуть стикатися з тривалим часом очікування на прийом замовлення та його подачу. Це особливо актуально в закладах з великою кількістю відвідувачів, де офіціанти не завжди встигають швидко обслуговувати всіх клієнтів.
   - Затримки у виставленні рахунку: Після завершення трапези, клієнти часто змушені чекати на виставлення рахунку, що може викликати роздратування, особливо якщо вони поспішають.
2. Помилки в замовленнях
   - Неправильні замовлення: Через людський фактор офіціанти можуть допускати помилки при прийомі або передаванні замовлень на кухню. Це призводить до подачі неправильних страв або напоїв, що створює незручності для клієнтів.
   - Нестача або надлишок замовлень: Можливі ситуації, коли клієнти не отримують усі замовлені страви або отримують додаткові, які вони не замовляли. Це знижує рівень задоволеності клієнтів та негативно впливає на їхній досвід.
3. Відсутність персоналізованого підходу
   - Індивідуальні уподобання та обмеження: Клієнти можуть мати специфічні вподобання або дієтичні обмеження, які не завжди враховуються офіціантами

через недостатнє знання або комунікаційні проблеми. Це може призвести до невідповідності замовлень потребам клієнтів.

- Брак інформації про клієнтів: Без відповідних технологічних рішень, офіціанти не можуть надавати персоналізовані рекомендації та пропозиції на основі попереднього досвіду клієнтів, що знижує якість обслуговування.

4. Недостатня увага

- Перевантаженість офіціантів: У пікові години офіціанти можуть не встигати приділяти достатньо уваги кожному клієнту, що негативно впливає на загальне враження від відвідування закладу. Перевантаженість персоналу призводить до зниження якості обслуговування.
- Низька якість обслуговування**: Висока завантаженість і стресові ситуації можуть призводити до зниження якості обслуговування, що відображається у поведінці та комунікації офіціантів з клієнтами.

Потреби користувачів

1. Швидке та ефективне обслуговування

- Оптимізація часу очікування: Клієнти очікують мінімального часу очікування на вільний столик, прийом замовлення та подачу страв. Впровадження сучасних технологій, таких як системи управління чергами та бронювання столиків, допомагає знизити час очікування.
- Ефективне управління чергою: Використання технологій для управління чергою та бронювання столиків допомагає знизити час очікування та покращити досвід клієнтів.

2. Точність та надійність

- Безпомилкові замовлення: Клієнти хочуть бути впевненими, що їхні замовлення будуть прийняті та виконані точно, без помилок. Впровадження електронних систем замовлення знижує ризик помилок і підвищує точність виконання замовлень.
- Своєчасна подача страв: Важливо, щоб страви подавалися у правильному порядку та відповідно до замовлення клієнтів. Це підвищує задоволеність відвідувачів та покращує загальний досвід.

3. Персоналізований підхід

- Знання уподобань клієнтів: Клієнти цінують, коли офіціанти знають їхні уподобання та можуть запропонувати страви або напої, які відповідають їхнім індивідуальним потребам. Використання CRM-систем допомагає збирати та аналізувати дані про клієнтів для надання персоналізованих пропозицій.
- Індивідуальні рекомендації: Використання даних про попередні відвідування для надання персоналізованих рекомендацій та пропозицій підвищує лояльність клієнтів та їхнє задоволення від обслуговування.

4. Комфорт та зручність

- Простий процес замовлення та оплати: Клієнти очікують зручного процесу замовлення та оплати, включаючи можливість безконтактної оплати та використання мобільних додатків. Це знижує час очікування та підвищує зручність обслуговування.
- Зрозуміле меню: Чітке та зрозуміле меню з детальною інформацією про страви та напої, включаючи алергени та дієтичні опції, допомагає клієнтам робити обґрунтовані вибори та підвищує їхнє задоволення від відвідування ресторану.

Враховуючи вищезазначені проблеми та потреби користувачів, стає очевидною необхідність впровадження сучасних технологій та інноваційних рішень у ресторанний бізнес. Це дозволить підвищити якість обслуговування, зменшити навантаження на персонал та покращити загальний досвід відвідування для клієнтів.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Середовище розробки

Android Studio - офіційне середовище розробки для Android-застосунків. Воно надає широкий набір інструментів та функцій для розробників. Android Studio містить в собі налагоджувач, інструменти для профілювання, емулятор Android, візуальний редактор для створення інтерфейсів, кодовий редактор з функціями автодоповнення та підсвічування синтаксису. Завдяки інтеграції з Gradle для управління залежностями, Android Studio дозволяє експортувати та підписувати додатки у форматі APK, підтримує локалізацію та інтеграцію з сервісами Google Cloud. Регулярні оновлення забезпечують розробників актуальними інструментами та можливостями, підвищуючи ефективність процесу розробки. Android Studio також надає доступ до широкого спектру шаблонів для створення різноманітних типів додатків, включаючи платформи для мобільних пристроїв, планшетів, Android TV, Android Wear та IoT-пристроїв. Підтримуючи різні мови програмування, такі як Java, Kotlin і C++, Android Studio дозволяє розробникам вибирати найбільш підходящий для них інструмент. Інтегроване з Android SDK, це середовище спрощує розробку та тестування додатків, а також забезпечує доступ до всіх потрібних API для взаємодії з різними функціями пристроїв. Android Studio - це версія IntelliJ IDEA, одного з найпопулярніших інтегрованих середовищ розробки для програмування на Java [3].

2.2 Мова програмування

Java - найпопулярніша мова, що використовується для розробки Android-додатків. Пристрої на базі Android не підтримують запуск файлів .class та .jar. Натомість для підвищення продуктивності та ефективного використання батареї вони використовують власні оптимізовані формати компіляції коду. Це означає, що стандартне середовище розробки для мови Java не підходить, і вам знадобляться спеціальні інструменти для перетворення скомпільований коду у

формат, сумісний з Android, а також для його встановлення та налагодження на Android-пристроях. Усе необхідне міститься в Android SDK [2].

Переваги:

По-перше, Java є однією з найпопулярніших мов програмування завдяки своїй крос-платформності. Це означає, що програми, написані на Java, можуть працювати на будь-якій платформі, що підтримує віртуальну машину Java (JVM), що робить її ідеальним вибором для розробки універсальних додатків.

По-друге, Java має розгорнутий екосистему інструментів та бібліотек, що значно полегшує процес розробки. Інтеграція з популярними IDE, такими як IntelliJ IDEA або Eclipse, надає розробникам потужні інструменти для написання, налагодження та тестування коду.

По-третє, Java в Android Studio забезпечує широкий набір інструментів та ресурсів для розробників, що сприяє швидкій та ефективній розробці додатків.

Що стосується слабких сторін:

Відносно невелика продуктивність порівняно з іншими мовами програмування, такими як C++ або Python. Хоча з кожним новим випуском Java продуктивність покращується, вона все ще може відрізнятися від інших мов.

Також, Java має дещо строгу синтаксичну структуру, що може бути складною для початківців. Це може призвести до більшої кількості коду порівняно з іншими мовами, що може збільшити час розробки та підтримки програмного забезпечення.

Використання Java може бути трохи менш продуктивним порівняно з іншими мовами програмування, такими як Kotlin, який набуває популярності серед розробників Android. Також, вважається, що синтаксична структура Java може бути дещо складною для розуміння, що може призвести до більшого обсягу коду.

2.3 Firestore Cloud

Cloud Firestore — це гнучка та масштабована база даних від Firebase і Google Cloud, призначена для мобільних, веб- та серверних додатків. Як і Firebase

Realtime Database, вона дозволяє синхронізувати дані між клієнтськими додатками за допомогою прослуховувачів у реальному часі та підтримує автономну роботу на мобільних пристроях та веб-сайтах. Це дає можливість створювати адаптивні додатки, які продовжують працювати незалежно від затримок мережі чи відсутності інтернет-з'єднання. Cloud Firestore також легко інтегрується з іншими продуктами Firebase та Google Cloud, такими як Cloud Functions [4].

Ключові можливості Cloud Firestore:

1. Гнучкість моделі даних: Можливість створення гнучких ієрархічних структур даних, де дані зберігаються у вигляді документів, що організовані у колекції. Документи можуть містити складні вкладені об'єкти та підколекції, що спрощує організацію даних.

2. Експресивне запитування: Cloud Firestore дозволяє виконувати запити для отримання конкретних документів або всіх документів у колекції, які відповідають певним критеріям. Запити можуть містити кілька фільтрів та сортувань, а також індексуються за умовчанням, що підвищує продуктивність.

3. Оновлення в реальному часі: Забезпечується синхронізація даних для оновлення на всіх підключених пристроях, що дозволяє ефективно виконувати як одноразові, так і постійні запити.

4. Офлайн підтримка: Cloud Firestore кешує активно використовувані дані, дозволяючи доступ до них навіть у відсутність Інтернет-підключення. При з'єднанні з мережею всі локальні зміни синхронізуються з основною базою даних.

5. Масштабованість: Побудований на потужній інфраструктурі Google Cloud, Cloud Firestore автоматично масштабується для забезпечення високої доступності, надійності та продуктивності. Надається підтримка для автоматичного тиражування даних у кількох регіонах, гарантій узгодженості, атомарних операцій та реальних транзакцій [4].

6. Безпека даних: Cloud Firestore забезпечує різні рівні безпеки для ваших даних. Це включає обмеження доступу до документів та колекцій за допомогою прав доступу на рівні користувачів і ролей, аутентифікацію користувачів та можливість шифрування даних в покої для зберігання.

7. Моніторинг та аналітика: Cloud Firestore надає інструменти для моніторингу та аналізу продуктивності вашої бази даних. Це дозволяє вам відслідковувати використання ресурсів, виявляти потенційні проблеми та оптимізувати роботу з даними.

8. Інтеграція з іншими сервісами Google Cloud: Cloud Firestore легко інтегрується з іншими сервісами Google Cloud, такими як Firebase, Google Cloud Functions, Google Cloud Storage та іншими. Це дозволяє вам створювати складні та потужні додатки, які використовують цілу екосистему Google Cloud.

9. Резервне копіювання та відновлення даних: Cloud Firestore автоматично створює резервні копії ваших даних для захисту від втрати інформації. Ви також можете налаштувати власні політики резервного копіювання та відновлення для задоволення ваших конкретних потреб у збереженні даних [5].

2.4 Gradle

Під час інтеграційної фази проєкту команди часто стикаються з проблемами конфігурації, взаємодії, локалізації та іншими аспектами. Gradle, як інструмент автоматизації збірки, допомагає вирішити ці проблеми. Він дозволяє створювати скрипти для автоматизації процесів збірки, тестування та розгортання програмного забезпечення, що сприяє швидшому та ефективнішому вирішенню конфігураційних та інтеграційних питань, а також забезпечує повторюваність, надійність і переносимість процесу збірки та розгортання [7].

Gradle – це система збірки з відкритим кодом, яка використовується для автоматизації створення, тестування та розгортання проєктів. Файл «build.gradle» – це сценарій, що автоматизує різноманітні завдання. Наприклад, завдання копіювання файлів з одного каталогу в інший можна виконати за допомогою Gradle перед початком фактичного процесу збірки. Кожен Android-проєкт використовує

Gradle для генерації APK-файлу з .java та .xml файлів. Простіше кажучи, Gradle обробляє всі вихідні файли, перетворюючи .java файли у .dex файли та стискаючи їх у єдиний файл APK [8].

Gradle підтримує два основних типи скриптів: верхнього рівня, що визначає загальні конфігурації збірки для всіх модулів проєкту, та на рівні модуля, що містить залежності та версії SDK. Основні функції включають налаштування репозиторіїв, залежностей, специфічних параметрів збірки Android, унікального ідентифікатора програми, мінімального та цільового рівнів API, а також налаштування стиснення коду та файлів конфігурації proguard [6].

