Київський національний університет
імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра теоретичної кібернетики

Кваліфікаційна робота
на здобуття ступеню бакалавра
за спеціальністю 122 Комп’ютерні науки
на тему:

Розробка фулстек застосунку для замовлення їжі на платформі Android

Виконав студент 4-го курсу ____________
Дмитро ВАЛЬДМАН (підпис)

Науковий керівник:
Доцент ____________
Андрій СТАВРОВСЬКИЙ (підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент ____________
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри теоретичної кібернетики
«____»________________2023 р., протокол №____

Завідувач кафедри
Юрій КРАК ___________
(підпис)

Київ - 2023

Реферат

Обсяг роботи 49 сторінок, 9 ілюстрацій, 19 джерел посилань, 1 додаток.

ДОДАТОК, БАЗА ДАНИХ, СЕРВЕР, АРХІТЕКТУРА, SPRING, HIBERNATE, АНАЛІЗ РИНКУ, RETROFIT, JAVA, ANDROID STUDIO, IDEA, ГРАФІЧНИЙ ІНТЕРФЕЙС.

Об'єктом роботи є розробка додатка на Android для замовлення їжі.

Метою роботи є створення додатка для замовлення їжі з усіма необхідними функціями та зручним користувацьким інтерфейсом. Також метою роботи є показати процес розробки цього додатка та ознайомитись з альтернативними технологіями.

Методи розробки: інкрементальний метод розробки. Інструментами реалізації є мова програмування Java, фреймворки Spring та Hibernate, бібліотека Retrofit.

Результат роботи: створено додаток на базі Android. Додаток працює коректно, з приємним користувацьким інтерфейсом та більшістю потрібних функцій. Завдяки аналізу користувацьких потреб, дизайн додатка був створений так, щоб користувачу було максимально зручно при використанні.

Програмний продукт можна застосовувати в сфері замовлення їжі. Також, завдяки відносно базовій архітектурі, може бути змінений під застосування в інших сферах, наприклад в сфері моди та шопінгу, господарській сфері, тощо.

Зміст

Вступ **3**

1 Планування розробки додатка **7**

1.1 **Дослідження методів розробки** 7

1.2 **Вибір архітектури сервера та Android додатка** 14

1.2.1 Які є архітектури сервера? 15

1.2.2 Які є архітектури додатка? 19

1.3 **Дизайн графічного інтерфейсу додатка на Android** 23

2 Дослідження технологій, які використовуються при розробці додатка **26**

2.1 **Середовище розробки** 26

2.2 **Java або Kotlin** 29

2.3 **Вибір серверних технологій** 31

2.4 **Вибір технологій, що поєднують клієнт з сервером** 34

3 Процес розробки **37**

3.1 **Аналіз користувацьких потреб для зручного користування додатком** 37

3.2 **Розробка графічного інтерфейсу та перших функцій додатка** 38

3.3 **Розробка сервера під потреби Android-клієнта** 39

3.4 **Завершення процесу розробки** 42

3.5 **Подальший розвиток додатка** 45

Висновки **46**

Перелік джерел посилання **47**

Додаток А **49**

Вступ

Оцінка сучасного стану додатків для замовленню їжі

Бурхливий розвиток технологій у сучасному світі дозволив людині нашого часу те, про що наші найдавніші предки не змогли би й навіть близько мріяти: замість небезпечного і виснажливого полювання зі списом на мамонта чи іншого дикого звіра – можливість здобути їжу всього в один дотик до екрану мобільного телефона. Додатки для замовлення їжі вже давно перетворилися на індустрію, і з кожним днем конкуренція у цій індустрії зростає дедалі більше.

Як і в більшості економічних моделей сучасного світу, додатки для замовлення їжі мають своїх «гігантів» на ринку, які кожен день намагаються знайти спосіб вийти вперед у цій нескінченній гонці за право вважатися єдиним власником всієї монополії, таким собі «монархом», здатним власноруч диктувати правила бурхливому та непередбачуваному ринку. Конкуренція зростає з кожним днем, адже бурхливий розвиток технологій дозволяє різним учасникам перехоплювати ініціативу, та переганяти цілу низку конкурентів. Ще вчора маленька компанія, яку зневажливо б назвали «малим бізнесом», може завдяки новій технології, яка буде вирізняти її з-поміж інших, зайняти нові для себе позиції, вийти на міжнародний рівень та здобути статус компанії «на слуху». Але звісно ж, лідери просто так не будуть здавати своїх позицій, які вони займали довгими роками, і вони будуть робити все, щоб зберегти свій статус «одних з найкращих». До таких «компаній-акул» можна віднести такі бренди як Uber Eats, DoorDash, Glovo, Just Eat, FoodPanda, Deliveroo та декілька інших.