2.5 Бібліотека Glide

Glide – це швидка й ефективна бібліотека з відкритим кодом для завантаження та обробки зображень у додатках Android, яка значно спрощує управління зображеннями. Вона підтримує завантаження, декодування та відображення зображень, включаючи анімовані GIF-файли, забезпечуючи кешування як у пам'яті, так і на диску, оптимізуючи використання ресурсів. Glide надає гнучкий API, що дозволяє підключатися до різних мережевих стеків, таких як налаштований HttpUrlConnection, Volley від Google та OkHttp від Square [9].

Основне призначення Glide – забезпечити плавне та швидке прокручування списків зображень, хоча вона також ефективна для отримання, зміни розміру та відображення зображень з віддалених джерел. Використовувана в багатьох проєктах Google, включаючи офіційний додаток Google I/O, Glide є рекомендованою бібліотекою для завантаження зображень у Android-додатках. Вона допомагає розробникам вирішувати проблеми повільного завантаження, проблеми з пам'яттю та помилки завантаження, роблячи інтерфейс додатків більш інтерактивним та зручним для користувачів [10].

2.6 XML (Extensible Markup Language)

XML — це протокол для зберігання та керування інформацією через розмітку, а також набір технологій для форматування документів і фільтрації даних. Глибоке розуміння XML включає такі рівні:

1. Базовий рівень: XML зберігає і керує інформацією за допомогою розмітки, що необхідно для наступних рівнів використання.

2. Супутникові технології: XML використовується для стилізації, перетворення даних і створення власних мов розмітки.

3. Філософія обробки інформації: XML забезпечує гнучкість і корисність даних шляхом їх структуризації.

Розмітка додає інформацію до документа, що покращує його значення і визначає частини та їх взаємозв'язок. Комп'ютерні програми потребують розмітки для обробки електронних документів.

XML є набором інструментів для зберігання даних, що має такі переваги:

- Зберігає та організує інформацію відповідно до потреб користувача.
- Є відкритим стандартом, не прив'язаним до конкретного програмного забезпечення.
- Підтримує Unicode, що дозволяє працювати з різними системами письма.
- Пропонує різні способи перевірки якості документа.
- Має чіткий синтаксис і структуру, що легко читаються.
- Легко поєднується з таблицями стилів для створення форматованих документів.

XML став успішним завдяки своїй здатності адаптуватися до потреб сучасної інформаційної інфраструктури, забезпечуючи сумісність і відкритий доступ до даних [11].

Використання XML для UI в Android

XML (eXtensible Markup Language) використовується в Android для створення інтерфейсу користувача (UI) завдяки своїй простоті, структурованості та

читабельності. Використання XML для UI дозволяє відокремити логіку програми від її презентаційного шару, що полегшує розробку, тестування та підтримку додатків. Основні переваги використання XML для UI в Android включають:

- Модульність: Всі елементи інтерфейсу, такі як кнопки, текстові поля, списки та інші віджети, визначаються в окремих XML-файлах, що дозволяє легше управляти та змінювати компоненти інтерфейсу без необхідності змінювати програмний код.
- Спрощена розмітка: XML забезпечує чітку ієрархічну структуру для визначення компонентів інтерфейсу користувача, їх властивостей та відношень між ними. Це полегшує створення складних інтерфейсів за допомогою декларативного підходу.
- Повторне використання: Компоненти інтерфейсу, визначені в XML, можуть бути легко повторно використані в різних частинах програми. Це дозволяє зменшити дублювання коду та забезпечує консистентність UI.
- Підтримка різних екранних розмірів та орієнтацій: XML-файли розмітки можуть бути адаптовані до різних розмірів екрану, дозволяючи створювати інтерфейси, які добре виглядають на різних пристроях і в різних орієнтаціях.
- Інструменти розробки: IDE для Android, такі як Android Studio, мають вбудовані інструменти для роботи з XML, включаючи візуальні редактори, що дозволяють розробникам бачити зміни в інтерфейсі в режимі реального часу. Це значно пришвидшує процес розробки та налагодження.
- Відокремлення логіки від представлення: XML-файли містять лише опис інтерфейсу, тоді як логіка програми визначається у файлах з кодом на мові Java або Kotlin. Це розділення сприяє кращому розумінню та підтримці коду.
- Можливість створення кастомних компонентів: XML дозволяє легко інтегрувати та налаштовувати власні (кастомні) компоненти, що можуть бути створені розробником для специфічних потреб програми.

Загалом, використання XML для визначення інтерфейсу користувача в Android є потужним і гнучким інструментом, який дозволяє створювати ефективні та адаптивні UI для різноманітних мобільних додатків [2].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Розроблення застосунку “Розумний офіціант”

Мова програмування

Основною мовою програмування для розробки мобільного застосунку "Розумний офіціант" є Java. Ця мова обрана завдяки своїй сумісності з Android-платформою, широкій підтримці спільноти розробників та наявності великої кількості бібліотек і фреймворків.

Інтегроване середовище розробки (IDE)

Для розробки використовується Android Studio - офіційне середовище розробки для Android, яке забезпечує повний набір інструментів для створення, тестування та відлагодження мобільних застосунків.

База даних

Для зберігання даних використовується Firestore Cloud від Firebase. Це NoSQL база даних, що дозволяє зберігати дані у вигляді документів та колекцій, забезпечуючи високу продуктивність та масштабованість.

3.2 Проектування бази даних

Проектування бази даних є ключовим етапом розробки мобільного застосунку "Розумний офіціант". База даних повинна забезпечити зберігання та швидкий доступ до всіх необхідних даних: інформації про користувачів, меню, замовлення та рахунки.

Вимоги до бази даних

Перед проектуванням бази даних були визначені наступні вимоги:

- **Зберігання інформації про меню**: Назва категорії, іконка категорії, назва страв, ціни, описи, зображення.

- **Замовлення клієнтів**: Інформація про страви, що були замовлені, статуси замовлень.
- **Інформація про столики**: Номери столиків.
- **Користувачі (офіціанти та адміністратори)**: Логіни, паролі, рівні доступу.

СУБД та структура бази даних

Для зберігання даних використовується Firestore Cloud від Firebase. Ця NoSQL база даних забезпечує гнучкість, масштабованість і високу продуктивність, що є критично важливим для мобільних застосунків.

База даних складається з кількох колекцій, кожна з яких містить документи зі структурованими даними. Основні колекції:

1. **Колекція Users (Користувачі):**
   - ❖ Документ Username (унікальний ідентифікатор)
     Кожен документ представляє окремого користувача
     - Поле Name (Ім’я та прізвище клієнта)
     - Поле Phone (телефон)
     - Поле count (кількість зроблених замовлень)
2. **Колекція Waiters (Офіціанти):**
   - ❖ Документ Name (унікальний ідентифікатор)
     Кожен документ представляє окремого офіціанта
     - Поле CountVoted (Кількість голосів зібраних клієнтами. Можна проголосувати лише раз за замовлення)
     - Поле Image (Зображення)
3. **Колекція Menu (Меню):**
   - ❖ Документ name (назва категорії і унікальний ідентифікатор)
     Кожен документ представляє окрему категорію меню
     - Поле Image (Зображення)
       - Колекція Category (назва категорії)
         Кожна колекція представляє окрему категорію

❖ Документ Meal_name (унікальний ідентифікатор і назва страви)
Кожен документ представляє окрему страву

- Поле count (кількість замовленої однакової страви)

4. **Колекція «Замовлення»(виступає в ролі архівації замовлень):**
   ❖ Документ Table (унікальний ідентифікатор, номер столика)
   Кожен документ представляє окремий столик
   - Колекція Name_SumPrice (Ім’я користувача і загальна сума замовлення, також унікальний ідентифікатор)
     Кожна колекція представляє окреме замовлення від користувача
     ❖ Документ Name(Унікальний ідентифікатор і назва страви що замовили)
     Кожен документ представляє окрему страву
     - Поле count (кількість страв)
5. **Колекція Meal (Страва):**
   ❖ Документ Meal_name (унікальний ідентифікатор і назва страви)
   Кожен документ представляє окрему страву
   - Поле Description (опис страви)
   - Поле Image (посилання на зображення страви)
   - Поле Price(Ціна за одиницю)
6. **Колекція Table_number (Номер столика):**
   ❖ Документ Meal_name (Назва страви і унікальний ідентифікатор)
   Кожен документ представляє окрему страву
   - Поле count (кількість замовлених страв)

Взаємодія з базою даних

Операції з базою даних включають додавання, оновлення, видалення та читання документів. Приклад типових операцій на рисунках 3.1 - 3.4.

```
binding.finishBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Name_of_meal = binding.mealName.getText().toString();
        Map<String, Object> mealData = new HashMap<>();
        mealData.put("Description", binding.textDescription.getText().toString());
        mealData.put("Price", binding.textPrice.getText().toString());
        mealData.put("Image", Url);
        db.collection(category).document(Name_of_meal).set(mealData);
        finish();
    }
});
```

Рисунок 3.1 - Додавання нової страви до меню

```
db.collection(UserName).document(documentId).delete();
```

Рисунок 3.2 - Видалення замовлення з корзини

```
binding.finishBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Name_of_meal = binding.mealName.getText().toString();
        if (meal_name.equals(Name_of_meal)) {
            Map<String, Object> mealData = new HashMap<>();
            mealData.put("Description", binding.textDescription.getText().toString());
            mealData.put("Price", binding.textPrice.getText().toString());
            mealData.put("Image", Url);
            db.collection(category).document(Name_of_meal).set(mealData);
        }else {
            Map<String, Object> mealData = new HashMap<>();
            mealData.put("Description", binding.textDescription.getText().toString());
            mealData.put("Price", binding.textPrice.getText().toString());
            mealData.put("Image", Url);
            db.collection(category).document(Name_of_meal).set(mealData);
            db.collection(category).document(meal_name).delete();
        }
        finish();
    }
});
```

Рисунок 3.3 - Оновлення існуючої страви в меню

Тут описано 2 випадки:

1. **Якщо назву елемента меню хочуть змінити**

В колекції створюється новий документ з новою назвою, допустимо, страви, і записуються заново поля з попередньої страви. Старий документ з старою назвою видаляється. Видалення і запис зумовлено тим, що в Firestore Cloud не можна змінювати унікальний ідентифікатор документа, а можливо створити лише новий з новими полями.

2. **Якщо назву елемента меню не чіпають, а хочуть змінити решту полів.**

Тоді все просто. Ми просто перезаписуємо поля цього ж документу.

```
db.collection( collectionPath: "Menu") CollectionReference
        .get() Task<QuerySnapshot>
        .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
            @Override
            public void onComplete(@NonNull Task<QuerySnapshot> task) {
                if (task.isSuccessful()) {
                    for (QueryDocumentSnapshot document : task.getResult(   )) {
                        Category cat = new Category(document.getId(),document.getString( field: "Image"));
                        cats.add(cat);
                    }
                    if (progressDialog.isShowing()) {
                        progressDialog.dismiss();
                    }
                    menuRecycler.notifyDataSetChanged();
                } else {
                    Log.w(TAG,  msg: "Error getting documents.", task.getException());
                }
            }
        });
```

Рисунок 3.4 - Отримання всіх категорій меню

Безпека та контроль доступу

Firestore Cloud надає можливості для налаштування правил безпеки, що визначають, хто і які операції може виконувати з базою даних. Приклад правил безпеки:

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /Orders/{orderId} {
```

```
    allow read, write: if request.auth != null && request.auth.token.role == 'офіціант';
  }
  match /Users/{userId} {
    allow read, write: if request.auth != null && request.auth.token.role ==
'адміністратор';
  }
 }
}
```

3.3 Побудова об’єктно-орієнтованої моделі

Об’єктно-орієнтована модель (ООМ) допомагає відобразити структуру бази даних у вигляді класів та об’єктів в коді застосунку.

Вимоги до об’єктно-орієнтованої моделі

Перед побудовою об'єктно-орієнтованої моделі були визначені наступні вимоги:

- Модель повинна включати об'єкти, які представляють меню, замовлення, столики та користувачів.
- Кожен об'єкт повинен мати відповідні властивості та методи для взаємодії з базою даних Firestore Cloud.
- Модель повинна підтримувати основні операції CRUD (створення, читання, оновлення, видалення).

Класи та їхні атрибути

Клас Meal (Страва) - рисунок 3.5.