Одним з найбільших показників якості компанії по доставці їжі є її додаток. Від того, наскільки він буде якісно зроблений залежить те, скільки потенційних користувачів може бути у компанії. Тож сучасний

стан додатків для замовлення їжі такий, що всі учасники «перегонів» за перше місце на ринку змагаються між собою в унікальності та зручності їх додатків, наразі ця боротьба проходить дуже активно та на досить високому рівні, і з кожним роком стає дедалі більшою та конкурентнішою. Саме тому наша мета – розробити такий додаток, який буде відповідати всім вимогам якості, та буде «зразком» того, яким має бути сучасний мобільний додаток.

Актуальність роботи та підстави для її виконання

Як було сказано трошки вище – дослідження мобільних додатків для замовлення їжі є дуже актуальними, особливо в контексті швидко зростаючого ринку онлайн-замовлень їжі та змін у споживацьких поведінках.

Є ціла низка причин, чому це дослідження є актуальним. Нижче перераховані основні серед них:

1. **Зростання популярності**

Мобільні додатки для замовлення їжі набувають все більшої популярності серед споживачів. Дослідження можуть допомогти визначити причини цього зростання популярності, зрозуміти споживацькі уподобання та очікування, а також виявити можливості для поліпшення додатків та їх функціоналу.

2. **Вплив на ресторанний бізнес**

Мобільні додатки для замовлення їжі мають значний вплив на ресторанний бізнес. Дослідження можуть допомогти рестораторам зрозуміти, як ці додатки впливають на їхнє підприємство, як покращити процес замовлення та доставки, а також як розширити свою аудиторію через ці платформи.

3. **Технологічний розвиток**

Замовлення їжі через мобільні додатки залежить від широкого спектру технологій, таких як геолокація, онлайн-оплата, відстеження замовлення та доставки. Дослідження можуть допомогти в розумінні технологічних тенденцій, інновацій та їх впливу на досвід користувача та операційні процеси.

4. **Виклики та можливості**

Дослідження можуть виявити виклики, на які натрапляють користувачі та постачальники послуг у сфері замовлення їжі через мобільні додатки. Також вони можуть ідентифікувати можливості для покращення додатків, вирішення проблем та створення нових функціональних можливостей.

5. **Вплив на споживацьку поведінку**

Мобільні додатки для замовлення їжі впливають на споживацьку поведінку, зокрема на звички щодо харчування, вибір ресторанів та ставлення до доставки їжі. Дослідження можуть вивчати ці впливи та їх наслідки для здоров'я, соціальної взаємодії та інших аспектів життя користувачів.

Отже, дослідження мобільних додатків для замовлення їжі допомагають зрозуміти їхню роль у сучасному суспільстві, виявити переваги та недоліки, визначити напрями подальшого розвитку та покращення. Це дозволяє компаніям у цій галузі вдосконалювати свої продукти, а також забезпечує користувачам кращий досвід замовлення їжі через мобільні додатки.

Мета й завдання роботи

Мета кваліфікаційної роботи полягає у створенні мобільного додатка для замовлення їжі, який буде реалізовувати більшість функцій, потрібних для сучасного ринку. Розроблюваний засіб має працювати на платформі Android та використовувати клієнт-серверну архітектуру.

Для досягнення цієї мети поставлені наступні завдання:

1. Ознайомитись з Retrofit, Spring Boot;
2. Дослідити процес розробки Android додатків з клієнт-серверною архітектури;
3. Ознайомитись та підібрати найзручніше середовище розробки;
4. Обрати мову написання коду;
5. Розробити зручний користувацький інтерфейс;
6. Розробити логічну складову додатка.

Об'єкт, методи й засоби розроблення

Використовувались такі інструментальні засоби, бібліотеки та технології: Android Studio, Intellij Idea, REST API, Spring Boot, Hibernate, MySQL, java 11, Android sdk 26+, Retrofit.

Можливі сфери застосування

Розроблене програмне забезпечення може використовуватись в сфері замовлення їжі. Додаток є досить функціональним, тож з його допомогою можна реалізувати онлайн ресторан, в якому користувачі зможуть замовляти улюблені страви. Завдяки відносно базовій архітектурі додатка «клієнт-сервер», його може бути змінено та використано в багатьох інших сферах, наприклад створенні онлайн магазину з побутовими товарами, онлайн аптеки, гіпермаркету, тощо.

1 Планування розробки додатка

Створення будь-чого починається з попереднього планування. Неможливо створити щось якісне, попередньо не розпланувавши це. Коли мова йде про розробку мобільного додатка – без планування неможливо створити щось навіть віддалено схоже на функціональний продукт. Так чому планування так важливо при розробці мобільного додатка?

Планування розробки додатка є важливим етапом в процесі створення програмного забезпечення, оскільки визначає ключові аспекти проєкту та надає основу для подальшої роботи. У цій підтемі будуть розглянуті наступні питання, пов'язані з плануванням розробки додатка:

- **Дослідження методів розробки;**
- **Вибір архітектури сервера та Android додатка;**
- **Дизайн графічного інтерфейсу додатка на Android**.