```
package com.example.myapplication;
42 usages
public class Meal {
    6 usages
    String Description, Image, Price,id, name, countSumPrice;
    7 usages
    String count = "1";
    public String getId() { return id; }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    1 usage
    public String getDescription() { return Description; }
    public String getPrice() { return Price; }
    7 usages
    public Meal(String Price, String name, String count) {
        this.Price = Price;
        this.name =name;
        this.count = count;
    }
    3 usages
    public Meal(String Description, String Price, String Image, String name) {
        this.Description = Description;
        this.Price = Price;
        this.Image = Image;
        this.name =name;
    }
    2 usages
    public Meal(String id, String count, String Description, String Price, String Image, String name, String countSumPrice) {
        this.count = count;
        this.Description = Description;
        this.Price = Price;
        this.Image = Image;
        this.id = id;
        this.name =name;
        this.countSumPrice= countSumPrice;
    }
     public String getCount() { return count; }
     public void setCount(String count) { this.count = count; }
     4 usages
     public String getUrl() {
         return Image;

     }

}
```

Рисунок 3.5 - Код класу Meal (Страва)

Клас Waiter (Офіціант) - рисунок 3.6.

```
package com.example.myapplication;
14 usages
public class Waiter {
    2 usages
    String Image, name, count;
    public String getCount() { return count; }
    public void setCount(String count) { this.count = count; }
    3 usages
    public Waiter(String Image, String name, String count) {
        this.Image = Image;
        this.name = name;
        this.count = count;
    }
    2 usages
    public String getImageUrl() { return Image; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}
```

Рисунок 3.6 - Код класу Waiter (Офіціант)

Клас Table (Столик) - рисунок 3.7.

```
1  package com.example.myapplication;
2
   6 usages
3  public class Table {
       3 usages
4      String number;
       1 usage
5      public Table(String number) {
6          this.number = number;
7      }
8  }
9
```

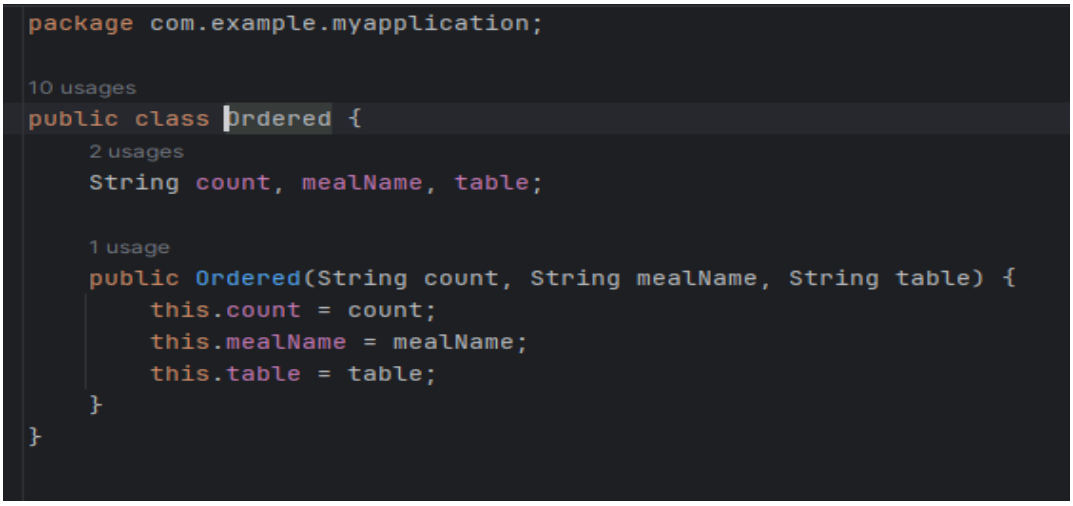
Рисунок 3.7 - Код класу Table (Столик)

Клас Category (Категорія) - рисунок 3.8.

```
package com.example.myapplication;

14 usages
public class Category {
    4 usages
    String cat_name;
    2 usages
    String Image;

    1 usage
    public Category(String cat_name, String Image) {

        this.cat_name = cat_name;
        this.Image = Image;
    }
    2 usages
    public Category(String cat_name) {

        this.cat_name = cat_name;
    }
    1 usage
    public String getUrl() {
        return Image;

    }
}
```

Рисунок 3.8 - Код класу Category (Категорія)

Клас Ordered (Замовлення) - рисунок 3.9.



```
package com.example.myapplication;

10 usages
public class Ordered {
    2 usages
    String count, mealName, table;

    1 usage
    public Ordered(String count, String mealName, String table) {
        this.count = count;
        this.mealName = mealName;
        this.table = table;
    }
}
```

Рисунок 3.9 - Код класу Ordered (Замовлення)

3.4 Опис програмної реалізації

Архітектура застосунку

Архітектура застосунку базується на патерні MVC (Model-View-Controller), що розділяє додаток на три основні компоненти: модель (Model), представлення (View) та контролер (Controller), що забезпечує гнучкість, розширюваність та зручність підтримки коду.

- **Model (Модель)**: відповідає за управління даними, логікою та правилами бізнесу. У цьому компоненті зберігаються та обробляються дані, які надходять з бази даних Firestore Cloud.
- **View (Представлення)**: відповідає за відображення даних користувачеві та прийом його введення. В Android Studio це зазвичай реалізується через Activity, Fragment та XML-розмітку.
- **Controller (Контролер)**: відповідає за зв'язок між моделлю та представленням. Контролер обробляє введення користувача, взаємодіє з моделлю для отримання та оновлення даних, а також оновлює представлення відповідно до змін у даних.

Основні компоненти застосунку

Model (Модель)

- Класи моделей: “Ordered”, “Category”, “Meal”,” Waiter”, “Table” (були показані вище)

View (Представлення)

- Активності:
- Макети XML для кожної активності.

Представлення відповідає за відображення даних користувачеві. Це включає Activity(на рисунках 3.12 - 3.14.), Fragment та XML-розмітку (на рисунках 3.10 - 3.11.). Представлення є максимально відокремлене від моделі та не містить бізнес-логіки.

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".Menu2"
    android:background="#F4EBC4">
    <ImageButton
        android:id="@+id/btn_back"
        android:layout_width="70dp"
        android:layout_height="70dp"
        android:src="@drawable/back_icn_foreground"
        android:background="@drawable/header"
        android:layout_margin="20dp"
        android:contentDescription="Назад" />
    <ImageButton
        android:id="@+id/cart"
        android:layout_width="70dp"
        android:layout_height="70dp"
        android:src="@drawable/back_icn_rounded_foreground"
        android:background="@drawable/header"
        android:layout_marginTop="20dp"
        android:layout_marginStart="100dp"
        android:contentDescription="Корзина" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="30dp"
        android:padding="25dp"
        android:id="@+id/MenuText"
        android:text="Меню"
        android:textColor="@color/milk"
        android:layout_alignParentEnd="true"
        android:textSize="30sp"
        android:background="@drawable/header"
        tools:ignore="RelativeOverlap" />
    <androidx.recyclerview.widget.RecyclerView
```

Рисунок 3.10 - Приклад коду для XML-розмітки активності “Menu2”

```
    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/menuCat"
        android:layout_width="match_parent"
        android:layout_height="600dp"
        android:layout_marginTop="150dp"
        android:layout_marginHorizontal="15dp"
        android:padding="5dp"
        app:layoutManager="androidx.recyclerview.widget.GridLayoutManager"
        app:spanCount="3"/>
    <androidx.appcompat.widget.AppCompatButton
        android:layout_width="match_parent"
        android:id="@+id/Loyalty"
        android:layout_margin="50dp"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:textColor="#F4EBC4"
        android:background="@drawable/btns_menu"
        android:text="Карта лояльності"
        android:textStyle="bold"/>
</RelativeLayout>
```

Рисунок 3.11 - продовження прикладу коду для XML-розмітки активності “Menu2”

```
public class Menu2 extends AppCompatActivity {
    8 usages
    ActivityMenu2Binding binding;
    3 usages
    ArrayList<Category> cats = new ArrayList<>();
    2 usages
    FirebaseFirestore db;
    3 usages
    MenuRecycler menuRecycler;
    6 usages
    ProgressDialog progressDialog;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMenu2Binding.inflate(getLayoutInflater());
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(binding.getRoot());

        progressDialog = new ProgressDialog( context: this);
        progressDialog.setCancelable(false);
        progressDialog.setMessage("Fetching Data...");
        progressDialog.show();
```

Рисунок 3.12 - Приклад коду активності “Menu2”

```
binding.menuCat.setLayoutManager(new GridLayoutManager( context: this,
        spanCount: 3,RecyclerView.VERTICAL, reverseLayout: false));
binding.menuCat.setAdapter(new MenuRecycler(cats, context: this));

db = FirebaseFirestore.getInstance();
menuRecycler = new MenuRecycler(cats, context: Menu2.this);

binding.menuCat.setAdapter(menuRecycler);

binding.btnBack.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent( packageContext: Menu2.this,MainActivity.class));
    }
});
binding.cart.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent( packageContext: Menu2.this,Cart.class));
    }
});
binding.Loyalty.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent( packageContext: Menu2.this,loyalty.class));
    }
});

db.collection( collectionPath: "Menu") CollectionReference
        .get() Task<QuerySnapshot>
        .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
            @Override
```

Рисунок 3.13 - продовження прикладу коду активності “Menu2”

```
        db.collection( collectionPath: "Menu") CollectionReference
                .get() Task<QuerySnapshot>
                .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                    @Override
                    public void onComplete(@NonNull Task<QuerySnapshot> task) {
                        if (task.isSuccessful()) {
                            for (QueryDocumentSnapshot document : task.getResult(   )) {
                                Category cat = new Category(document.getId(),document.getString( field: "Image"));
                                cats.add(cat);
                            }
                            // Hide progress dialog after data is processed
                            if (progressDialog.isShowing()) {
                                progressDialog.dismiss();
                            }
                            menuRecycler.notifyDataSetChanged(); // Update RecyclerView adapter
                        } else {
                            Log.w(TAG,  msg: "Error getting documents.", task.getException());
                            // Handle errors if data retrieval fails (optional)
                        }
                    }
                });


        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }
}
```

Рисунок 3.14 - продовження прикладу коду активності “Menu2”

Controller (Контроллер)

- Контролер забезпечує взаємодію між моделлю та представленням. Він обробляє події користувача та оновлює модель та представлення відповідно до змін.

Взаємодія між компонентами

Компоненти моделі, представлення та контролера взаємодіють наступним чином:

- Користувач взаємодіє з представленням (View), наприклад, натискаючи кнопки або вводячи дані.
- Представлення передає події контролеру (Controller).
- Контролер обробляє події, взаємодіє з моделлю (Model) для отримання або оновлення даних, а потім оновлює представлення відповідно до змін у моделі.