1.1 Дослідження методів розробки

Як було сказано раніше – розробити мобільний додаток можливо тільки за допомогою ретельного планування. Застосунок на мобільні телефони – це величезний проєкт, над яким, зазвичай, працює чимала команда людей, які вдосконалюють кожний аспект майбутнього продукту. Багато людей, коли бачать якісний застосунок, улюблену мобільну гру чи інше - часто задаються цікавим: яким був перший крок? З чого почалась розробка? Як виглядала перша строка коду?

Тож з чого почати? Перед тим як розпочати розробку Android-застосунку, важливо ретельно розглянути **методологію розробки ПЗ**, яка найкраще підходить для вашого проєкту [1].

Вибір методу розробки ПЗ для Android-застосунку може вирішити багато проблем, з якими можуть зіткнутися розробники. Коректно обрана методологія допоможе покращити продуктивність, зменшити ризики та забезпечити якісний результат. Кожен проєкт має унікальні вимоги, такі як обсяг роботи, терміни, бюджет, замовників очікування. Вибір методу розробки ПЗ дозволяє зорієнтуватися на конкретні потреби проєкту і забезпечити відповідність вимогам замовника. Також важливо мати саме такий метод, який зможе пристосуватися до нових обставин, адже ринок технологій швидко змінюється, вимоги замовників можуть змінюватися, все в сучасному світі має тенденцію змінюватися.

В комплексному додатка не менш важливо те, щоб обраний метод мав низку перевірок якості продукту. Деякі методи мають вбудовані етапи тестування та перевірки якості, що допомагає забезпечити належну якість програмного забезпечення перед його впровадженням.

Вибір методу розробки також впливає на комунікацію та співпрацю у команді. Деякі методи, наприклад Agile, акцентують на постійній взаємодії замовника, ефективному спілкуванні та колективній власності проєкту. Це надзвичайно важливо при розробці мобільного додатка.

Які є методи розробки?

Каскадний метод (Waterfall). Цей метод є більш старішим методом, та вважається «класичним». Метод передбачає в собі послідовне виконання фаз розробки в строго визначеному порядку, схожому на покрокове рухання вниз по ступенях водоспаду. Основні фази каскадного методу включають:

1. **Вимоги.** Етап, у якому встановлюються та документуються вимоги до програмного продукту. Здійснюється аналіз потреб користувачів

та визначаються функціональні та нефункціональні вимоги до системи.

2. **Проєктування.** Етап, у якому відбувається проєктування архітектури системи, складних компонентів та інтерфейсів. Розробляється детальний план робіт та документація, що описує структуру системи.
3. **Реалізація.** Етап, у якому розроблюються та програмуються компоненти системи згідно з вимогами та проєктом. Код написаний розробниками перевіряється на відповідність специфікаціям.
4. **Тестування.** У цій фазі проводяться різні види тестувань, щоб переконатися, що система працює належним чином та задовольняє вимоги. Тестування може включати функціональне тестування, інтеграційне тестування, системне тестування, регресійне тестування та інші види тестування.
5. **Впровадження.** Етап, у якому користувачі отримують доступ до продукту.
6. **Підтримка.**

Waterfall все ще можна назвати популярним методом, але з кожним роком він стає все дедалі менш актуальним. Серед головних недоліків методу можна виділити те, що зазвичай при розробці цим методом у команди з'являється дуже багато документів, які постійно треба оновлювати (актуалізувати), що часто дуже тормозить процес розробки. Методолія взагалі не є гнучкою, що є величезним недоліком, адже замовники часто можуть одразу і не знати, що саме вони хочуть, а також мають тенденцію «передумувати». Але головним мінусом методологоії є те, що замовника майже повністю «ізолюють» від проєкта, і він не має змоги вносити певні коригування до проєкту. В цілому метод підходить до більш конвеєрних завдань, і є не дуже доречним в нашому проєкті.

Ітеративний та інкрементальний методи. На відміну від попереднього методу, ітеративний та інкрементальний методи розробки ПЗ є гнучкими підходами, які ставлять акцент на поступову еволюцію та постійний зворотний зв'язок замовника. Ці методи дозволяють розробникам швидко отримувати результати, а користувачам - бачити певну функціональність системи на ранніх етапах розробки. Як було сказано – ці два методи є гнучкими, тож вони не мають певних «суворих» етапів, зате в них можна виділити чіткі риси та певні особливості.

Риси ітеративного методу:

- Розробка відбувається через серію ітерацій або циклів, кожен з яких представляє собою повний цикл розробки від вимог до впровадження;
- На кожній ітерації розробка включає фази вимог, проєктування, реалізації, тестування та впровадження;
- Після кожної ітерації замовник отримує можливість переглянути та оцінити поточний стан системи, а також внести зміни та корективи до подальших ітерацій;
- Кожна нова ітерація додає нові функціональні можливості до системи, поступово покращуючи її.

Риси інкрементального методу:

- Розробка відбувається шляхом послідовного додавання функціональності до системи в кілька інкрементів або етапів;
- Кожен інкремент - це повний цикл розробки, який включає фази вимог, проєктування, реалізації, тестування та впровадження;
- Кожен інкремент пропонує нову функціональність, яка додається до попереднього вже існуючого функціоналу;
- Кожен інкремент може бути випущений окремо та функціонувати самостійно.

Основною перевагою ітеративного та інкрементального методів є можливість швидко реагувати на зміни вимог, впроваджувати функціональність поступово та отримувати зворотний зв'язок від замовника на ранніх етапах розробки. Ці методи особливо корисні в ситуаціях, коли вимоги не є повністю зрозумілими або можуть змінюватися протягом проєкту, що характерно в нашому випадку, адже процес розробки застосунку є гнучким. При розробці можуть виникати нові ідеї «як можна покращити/оптимізувати застосунок», тому необхідна модель, яка задовольнить наші вимоги по гнучкості. Саме тому була обрана ця методологія.

Agile-методи. Agile є сім'єю гнучких методологій розробки, таких як Scrum, Kanban, XP (Extreme Programming) та інші. Вони спрямовані на ітеративну та інкрементальну розробку, з високою залученістю замовника, ефективним комунікаціями в команді та постійною здатністю адаптуватися до змін.

На перший погляд Agile-методи можуть здатися тим самим, що і ітеративний та інкрементальний методи. Дійсно, ці дві методології досить схожі між собою, десь вони навіть включають одне одну в свої риси, проте вони мають між собою деякі відмінності. **Ключовими відмінностями є:**

- **Спосіб залучення замовника у проєкт.** Хоч ітеративний та інкрементальний методи мають досить тісну комунікацію з замовником, все ж Agile-методи надають більш активну взаємодію замовника під час всього процесу розробки. Замовник бере активну участь у плануванні, пріоритезації ітерацій та надає зворотний зв'язок на кожному етапі розробки.

- **Керування проєктом.** Інкрементальний метод може мати меншу кількість формалізованого керування проєктом. Кожен інкремент може розглядатися як окремий проєкт зі своїми власними етапами та завершенням. В свою чергу, Agile використовує більш формальні методи керування проєктом, такі як Scrum або Kanban. Вони включають планування спринтів, стендапи (щоденні зустрічі команди), оцінки завершеності робіт та інші методи для забезпечення ефективного керування проєктом.

Якщо надавати загальну оцінку Agile-методології, то варто відзначити, що це досить сучасний та зручний спосіб планування. Більшість компаній світу використовують саме цей метод. Проте, в нашому проєкті він не був використаний, адже ця методологія спрямована більше на роботу в команді, де кожен учасник виконує свою роботу. Варто відзначити, що потреба проводити щоденні зустрічі самому з собою більше схоже на помилку у виборі методології.:)

Spiral Model. Цей метод розробки програмного забезпечення, поєднує ітераційний підхід з елементами прототипування та керування ризиками. Цей підхід розглядається як розширення водоспадної моделі, що дозволяє більш гнучко виконувати проєкти, особливо ті, де ризики та невизначеність важливі.

Основна ідея Spiral Model полягає в тому, що розробка програмного забезпечення відбувається через повторювання циклів, де кожен цикл розширює функціональність системи та враховує результати попередніх циклів та ризики. Це дозволяє гнучко виправляти помилки, впроваджувати зміни та забезпечувати високу якість продукту. Серед недоліків цієї методології варто відзначити **складність** керування – цей метод потребує

досвідчену команду, в якій присутній високий рівень координації, **високі витрати** часу та ресурсів, та **відстуність такої гнучкості**, як, наприклад, в Agile-моделях.

Якщо підбивати підсумки, то можна сказати, що Spiral Model є хорошою методологією, проте вона ефективна тільки за наявності досвідченої команди розробників.

DevOps. це методологія розробки програмного забезпечення, яка поєднує розробку (Development) та експлуатацію (Operations). Вона ставить на меті покращити співпрацю та комунікацію між розробниками програмного забезпечення та операторами систем, щоб забезпечити швидку і надійну поставку програмного продукту.

DevOps включає в себе ряд практик та інструментів, які сприяють автоматизації процесів, стандартизації та збільшенню ефективності розробки та експлуатації програмного забезпечення. DevOps підкреслює важливість **співпраці та комунікації між командами розробки та експлуатації** - це включає обмін знаннями, спільні цілі та відповідальність за кінцевий результат. Одним з головних принципів DevOps є використання автоматизації для прискорення процесів розробки та експлуатації. Це включає автоматизоване тестування, встановлення та моніторинг, що дозволяє швидко виявляти та виправляти проблеми.