Аутентифікація користувачів:

Використання Firebase Cloud для реєстрації та входу користувачів на рисунках 3.15 - 3.17.

```
public class MainActivity extends AppCompatActivity {
    3 usages
    FirebaseFirestore db;
    8 usages
    ActivityMainBinding binding;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMainBinding.inflate(getLayoutInflater());
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(binding.getRoot());

        db = FirebaseFirestore.getInstance();

binding.btnCon.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if(binding.Name.getText().toString().isEmpty()
                ||binding.Surname.getText().toString().isEmpty()){
            Toast.makeText( context: MainActivity.this
                    , text: "Напишіть своє ім'я і прізвище!!", Toast.LENGTH_LONG).show();
        } else {
            String User = binding.Name.getText().toString()+binding.Surname.getText().toString();
            String User2 = User.replaceAll( regex: "\\s", replacement: "");
```

Рисунок 3.15 - Приклад коду активності “MainActivity” для реєстрації та входу

```
            String User = binding.Name.getText().toString()+binding.Surname.getText().toString();
            String User2 = User.replaceAll( regex: "\\s",  replacement: "");
            Intent intent = new Intent(v.getContext(), Menu2.class);
            intent.putExtra( name: "username", User2);
            v.getContext().startActivity(intent);
            MySingleton singleton = MySingleton.getInstance();
            singleton.setVariable(User2);

            db.collection( collectionPath: "Users") CollectionReference
                    .whereEqualTo( field: "Name",User2) Query
                    .get() Task<QuerySnapshot>
                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                        @Override
                        public void onComplete(@NonNull Task<QuerySnapshot> task) {
                                if (task.getResult().isEmpty()) {
                                    Map<String, Object> UserData = new HashMap();
                                    UserData.put("Name",User2);
                                    UserData.put("Phone","");
                                    UserData.put("count","0");
                                    db.collection( collectionPath: "Users").document(User2).set(UserData);
                                }
                        }
                    });
        }
    }
});
binding.admBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivity(new Intent( packageContext: MainActivity.this,Adm_form.class));
    }
});


        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }
}
```

Рисунок 3.16 - Продовження прикладу коду активності “MainActivity” для реєстрації та входу

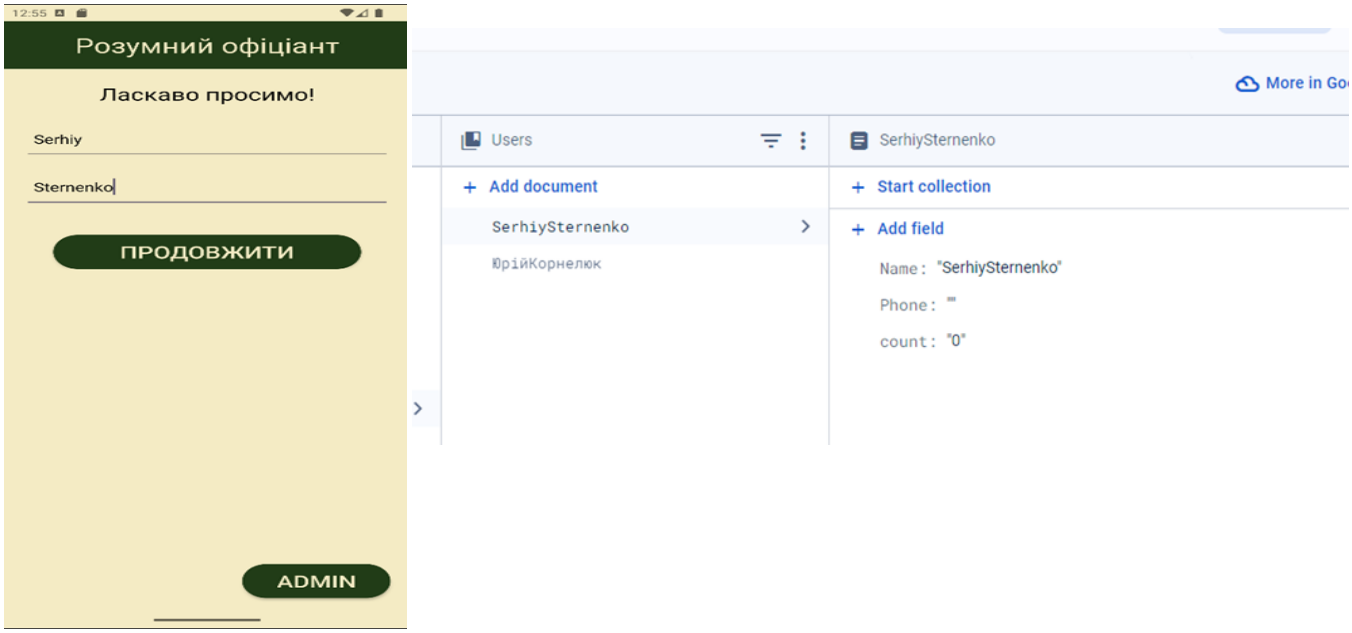


Рисунок 3.17 - Вигляд сторінки активності “MainActivity” для реєстрації та входу в додатку

Подача та обробка замовлень:

Клієнт сідає за вільний столик та додає страви з меню до замовлення. Переглянути на рисунку 3.18 - 3.22.

Рисунок 3.18 - Прийом та обробка замовлень в додатку

```
holder.buy_btn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        vibrator.vibrate( milliseconds: 100);
        HiddenSumPrice = Double.valueOf(meal.getPrice())*Integer.parseInt(meal.getCount());
        String HiddenSumPrice2=String.valueOf(HiddenSumPrice);

        int position = holder.getAdapterPosition();
        Meal meal = meals.get(position);
        Map<String, Object> mealData = new HashMap<>();
        mealData.put("name",meal.getName());
        mealData.put("description", meal.getDescription());
        mealData.put("price", meal.getPrice());
        mealData.put("count", meal.getCount());
        mealData.put("url", meal.getUrl());
        mealData.put("hiddenPrice", HiddenSumPrice2);

        MySingleton singleton = MySingleton.getInstance();
        String UserName = singleton.getVariable();
       db.collection(UserName).whereEqualTo( field: "name", meal.getName())
               .get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
           @Override
           public void onComplete(@NonNull Task<QuerySnapshot> task) {
               if (task.isSuccessful()){
                   if (task.getResult().isEmpty()){
                       db.collection(UserName) CollectionReference
                               .add(mealData)  // Add only the clicked meal
                               .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                                   @Override
```

Рисунок 3.19 - Код кнопки додавання в корзину страви

```
@Override
public void onComplete(@NonNull Task<QuerySnapshot> task) {
    if (task.isSuccessful()){
        if (task.getResult().isEmpty()){
            db.collection(UserName) CollectionReference
                    .add(mealData)  // Add only the clicked meal
                    .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
                        @Override
                        public void onSuccess(DocumentReference documentReference) {
                            Log.d(TAG,  msg: "Meal added with ID: " + documentReference.getId());

                            Map<String, Object> updates = new HashMap<>();
                            updates.put("id",documentReference.getId());
                            documentReference.update(updates)
                                    .addOnSuccessListener(new OnSuccessListener<Void>() {
                                        @Override
                                        public void onSuccess(Void unused) {
                                            Log.d(TAG,  msg: "Document updated");
                                            Toast.makeText(context, text: "Додано в кошик!", Toast.LENGTH_LONG).show();

                                        }
                                    })
                                    .addOnFailureListener(new OnFailureListener() {
                                        @Override
                                        public void onFailure(@NonNull Exception e) {
                                            Log.w(TAG,  msg: "Error updating document", e);
                                        }
                                    });
                        }
                    }) Task<DocumentReference>
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
```

Рисунок 3.20 - Продовження коду кнопки

```
                                    .addOnFailureListener(new OnFailureListener() {
                                        @Override
                                        public void onFailure(@NonNull Exception e) {
                                            Log.w(TAG,  msg: "Error updating document", e);
                                        }
                                    });
                        }
                    }) Task<DocumentReference>
                    .addOnFailureListener(new OnFailureListener() {
                        @Override
                        public void onFailure(@NonNull Exception e) {
                            Log.w(TAG,  msg: "Error adding meal", e);
                        }
                    });
        }else {
            Toast.makeText(context, text: "Страва вже додана до корзини! Кількість збільшено!", Toast.LENGTH_LONG).show();
            db.collection(UserName) CollectionReference
                    .whereEqualTo( field: "name", meal.getName()) Query
                    .get() Task<QuerySnapshot>
                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull Task<QuerySnapshot> task) {
                    if (task.isSuccessful()){
                        DocumentSnapshot document = task.getResult().getDocuments().get(0);
                            int count = Integer.parseInt(document.getString( field: "count"))+1;
                            double hidPr=count*Double.valueOf(document.getString( field: "price"));

                            db.collection(UserName).document(document.getId()).update( field: "hiddenPrice",String.valueOf(hidPr));
                            db.collection(UserName).document(document.getId()).update( field: "count",String.valueOf(count));

                    }
                }
            });
        }
    }
}
});
```

Рисунок 3.21 - Продовження коду кнопки

```
    @Override
    public int getItemCount() { return meals.size(); }

    4 usages
    static class ViewHolder extends RecyclerView.ViewHolder{
        2 usages
        TextView Description,  Price, count, name;
        2 usages
        AppCompatButton plus_btn, minus_btn;
        2 usages
        ImageButton buy_btn;
        2 usages
        ImageView Image;

        1 usage
        public ViewHolder( View itemView) {
            super(itemView);
            Description = itemView.findViewById(R.id.text_description);
            Image = itemView.findViewById(R.id.image_meal);
            Price = itemView.findViewById(R.id.text_price);
            count = itemView.findViewById(R.id.count_text);
            plus_btn = itemView.findViewById(R.id.plus_btn);
            minus_btn = itemView.findViewById(R.id.minus_btn);
            buy_btn = itemView.findViewById(R.id.buy_btn);
            name = itemView.findViewById(R.id.meal_name);

        }

    }
}
```

Рисунок 3.22 - Продовження коду кнопки

Після клієнт перевіряє чи все в замовленні вірно і обирає столик за яким сидить та спосіб оплати. Замовлення зберігається у Firestore Cloud та передається на кухню. Зображено на рисунку 3.23.

```
binding.payBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String sumprice = binding.SumPrice.getText().toString();
        String tableNum = String.valueOf( i: Integer.parseInt(binding.numOfTable.getSelectedItem().toString())+1);
            db.collection(UserName) CollectionReference
                    .get() Task<QuerySnapshot>
                    .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
                        @Override
                        public void onComplete(@NonNull Task<QuerySnapshot> task) {
                            if (task.isSuccessful()) {
                                for (QueryDocumentSnapshot document : task.getResult()) {
                                    Map<String, Object> mealData = new HashMap<>();
                                    mealData.put("count",document.getString( field: "count"));
                                    db.collection(tableNum)
                                            .document(document.getString( field: "name")).set(mealData);
                                    ordersRef.document(tableNum).collection( collectionPath: UserName+"_"+sumprice)
                                            .document(document.getString( field: "name").toString()).set(mealData);
                                }
                            }
                        }
                    });
        startActivity(new Intent( packageContext: SumOrder.this, Payment.class));
    }
});
```

Рисунок 3.23 - Запис замовлення в базу даних Firestore Cloud

На рисунку 3.24 - 3.25 видно, що замовлення для столику №5 сформоване.