Переваги DevOps включають в себе **покращення швидкості поставки, зниження ризиків, забезпечення стабільності та надійності** програмного забезпечення**, зменшення часу відновлення після помилок** та **покращення здатності до реагування на зміни.**

Однак, реалізація DevOps може стикатися з деякими викликами, такими як складність впровадження нових процесів та зміна культури

організації. Впровадження DevOps вимагає зміни мислення, співпраці та інфраструктури компанії.

Загалом, DevOps є потужним методом розробки програмного забезпечення, який допомагає створювати продукт швидше, надійніше та з меншими ризиками, але він є більш вузьким методом розробки, перейти на який може бути досить складно для команди розробників.

1.2 **Вибір архітектури сервера та Android додатка**

Як тільки була обрана методологія розробки, команда розробників переходить до більш детального планування. Потрібно створити фундамент майбутнього продукту – вибрати його архітектуру. В нашому випадку треба зробити одразу два вибори: визначити, яка буде архітектура сервера, та яка буде архітектура додатка.

Обрати правильну архітектуру сервера надзвичайно важливо: це дозволить розробнику досягти високої продуктивності системи, що надзвичайно важливо, адже це дозволить обробляти в майбутньому велику кількість запитів та великі обсяги даних. Також правильно обрана архітектура дає можливість легко впроваджувати зміни та розширювати функціональність системи. Гнучка архітектура дозволяє легко інтегрувати нові компоненти, змінювати логіку додатків та впроваджувати покращення без необхідності переробки всієї системи. Ну і звісно, не можна не виділити те, що правильно підібрана архітектура впливає на загальну вартість проєкту: витрати на обладнання, підтримку та експлуатацію та впровадження нових функцій. Це може бути дуже важливо для проєктів без великого фінансування, де розробники мають обмежені можливості по ресурсам.

1.2.1 Які є архітектури сервера?

SOAP (Simple Object Access Protocol). SOAP є **протоколом обміну** повідомленнями, який використовує XML для форматування повідомлень та протоколу HTTP, SMTP або інших для передачі повідомлень. SOAP надає більш формалізований підхід до обміну даними та підтримує широкий спектр функцій, включаючи безпеку та транзакції.

У SOAP повідомлення мають строгу структуру, визначену за допомогою WSDL (Web Services Description Language). WSDL описує доступні методи, параметри та типи даних, що можуть бути використані в обміні повідомленнями. Крім того, SOAP підтримує різні протоколи для передачі повідомлень, такі як HTTP, SMTP і т. д.

SOAP вважається більш застарілим ніж його аналоги. У свій час, на зміну SOAP-у прийшов REST API, який одразу вразив усіх своєю простотою, та майже повністю замінив SOAP.

GraphQL. GraphQL є **запитовою мовою** та архітектурним підходом розробленим компанією Facebook у 2015-му році, що дозволяє клієнту точно визначати дані, які він потребує від сервера. Замість того, щоб отримувати всі дані з одного ресурсу, як у REST, GraphQL дозволяє клієнту визначати структуру та поля даних, які йому потрібні. Це дозволяє зменшити кількість запитів та отримати більш гнучкий підхід до обміну даними. [3]

Запити GraphQL формуються клієнтом, в яких вказуються конкретні поля та взаємозв'язки, які потрібні клієнту. Це дозволяє уникнути проблеми "over-fetching" (отримання зайвих даних) та "under-fetching" (недостатнє отримання даних), які часто виникають у REST API.

GraphQL представляє схему, яка описує всі доступні дані та їх взаємозв'язки. Це дозволяє клієнтам дізнатись про можливості сервера та виконувати складні запити, використовуючи глибокі вкладеності та фрагменти.

Також варто відзначити, що GraphQL використовує один єдиний ендпоінт для всіх запитів, що дозволяє зменшити кількість запитів до сервера. Клієнт може запитувати різні дані в одному запиті, отримуючи лише необхідну інформацію.

За допомогою GraphQL розробники мають більший контроль над даними, які отримують клієнти, та можуть побудувати більш ефективні та гнучкі API.

gRPC (Google Remote Procedure Call). Це високопродуктивний **фреймворк** для віддалених викликів процедур, розроблений компанією Google. Він базується на протоколі HTTP/2 для передачі повідомлень та використовує Protocol Buffers для серіалізації даних. gRPC надає зручні інструменти для визначення інтерфейсів служби та автоматичної генерації коду для різних платформ.

Фреймворк надає простий та зрозумілий спосіб описувати інтерфейси служби за допомогою Protocol Buffers IDL (Interface Definition Language). За допомогою цього опису генерується код для клієнта та сервера, що спрощує розробку та інтеграцію.

Також gRPC підтримує три типи викликів: **unary** (одиночний виклик), **server streaming** (потоковий виклик з сервера) та **client streaming** (потоковий виклик з клієнта). Це дає можливість обробляти різні сценарії викликів, такі як одиночні запити, відправка потоку даних з сервера або клієнта та багато інших. [2]

gRPC знаходить широке використання в мікросервісній архітектурі, де вимагається ефективна взаємодія між різними сервісами. Він дозволяє розробникам створювати потужні та масштабовані системи, забезпечуючи ефективну комунікацію між компонентами.

REST (Representational State Transfer) API. REST є **архітектурним стилем**, що використовується для побудови веб-сервісів. Він має кілька переваг, які сприяють популярності та широкому використанню REST у веб-розробці [4]. Ось деякі з цих переваг:

- **Простота**. REST використовує стандартні HTTP-методи, такі як GET, POST, PUT і DELETE, для взаємодії з ресурсами. Це робить його простим для розуміння та використання, як для розробників, так і для клієнтів.
- **Широка підтримка**. REST є одним з найпоширеніших стандартів веб-сервісів, і більшість платформ та мов програмування мають підтримку для роботи з REST API.
- **Розділення клієнт-сервер**. REST використовує принцип розділення клієнта та сервера, де клієнт і сервер мають свої відповідні обов'язки.
- **Стандартизація**. REST використовує стандартні протоколи, такі як HTTP, що дозволяє легко взаємодіяти з існуючими інфраструктурами та інструментами, такими як проксі-сервери, кешування, балансування навантаження тощо.
- **Кешування**. REST підтримує механізми кешування на рівні протоколу HTTP.
- **Скалярність**. REST дозволяє гнучко масштабувати систему шляхом розділення функцій на різні ресурси та використання горизонтального масштабування. Кожен ресурс може бути оброблений окремо, що полегшує розподілену обробку завдань.

REST - простіший, легший у використанні підхід, який використовує HTTP-методи та формати даних, такі як JSON. Навідміну від SOAP він використовує простіші принципи і структуру. Для нашого додатка ми обрали саме цей архітектурний стиль, адже ми маємо невеликий сервер у нашому проєкті, і він набагато легше масштабується навідміну від аналогів, що дозволяє зекономити велику кількість роботи.

Порівняння основних перерахованих архітектур за певними критеріями можна побачити на рис. 1.1 з джерела [5].

API ARCHITECTURAL STYLES

	RPC	SOAP	REST	GraphQL
Organized in terms of	local procedure calling	enveloped message structure	compliance with six architectural constraints	schema & type system
Format	JSON, XML, Protobuf, Thrift, FlatBuffers	XML only	XML, JSON, HTML, plain text,	JSON
Learning curve	Easy	Difficult	Easy	Medium
Community	Large	Small	Large	Growing
Use cases	Command and action-oriented APIs; internal high performance communication in massive micro-services systems	Payment gateways, identity management CRM solutions financial and telecommunication services, legacy system support	Public APIs simple resource-driven apps	Mobile APIs, complex systems, micro-services

altexsoft
software r&d engineering

Рис 1.1 – Порівняння архітектур сервера

1.2.2 Які є архітектури додатка?

Model-View-Controller (MVC). MVC є однією з найпоширеніших архітектур для розробки програмного забезпечення, включаючи мобільні додатки на платформі Android. Вона розділяє програму на три основні компоненти: **модель** (Model), **представлення** (View) та **контролер** (Controller). Кожен з цих компонентів має свої відповідальності і функції. Основна ідея полягає в розділенні логіки додатка, представлення даних і керування взаємодією з користувачем.

Модель (Model): Модель відповідає за керування даними, бізнес-логікою та станом додатка. Вона може включати структури даних, бази даних, мережеві запити, а також операції збереження, отримання, оновлення та видалення даних. Модель не залежить від інших компонентів і може сповіщати представлення про зміни, якщо вони виникають.

Представлення (View): Представлення відповідає за відображення графічного інтерфейсу користувача (GUI) і відображення даних з моделі. Воно може бути відповідальним за відображення тексту, зображень, кнопок, списків тощо. Представлення також може взаємодіяти з контролером, повідомляючи про дії користувача або відображаючи дані з моделі.

Контролер (Controller): Контролер обробляє взаємодію користувача з додатком і керує потоком даних між моделлю та представленням. Він реагує на події користувача, такі як натискання кнопок, жести тощо, і виконує відповідні дії. Контролер може оновлювати модель, якщо необхідно, і оновлює представлення залежно від стану моделі.

Важливим принципом MVC є розділення відповідальностей між цими компонентами. Модель не залежить від представлення або контролера, тому вона може бути повторно використана та тестована

незалежно від інших компонентів. Представлення не залежить від моделі або контролера, що дозволяє змінювати спосіб відображення даних без впливу на решту системи. Контролер відповідає за керування взаємодією із користувачем, але не повинен містити значного обсягу логіки.