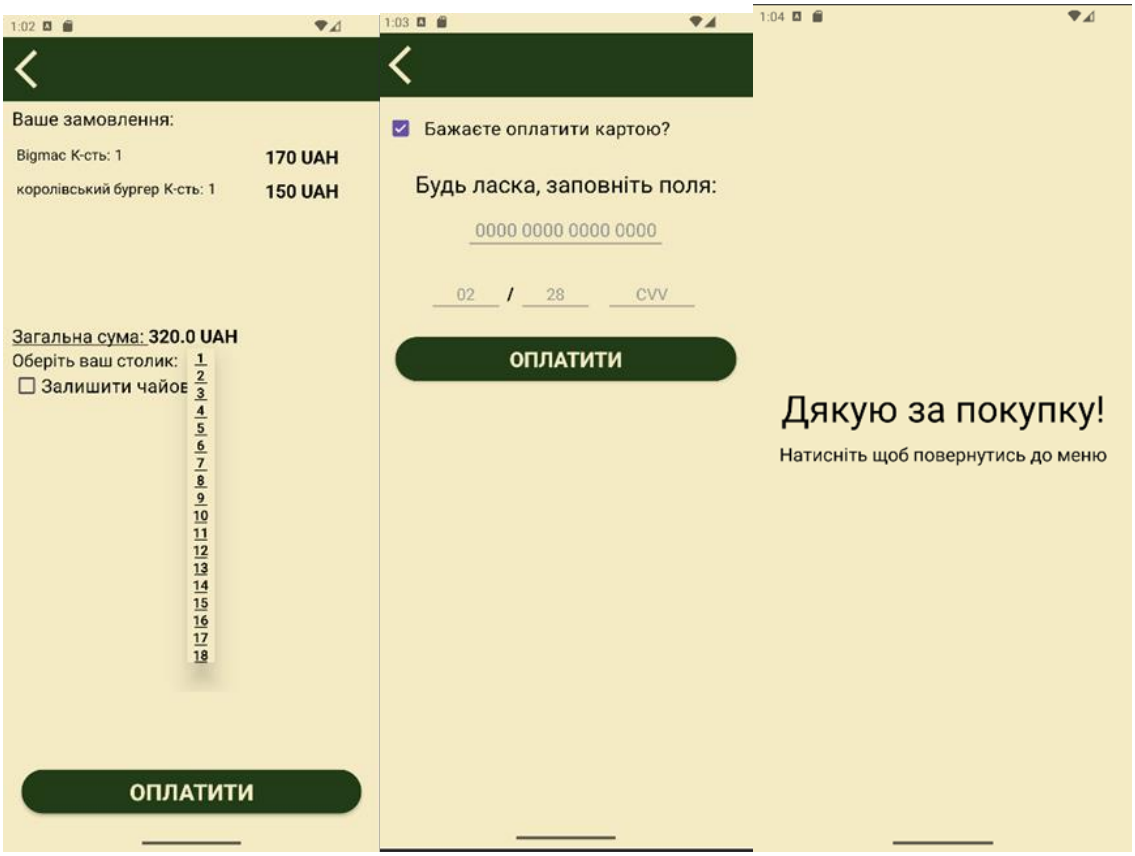


Рисунок 3.24 - Процес оформлення замовлення в додатку

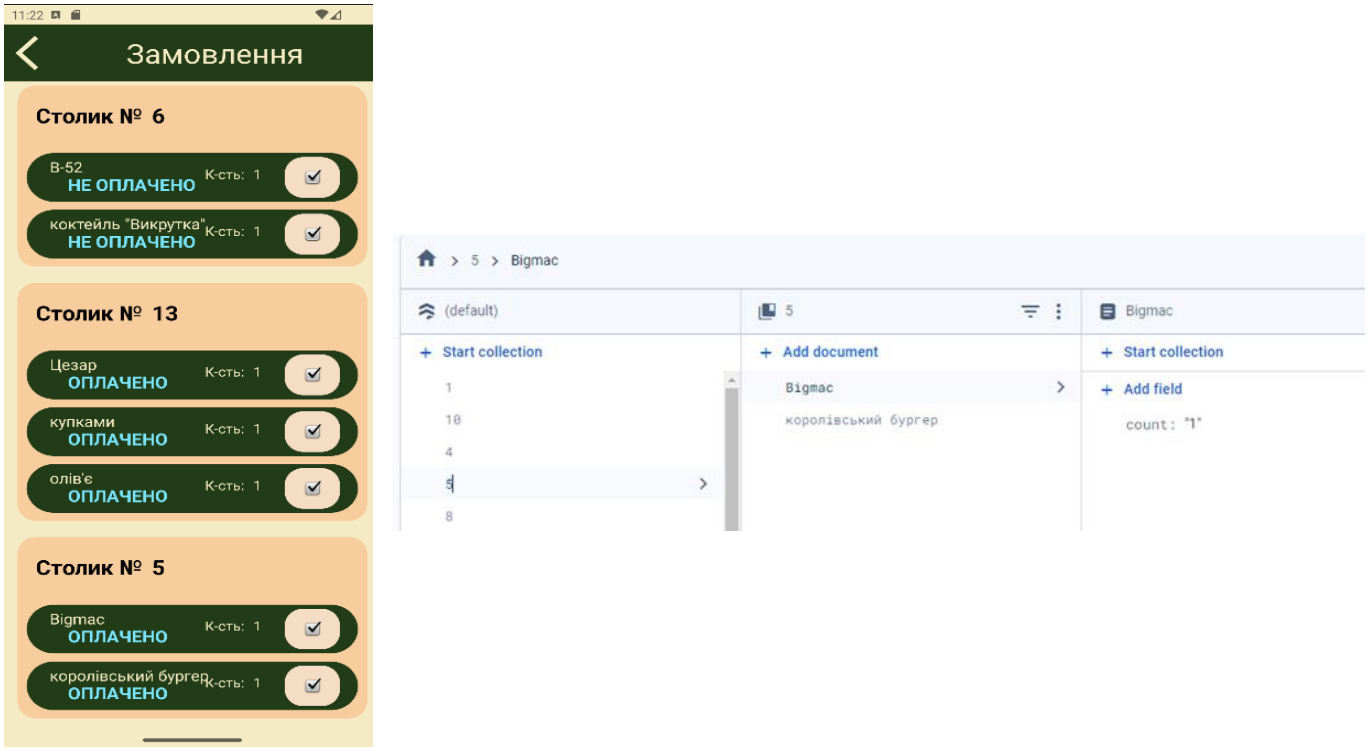


Рисунок 3.25 - Вигляд оформленого замовлення

Адміністрування бази даних

За допомогою логіну та паролю ми можемо ввійти в масив активностей для адміністрування наявної бази даних в Firestore Cloud. Можна змінювати меню та перелік офіціантів. Також в адмініструванні можна відслідковувати готові та не готові замовлення. Показано на рисунках 3.26 - 3.28.

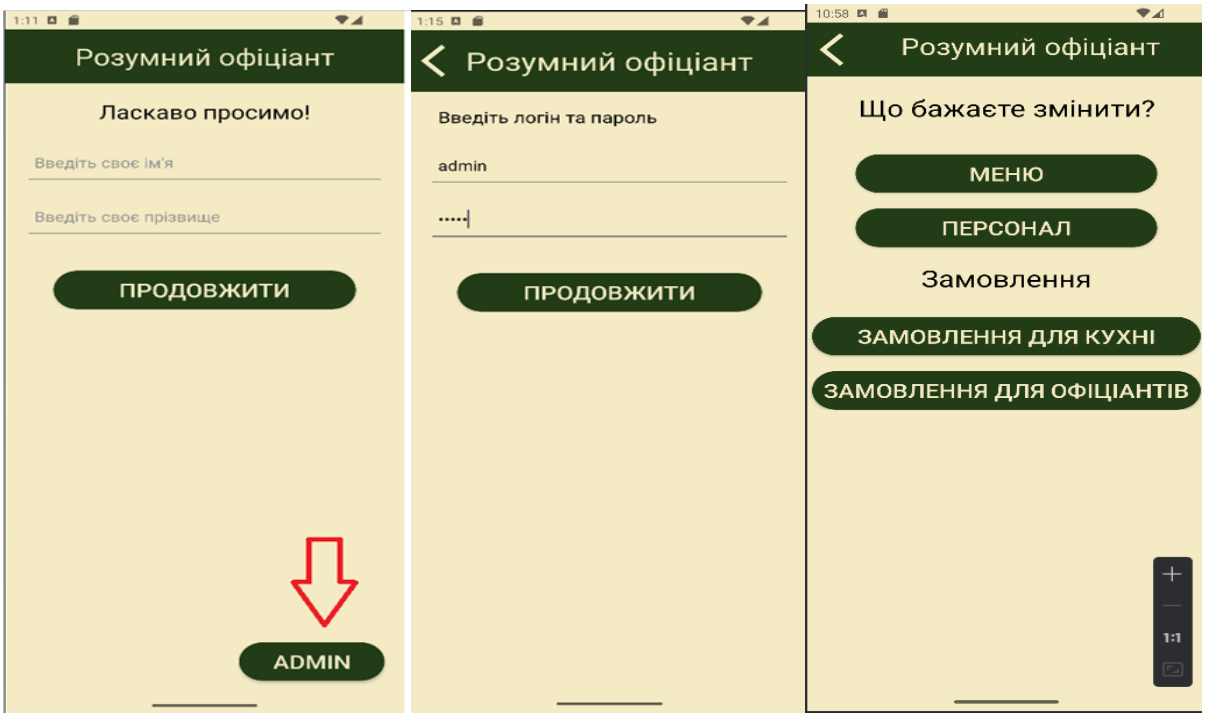


Рисунок 3.26 - Вхід до адміністрування через логін та пароль

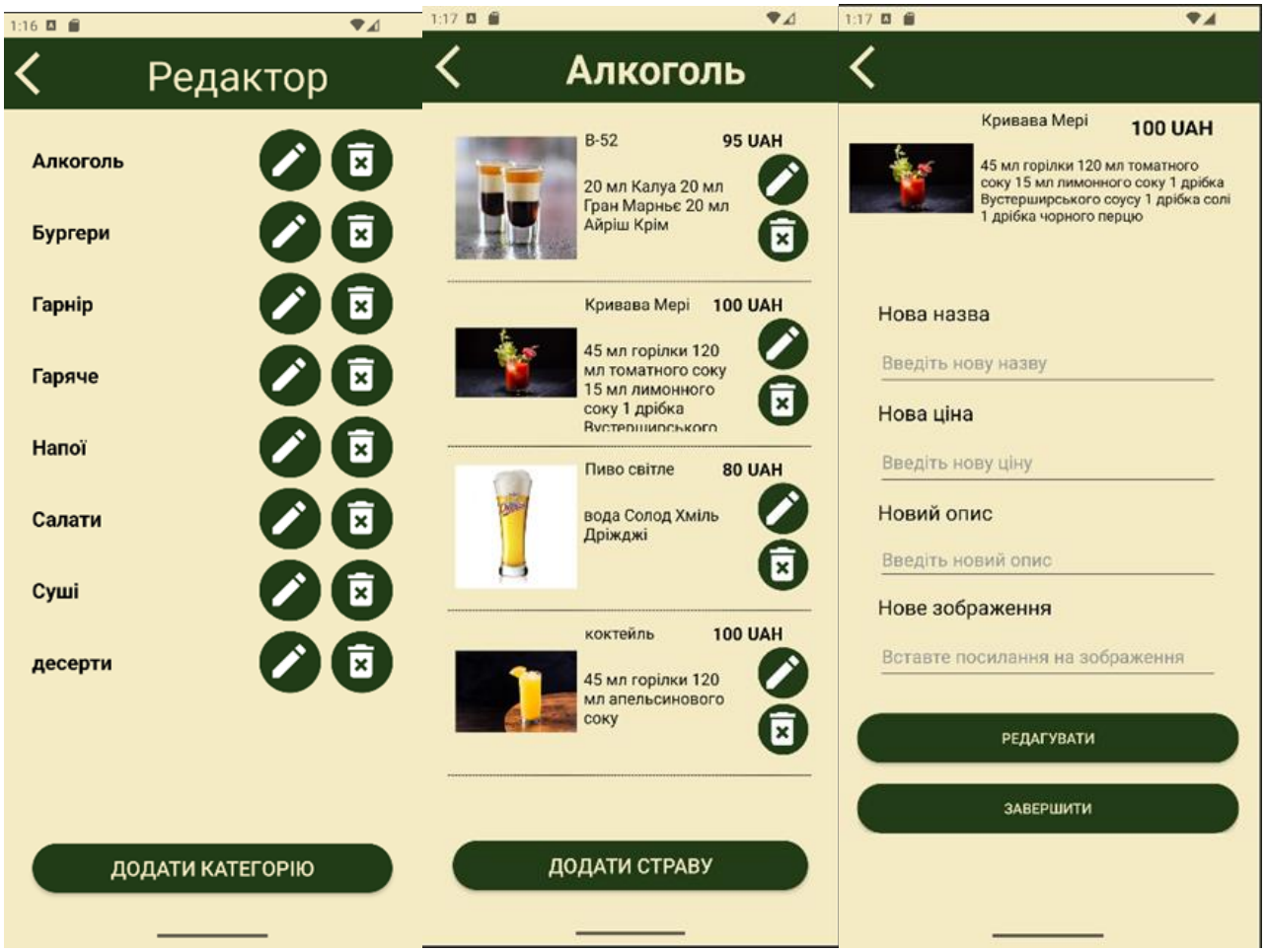


Рисунок 3.27 - Адміністрування меню та страв в застосунку

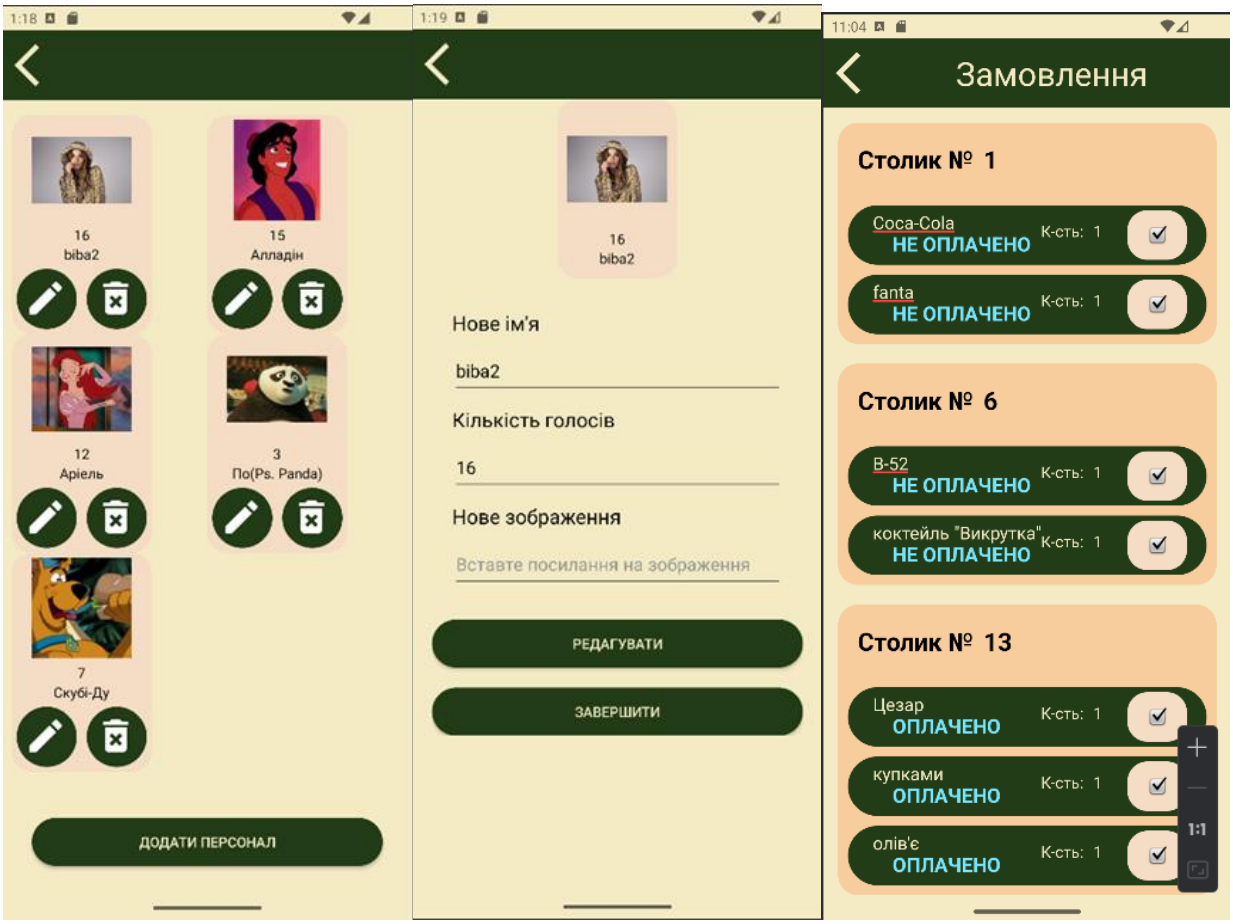


Рисунок 3.28 - Редагування колекції офіціантів та список замовлень для готування

Приклад коду активності для редагування чи створення страви або напою на рисунку 3.29 - 3.31.

```
public class MealEditor extends AppCompatActivity {
36 usages
ActivityMealEditorBinding binding;
4 usages
String Url;
5 usages
String Name_of_meal;
5 usages
FirebaseFirestore db;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityMealEditorBinding.inflate(getLayoutInflater());
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(binding.getRoot());

        db = FirebaseFirestore.getInstance();
        Intent intent = getIntent();
        String meal_name = intent.getStringExtra( name: "meal_name");

        MySingleton singleton = MySingleton.getInstance();
        String category = singleton.getVariable();
        binding.btnBack.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();           }
        });

        db.collection(category) CollectionReference
                .document(meal_name) DocumentReference
                .get() Task<DocumentSnapshot>
                .addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>()
                {
                    @Override
                    public void onComplete(@NonNull Task<DocumentSnapshot> task)
                    {
                        if(task.isSuccessful())
```

Рисунок 3.29 - Код активності MealEditor для редагування страви чи напою

```
            if(task.isSuccessful())
            {
                if (task.getResult().exists())
                {
                    DocumentSnapshot document = task.getResult();
                    String Image = document.getString( field: "Image");
                    Url = Image;
                    String description = document.getString( field: "Description");
                    String price = document.getString( field: "Price");
                    binding.mealName.setText(meal_name);
                    binding.textPrice.setText(price);
                    binding.textDescription.setText(description);
                    Glide.with( activity: MealEditor.this) RequestManager
                            .load(Image) RequestBuilder<Drawable>
                            .into(binding.imageMeal);
                }
            }
        }
    });

binding.editBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (binding.newMealName.getText().toString().isEmpty()
                && binding.newMealDescription.getText().toString().isEmpty()
                && binding.newMealPrice.getText().toString().isEmpty()
        && binding.newMealImage.getText().toString().isEmpty()){
            Toast.makeText( context: MealEditor.this, text: "Заповніть хоча б одне поле!", Toast.LENGTH_LONG).show();
        }else if(!binding.newMealName.getText().toString().isEmpty()
                || !binding.newMealDescription.getText().toString().isEmpty()
                || !binding.newMealPrice.getText().toString().isEmpty()
                || !binding.newMealImage.getText().toString().isEmpty()){
            if (!binding.newMealName.getText().toString().isEmpty()){
                Name_of_meal = binding.newMealName.getText().toString();
                binding.mealName.setText(binding.newMealName.getText());
            }
            if (!binding.newMealPrice.getText().toString().isEmpty()){
                binding.textPrice.setText(binding.newMealPrice.getText());
            }
```

Рисунок 3.30 - Продовження коду активності MealEditor

```
            if (!binding.newMealDescription.getText().toString().isEmpty()){
                binding.textDescription.setText(binding.newMealDescription.getText());
            }
            if (!binding.newMealImage.getText().toString().isEmpty()){
                Url = binding.newMealImage.getText().toString();
                Glide.with( activity: MealEditor.this) RequestManager
                        .load(binding.newMealImage.getText().toString()) RequestBuilder<Drawable>
                        .into(binding.imageMeal);
            }
        }
    }
});

binding.finishBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Name_of_meal = binding.mealName.getText().toString();
        if (meal_name.equals(Name_of_meal)) {
            Map<String, Object> mealData = new HashMap<>();
            mealData.put("Description", binding.textDescription.getText().toString());
            mealData.put("Price", binding.textPrice.getText().toString());
            mealData.put("Image", Url);
            db.collection(category).document(Name_of_meal).set(mealData);
        }else {
            Map<String, Object> mealData = new HashMap<>();
            mealData.put("Description", binding.textDescription.getText().toString());
            mealData.put("Price", binding.textPrice.getText().toString());
            mealData.put("Image", Url);
            db.collection(category).document(Name_of_meal).set(mealData);
            db.collection(category).document(meal_name).delete();
        }
        finish();
    }
});
```

Рисунок 3.31 - Продовження коду активності MealEditor

В кожному елементі, де присутнє перелічення документів з Firestore Cloud типу списку категорій меню чи страв, я використовував компонент користувацького інтерфейсу RecyclerView, призначений для прокручуваних списків. Для кожного такого компонента описував адаптер, який використовується для зв’язку XML макету та даних. Приклад одного з таких адаптерів на рисунках 3.32 - 3.33.

```
public class OrderedRecycler extends RecyclerView.Adapter<OrderedRecycler.ViewHolder>{
    2 usages
    private Vibrator vibrator;
    4 usages
    ArrayList<Meal> meals;
    4 usages
    Context context;
    1 usage
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    2 usages
    public OrderedRecycler(ArrayList<Meal> meals, Context context) {
        this.meals = meals;
        this.context = context;
    }
    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        return new OrderedRecycler.ViewHolder(LayoutInflater
                .from(context).inflate(R.layout.ordered_list, parent, attachToRoot: false));
    }
    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        Meal meal = meals.get(position);
        vibrator = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVICE);
        holder.name.setText(meal.name);
        holder.Description.setText(meal.Description);
        holder.hiddenPrice.setText(meal.countSumPrice);
        holder.count.setText(meal.count);
        holder.Price.setText(meal.Price);
        String imageUrl = meal.getUrl();
        Glide.with(context) RequestManager
                .load(imageUrl) RequestBuilder<Drawable>
                .into(holder.Image);

        holder.remove_btn.setOnClickListener(new View.OnClickListener() {
            @Override
```

Рисунок 3.32 - Приклад коду адаптера для RecyclerView компонента

```
            @Override
            public void onClick(View v) {
                vibrator.vibrate( milliseconds: 100);
                String documentId = meal.id;

                MySingleton singleton = MySingleton.getInstance();
                String UserName = singleton.getVariable();

                db.collection(UserName).document(documentId).delete();
                int index = holder.getAdapterPosition();
                meals.remove(index);

                notifyItemRemoved(index);
            }
        });
    }
    @Override
    public int getItemCount() { return meals.size(); }
    4 usages
    static class ViewHolder extends RecyclerView.ViewHolder{
        2 usages
        TextView Description,  Price, count, name, hiddenPrice;
        2 usages
        ImageButton remove_btn;
        2 usages
        ImageView Image;
        1 usage
        public ViewHolder( View itemView) {
            super(itemView);
            Description = itemView.findViewById(R.id.text_description);
            Image = itemView.findViewById(R.id.image_meal);
            Price = itemView.findViewById(R.id.text_price);
            count = itemView.findViewById(R.id.count_text);
            remove_btn = itemView.findViewById(R.id.remove_btn);
            name = itemView.findViewById(R.id.meal_name);
            hiddenPrice = itemView.findViewById(R.id.hiddenPrice);
        }
    }
}
```

Рисунок 3.33 - Продовження коду адаптера для RecyclerView компонента

Також була розроблена система лояльності, яка має прив’язку за номером телефону і автоматично встановлює певний відсоток знижки на загальну суму оплати залежно від кількості зроблених замовлень. На рисунках 3.34 - 3.35 ви можете побачити активності реєстрації в програму лояльності та можливість змінити номер телефону. Також прикріплено код активності(рисунки 3.36 - 3.38.).