MVC є гнучкою архітектурою, яка полегшує розробку, тестування та підтримку додатків. Вона дозволяє розробникам розмежовувати різні аспекти додатка та спрощує зміни і розширення.

Model-View-Presenter (MVP). Ця архітектура дещо схожа на попередню, проте вона має деякі відмінності. Головною відмінністю є «презентер» - замість контролеру, який обробляє взаємодію користувача з додатком, презентер виступає посередником між моделлю та представленням. Він отримує взаємодію користувача з представлення і виконує відповідні дії на основі цієї взаємодії. Далі він отримує дані з моделі, оновлює їх та передає до представлення для відображення. Він також може виконувати додаткові завдання, такі як обробка введених даних, валідація, навігація тощо. Це забезпечує більшу гнучкість і тестованість коду, оскільки можна легко замінити або модифікувати представлення та модель без змін в презентері.

MVP дозволяє ефективно організовувати код і полегшує розробку, тестування та підтримку додатків. Вона розділяє логіку бізнес-процесів від графічного інтерфейсу, що поліпшує супровідність і розширюваність додатка.

Model-View-View Model (MVVM). Це також досить популярна архітектура для розробки програмного забезпечення. Ця архітектура спрямована на покращення керування станом та забезпечення простоти тестування та розширюваності.

Ця архітектура має два спільних компонента з попередніми описаними архітектурами: Model та View, та відрізнається розподілом відповідальностей та способом взаємодії між компонентами. MVVM акцентує увагу на розділенні логіки представлення (View) та бізнес-логіки (Model) за допомогою проміжного компонента, відомого як в'ю-модель (ViewModel). Це дозволяє виділити із представлення всю логіку, пов'язану з відображенням та взаємодією з користувачем, тоді як бізнес-логіка знаходиться в моделі. У інших архітектурах, таких як MVC та MVP, логіка представлення і бізнес-логіка частково змішані.

Також MVVM використовує двосторонню прив'язку для автоматичного оновлення даних між представленням та в'ю-моделлю. Це дозволяє автоматично синхронізувати дані без необхідності вручну оновлювати інтерфейс. Інші архітектури не мають такого вбудованого механізму двосторонньої прив'язки даних. [6]