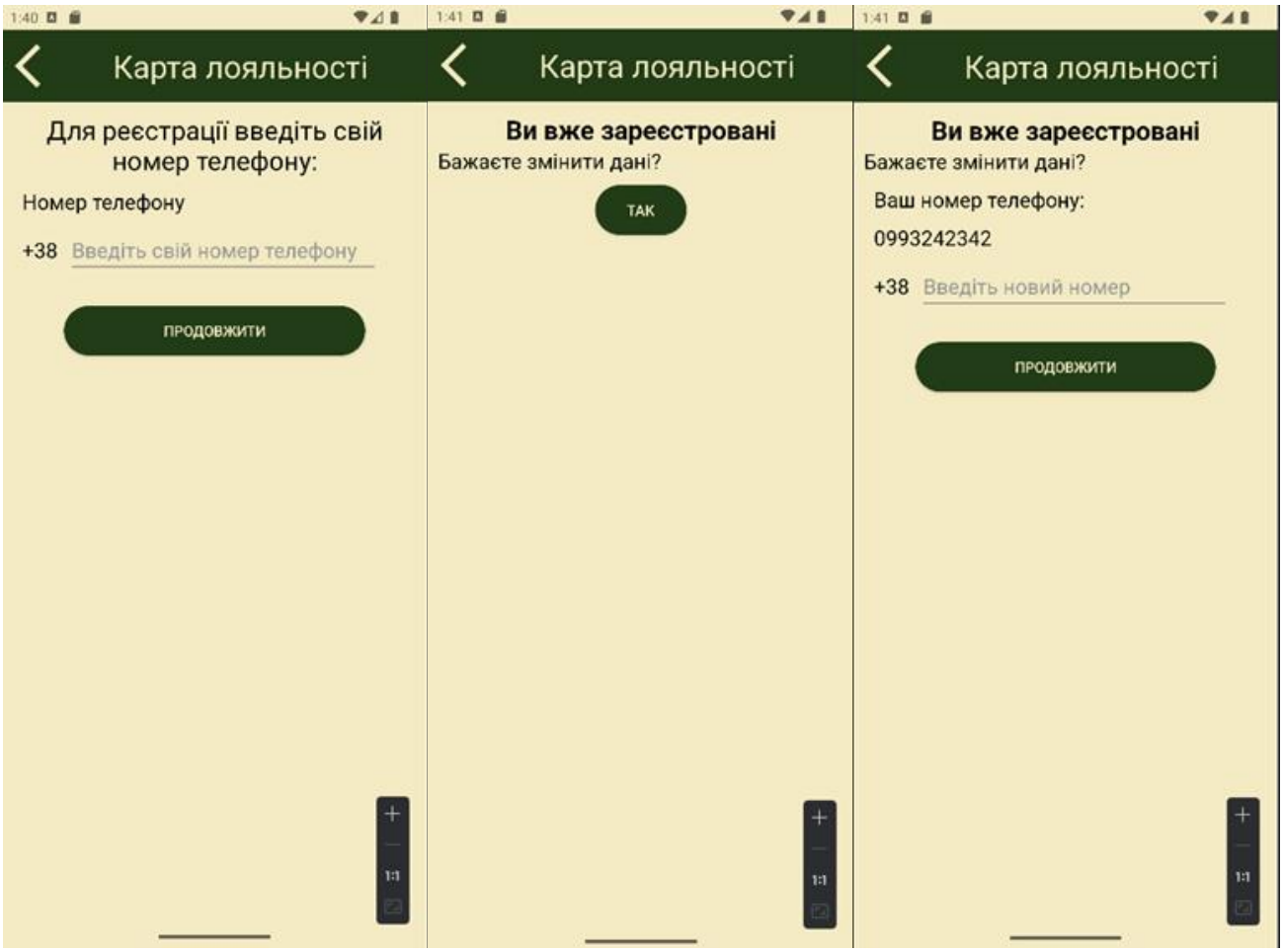


Рисунок 3.34 - Реєстрація в програму лояльності та можливість зміни номеру

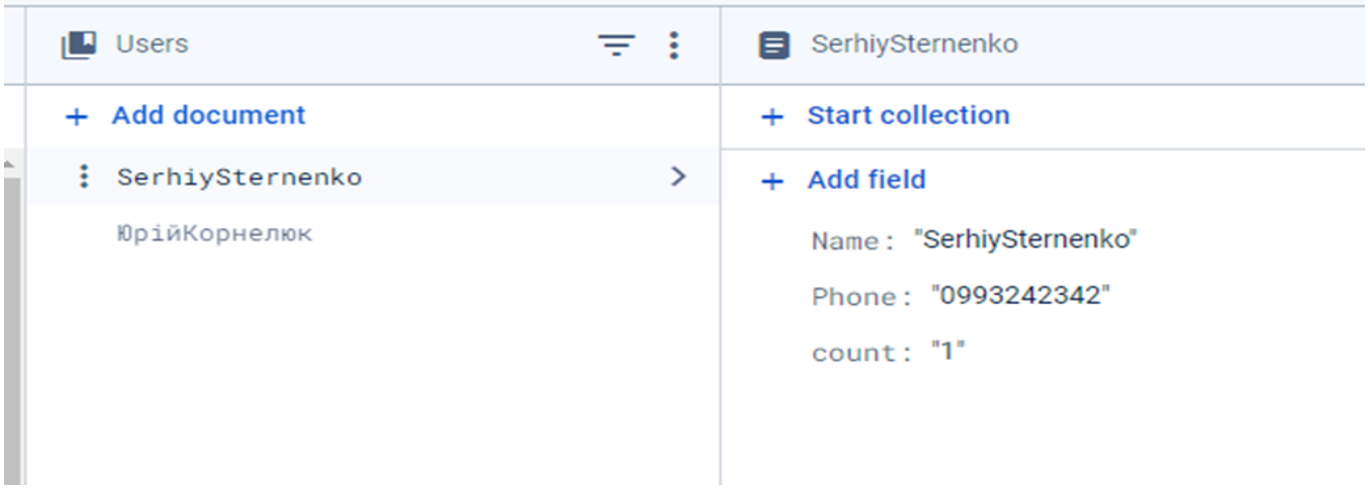


Рисунок 3.35 - Запис номера користувача в базу даних Firestore Cloud

```
public class loyalty extends AppCompatActivity {
21 usages
ActivityLoyaltyBinding binding;
5 usages
FirebaseFirestore db;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityLoyaltyBinding.inflate(getLayoutInflater());
        EdgeToEdge.enable( $this$enableEdgeToEdge: this);
        setContentView(binding.getRoot());


        db = FirebaseFirestore.getInstance();

        MySingleton singleton = MySingleton.getInstance();
        String UserName = singleton.getVariable();

        db.collection( collectionPath: "Users").document(UserName) DocumentReference
                .get() Task<DocumentSnapshot>
                .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
            @Override
            public void onSuccess(DocumentSnapshot documentSnapshot) {
                if (documentSnapshot.getString( field: "Phone").equals("")
                        ||documentSnapshot.getString( field: "Phone").isEmpty()){
                    binding.reg.setVisibility(View.VISIBLE);
                    binding.LogedIn.setVisibility(View.GONE);
                    binding.EditForm.setVisibility(View.GONE);
                }else{
                    binding.reg.setVisibility(View.GONE);
                    binding.LogedIn.setVisibility(View.VISIBLE);
                    binding.finish.setVisibility(View.INVISIBLE);
                    binding.CurrnetPhone.setText(documentSnapshot.getString( field: "Phone"));
                }
            }
        });
```

Рисунок 3.36 - Код активності “loyalty” для програми лояльності

```
binding.btnBack.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { finish(); }
});

binding.HellYeah.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        binding.EditForm.setVisibility(View.VISIBLE);
        binding.finish.setVisibility(View.VISIBLE);
        binding.HellYeah.setVisibility(View.GONE);
    }
});
binding.finish.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        db.collection( collectionPath: "Users").document(UserName) DocumentReference
                .get() Task<DocumentSnapshot>
                .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
                    @Override
                    public void onSuccess(DocumentSnapshot documentSnapshot) {
```

Рисунок 3.37 - Продовження коду активності “loyalty”

```
                    public void onSuccess(DocumentSnapshot documentSnapshot) {
                        if (documentSnapshot.getString( field: "Phone").isEmpty()
                                ||documentSnapshot.getString( field: "Phone").equals("")){
                            if (!binding.NumEditText2.getText().toString().isEmpty()){
                                Map<String,Object> UserData = new HashMap<>();
                                UserData.put("Phone", binding.NumEditText2.getText().toString());
                                db.collection( collectionPath: "Users").document(UserName).update(UserData);
                                finish();
                            }else{
                                Toast.makeText( context: loyalty.this, text: "Заповніть поле!", Toast.LENGTH_LONG).show();
                            }
                        }else if (!documentSnapshot.getString( field: "Phone").isEmpty()){
                            if (binding.NumEditText.getText().toString().isEmpty()){
                                Toast.makeText( context: loyalty.this, text: "Заповніть поле!", Toast.LENGTH_LONG).show();
                            }else if(binding.NumEditText.getText().toString()
                                    .equals(binding.CurrnetPhone.getText().toString())){
                                Toast.makeText( context: loyalty.this, text: "Ви ввели той самий номер!"
                                        , Toast.LENGTH_LONG).show();
                            }else {
                                Map<String,Object> UserData = new HashMap<>();
                                UserData.put("Phone", binding.NumEditText.getText().toString());
                                db.collection( collectionPath: "Users").document(UserName).update(UserData);
                                finish();
                            }

                        }
                    }
                });
        }
    });

    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}
}
```

Рисунок 3.38 - Продовження коду активності “loyalty”

ВИСНОВОК

В результаті виконання дипломної роботи бакалавра було розроблено Android-застосунок «Розумний офіціант», метою якого є оптимізація і автоматизація системи обслуговування певного закладу харчування, зменшення навантаження на працівників таких закладів та зменшення кількості помилок при обслуговуванні клієнтів.

Розроблений додаток для мобільних пристроїв на базі операційної системи Android.

У проєкті з використанням бібліотеки Glide, мови програмування Java та різних допоміжних інструментів середовища Android Studio було реалізовано масив сторінок(екранів) додатка з відповідним функціоналом і дизайном для зручності користувачів. З допомогою хмарної бази даних NoSQL Firestore Cloud будь-які нові дані оперативно зберігаються в хмарі для ефективної роботи застосунку і синхронізуються з додатком . За допомогою бібліотеки Glide було додано масив зображень для покращення візуальної складової та зручності вибору страви чи напою.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Kathy Sierra, Trisha Gee & Bert Bates. Head First Java: A Brain-Friendly Guide 3rd Edition. - O'Reilly Media, 2022. - 720 с.;
2. Dawn Griffiths, David Griffiths. Head First Android Development: A Brain-Friendly Guide 3rd Edition. - O'Reilly Media, 2021. - 800 с.;
3. FoxmindEd [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/shcho-potribno-znaty-android-rozrobnyku/ (Дата звернення: 08.06.2024);
4. Cloud Firestore Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore (Дата звернення: 08.06.2024);
5. Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/firestore#all-features (Дата звернення: 08.06.2024);
6. Gradle [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/build/gradle-tips (Дата звернення: 08.06.2024);
7. Mainak Mitra. Mastering Gradle. - Packt Pub Ltd, 2015. - 284 с.;
8. Android | build.gradle [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/android-build-gradle/ (Дата звернення: 08.06.2024);
9. Бібліотека Glide [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/bumptech/glide (Дата звернення: 08.06.2024);
10. Підручник із використання Glide [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/image-loading-caching-library-android-set-2/ (Дата звернення: 08.06.2024);
11. Erik T. Ray. Learning XML. - O'Reilly Media, 2001. - 368 с.;
12. Md Yusuf Hossein Khan & Afzal Hossain. The Effect of ICT Application on the Tourism and Hospitality Industries in London. - ARMG Publishing, 2020. - 68 с.;
13. Віктор Архіпов. Організація обслуговування в закладах ресторанного господарства. - ЦУЛ, 2021. - 342 с.;

14. О. В. Олійник, А. В. Шестакова, Д. І. Ярмолюк. Напрями цифровізації ресторанного бізнесу. (2023). [Електронний ресурс] – Режим доступу до ресурсу: https://doi.org/10.26642/ema-2023-1(103)-15-21 (Дата звернення: 09.06.2024);
15. Juan Llopis, Jose Gasco & Reyes Gonzalez. Information and communication technologies in food services and restaurants: a systematic review. - Emerald Group Publishing Limited, 2022. - 24с. [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/357911177_Information_and_communication_technologies_in_food_services_and_restaurants_a_systematic_review (Дата звернення: 09.06.2024).

ДОДАТКИ

Додаток А

MenuRecycler - адаптер компонента RecyclerView в активності Menu2

```
public class MenuRecycler extends
RecyclerView.Adapter<MenuRecycler.ViewHolder>{

private Vibrator vibrator;

   ArrayList<Category> cats;

   Context context;

   public MenuRecycler(ArrayList<Category> cats, Context context) {

       this.cats = cats;

       this.context = context;

   }

   @NonNull

   @Override

   public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {

       return new
ViewHolder(LayoutInflater.from(context).inflate(R.layout.category_item,
parent, false));

   }

   @Override

   public void onBindViewHolder(@NonNull ViewHolder holder, int position) {

       Category cat = cats.get(position);

       vibrator = (Vibrator)
context.getSystemService(Context.VIBRATOR_SERVICE);

       holder.cat1.setText(cat.cat_name);

       String imageUrl = cat.getUrl();

       Glide.with(context)

               .load(imageUrl)

               .into(holder.image);

       holder.Item.setOnClickListener(new View.OnClickListener() {

           @Override

           public void onClick(View v) {

               vibrator.vibrate(100);

               Intent intent = new Intent(v.getContext(), category1.class);
```

```
                intent.putExtra("cat_name", holder.cat1.getText());
                v.getContext().startActivity(intent);
            }
        });
    }
    @Override
    public int getItemCount() {
        return cats.size();
    }
    class ViewHolder extends RecyclerView.ViewHolder{
        TextView cat1;
        ImageView image;
        LinearLayout Item;
        public ViewHolder( View itemView) {
            super(itemView);
            cat1 = itemView.findViewById(R.id.cat1);
            image = itemView.findViewById(R.id.cat_image);
            Item = itemView.findViewById(R.id.menu_item);
        }
    }
}
```

Додаток Б

```
public class category1 extends AppCompatActivity {
    ActivityCategory1Binding binding;
    ArrayList<Meal> meals = new ArrayList<>();
    FirebaseFirestore db;
    MyRecycler myRecycler;
    ProgressDialog progressDialog;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        binding = ActivityCategory1Binding.inflate(getLayoutInflater());
        EdgeToEdge.enable(this);
        setContentView(binding.getRoot());
        progressDialog = new ProgressDialog(this);
        progressDialog.setCancelable(false);
        progressDialog.setMessage("Fetching Data...");
        progressDialog.show();
        binding.recyclerView.setLayoutManager(new LinearLayoutManager(this));
        binding.recyclerView.setAdapter(new MyRecycler(meals,this));
        db = FirebaseFirestore.getInstance();
        myRecycler = new MyRecycler(meals, category1.this);
        binding.recyclerView.setAdapter(myRecycler);
        Intent intent = getIntent();
        String category = intent.getStringExtra("cat_name");
        binding.CategoryName.setText(category);
        db.collection(category)
                .get()
                .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>()
{
                    @Override
                    public void onComplete(@NonNull Task<QuerySnapshot> task)
{
                        if (task.isSuccessful()) {
```

```
                            for (QueryDocumentSnapshot document :
task.getResult()) {
                                Meal meal = new
Meal(document.getString("Description"), document.getString("Price"),
document.getString("Image"),document.getId());
                                meals.add(meal);
                            }
                            if (progressDialog.isShowing()) {
                                progressDialog.dismiss();
                            }
                            myRecycler.notifyDataSetChanged();
                        } else {
                            Log.w(TAG, "Error getting documents.",
task.getException());
                        }
                    }
                });
        binding.btnBack.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(category1.this,Menu2.class));
            }
        });
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v,
insets) -> {
            Insets systemBars =
insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });


        }
}
```

Додаток В

MyRecycler - адаптер для компонента RecyclerView, який знаходиться в category1

```
public class MyRecycler extends RecyclerView.Adapter<MyRecycler.ViewHolder> {
   private Vibrator vibrator;
   ArrayList<Meal> meals;
   double HiddenSumPrice=0;
   Context context;
   FirebaseFirestore db = FirebaseFirestore.getInstance();
   public MyRecycler(ArrayList<Meal> meals, Context context) {
       this.meals = meals;
       this.context = context;
   }
   @NonNull
   @Override
   public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int
viewType) {
       return new
ViewHolder(LayoutInflater.from(context).inflate(R.layout.item_list, parent,
false));
   }
   @Override
   public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
       Meal meal = meals.get(position);
       vibrator = (Vibrator)
context.getSystemService(Context.VIBRATOR_SERVICE);
       holder.name.setText(meal.name);
       holder.Description.setText(meal.Description);
       holder.Price.setText(meal.Price);
       holder.count.setText(meal.count);
       String imageUrl = meal.getUrl();
       Glide.with(context)
               .load(imageUrl)
               .into(holder.Image);
       holder.plus_btn.setOnClickListener(new View.OnClickListener() {
```

```
            @Override
            public void onClick(View v) {
                vibrator.vibrate(100);
                int count1 = Integer.parseInt(meal.getCount())+1;
                String count2 = String.valueOf(count1);
                meal.setCount(count2);
                holder.count.setText(String.valueOf(meal.getCount()));
            }
        });
        holder.minus_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                vibrator.vibrate(100);
                int count1 = Integer.parseInt(meal.getCount());
                int count2 = Math.max(count1 - 1, 1);
                String newCount = String.valueOf(count2);
                meal.setCount(newCount);
                holder.count.setText(String.valueOf(newCount));
            }
        });
        holder.buy_btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                vibrator.vibrate(100);
                HiddenSumPrice =
Double.valueOf(meal.getPrice())*Integer.parseInt(meal.getCount());
                String HiddenSumPrice2=String.valueOf(HiddenSumPrice);
                int position = holder.getAdapterPosition();
                Meal meal = meals.get(position);
                Map<String, Object> mealData = new HashMap<>();
                mealData.put("name",meal.getName());
                mealData.put("description", meal.getDescription());
                mealData.put("price", meal.getPrice());
                mealData.put("count", meal.getCount());
```

```
                mealData.put("url", meal.getUrl());
                mealData.put("hiddenPrice", HiddenSumPrice2);
                MySingleton singleton = MySingleton.getInstance();
                String UserName = singleton.getVariable();
               db.collection(UserName).whereEqualTo("name", meal.getName())
                       .get().addOnCompleteListener(new
OnCompleteListener<QuerySnapshot>() {
                    @Override
                    public void onComplete(@NonNull Task<QuerySnapshot> task)
{
                        if (task.isSuccessful()){
                            if (task.getResult().isEmpty()){
                                db.collection(UserName)
                                        .add(mealData)
                                        .addOnSuccessListener(new
OnSuccessListener<DocumentReference>() {
                                            @Override
                                            public void
onSuccess(DocumentReference documentReference) {
                                                Log.d(TAG, "Meal added with
ID: " + documentReference.getId());
                                                Map<String, Object> updates =
new HashMap<>();

updates.put("id",documentReference.getId());

documentReference.update(updates)

.addOnSuccessListener(new OnSuccessListener<Void>() {
                                                                @Override
                                                                public void
onSuccess(Void unused) {
                                                                    Log.d(TAG,
"Document updated");

Toast.makeText(context,"Додано в кошик!", Toast.LENGTH_LONG).show();
                                                                }
                                                            })
```

```
.addOnFailureListener(new OnFailureListener() {
                                                                @Override
                                                                public void
onFailure(@NonNull Exception e) {
                                                                    Log.w(TAG,
"Error updating document", e);
                                                                }
                                                            });
                                                    }
                                                })
                                                .addOnFailureListener(new
OnFailureListener() {
                                                    @Override
                                                    public void onFailure(@NonNull
Exception e) {
                                                        Log.w(TAG, "Error adding
meal", e);
                                                    }
                                                });
                                    }else {
                                        Toast.makeText(context,"Страва вже додана до
корзини! Кількість збільшено!", Toast.LENGTH_LONG).show();
                                        db.collection(UserName)
                                                .whereEqualTo("name", meal.getName())
                                                .get()
                                                .addOnCompleteListener(new
OnCompleteListener<QuerySnapshot>() {
                                            @Override
                                            public void onComplete(@NonNull
Task<QuerySnapshot> task) {
                                                if (task.isSuccessful()){
                                                    DocumentSnapshot document =
task.getResult().getDocuments().get(0);
                                                    int count =
Integer.parseInt(document.getString("count"))+1;
                                                    double
hidPr=count*Double.valueOf(document.getString("price"));
```

```
db.collection(UserName).document(document.getId()).update("hiddenPrice",Strin
g.valueOf(hidPr));


db.collection(UserName).document(document.getId()).update("count",String.valu
eOf(count));


                                        }
                                    }
                                });
                            }
                        }
                    }
                });
            }
        });
    }
    @Override
    public int getItemCount() {
        return meals.size();
    }
    static class ViewHolder extends RecyclerView.ViewHolder{
        TextView Description,  Price, count, name;
        AppCompatButton plus_btn, minus_btn;
        ImageButton buy_btn;
        ImageView Image;
        public ViewHolder( View itemView) {
            super(itemView);
            Description = itemView.findViewById(R.id.text_description);
            Image = itemView.findViewById(R.id.image_meal);
            Price = itemView.findViewById(R.id.text_price);
            count = itemView.findViewById(R.id.count_text);
            plus_btn = itemView.findViewById(R.id.plus_btn);
            minus_btn = itemView.findViewById(R.id.minus_btn);
            buy_btn = itemView.findViewById(R.id.buy_btn);
```

```
            name = itemView.findViewById(R.id.meal_name);
        }
    }
}
```