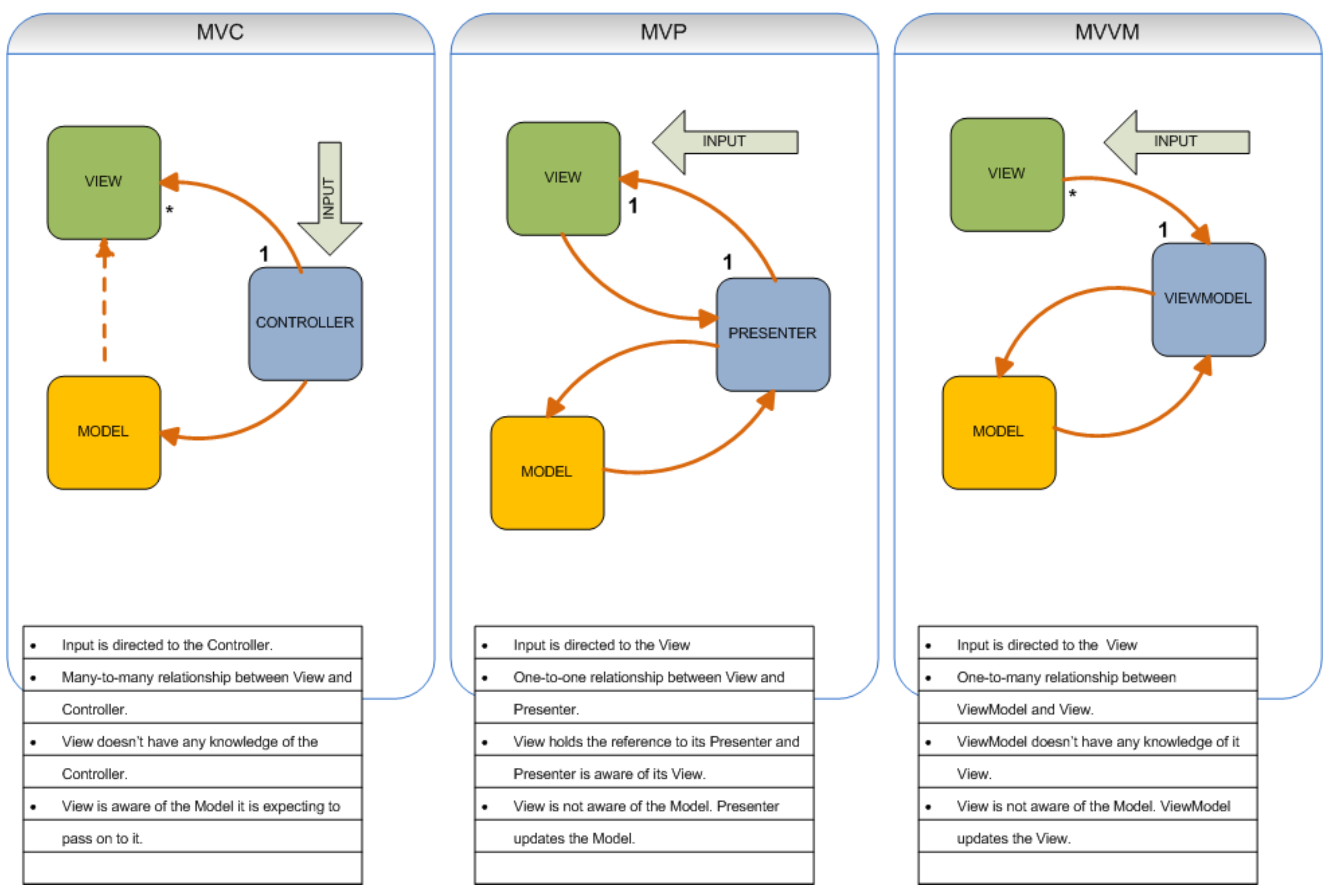

Рис. 1.2 – Порівняння архітектур додатка

У проєкті була обрана саме ця архітектура, і, напевно, головною причиною цьому є **простота тестування** в MvvM, а також в цій архітектурі бізнес-логіка добре відділена від користувацького інтерфейсу, що дає змогу краще розуміти структуру проетку. В'ю-модель може бути тестована без необхідності запуску інтерфейсу користувача. Крім того, завдяки двосторонній прив'язці даних, можна легко тестувати взаємодію між в'ю-моделлю та представленням (view).

Порівняння архітектур MVP, MVC та MVVM можна побачити на рисунку 1.2 з джерела [7].

Clean Architecture. Це архітектурний підхід до розробки програмного забезпечення, який пропонує принципи та шаблони для організації коду таким чином, щоб він був масштабованим, гнучким та легко тестованим. Цей підхід був розроблений Робертом Мартіном (також відомий як Uncle Bob) і спрямований на розділення програми на незалежні від реалізації шари з підвищеною залежністю від абстракцій.

Головним принципом Clean Architecture є розділення шарів та направлення залежностей. Вона дозволяє змінювати або модифікувати один шар, не впливаючи на інші. Це полегшує розширення, підтримку та тестування коду, оскільки можна змінювати окремі компоненти безпосередньо без впливу на решту системи. Clean Architecture допомагає підтримувати код організованим, чистим та легко зрозумілим. Вона сприяє масштабованості додатків та полегшує роботу команди розробників, оскільки кожен шар має чітко визначені відповідальності та границі.

Ця архітектура дещо поступається аналогам через деякі недоліки. Перший недолік це **складність** архітектури. Вона вимагає визначення багатьох інтерфейсів та абстракцій, а також розділення логіки на різні шари, що може призвести до збільшення загального обсягу коду та втрати

простоти. Також суттєвим недоліком є **перевитрата ресурсів**. Ізоляція шарів та збільшення кількості абстракцій може призвести до надмірного використання ресурсів, таких як пам'ять та процесорний час. Запити на переходи між різними шарами, виконання логіки переходу та перетворення даних можуть займати більше часу і ресурсів, порівняно з більш простими архітектурами. В цілому Clean Architecture не підходить для менш досвідчених команд розробників, бо вимоги щодо розділення логіки та визначення інтерфейсів можуть вимагати додаткової підготовки та навчання для команди.

1.3 Дизайн графічного інтерфейсу додатка на Android

Після того, як було створено фундамент майбутнього додатка, а саме обрали методологію розробки та потрібні нам архітектури – час переходити до більш креативної частини проєкту. Створення графічного дизайну додатка є не менш важливою частиною, ніж написання коду, та створення логічної частини проєкту. Саме завдяки вдалим рішенням при створенні графічного інтерфейсу деякі додатки стають популярнішими за інші. При плануванні вигляду майбутного проєкту важливо пам’ятати: користувач повинен почувати себе максимально комфортно, він не має замислюватися про всі процеси, які передували створенню застосунку, він має тільки отримувати задоволення від вдалості графічно-дизайнерських рішень.

Саме по собі слово «Інтерфейс» розкриває саму сутність важливості цієї проблеми. Інтерфейс – це **точка дотику між користувачем та логічною складовою продукта**, і вона має бути максимально лагідною та приємною. Розробка графічного інтерфейсу додатка на Android вимагає максимальної уваги до деталей, враховування контексту та призначення

додатка, а також забезпечення позитивного користувацького досвіду на різних пристроях.

Існує перелік базових принципів, без яких створення вдалого користувацького інтерфейсу неможливе. Ці принципи встановлювалися компанією Google, та стосуються додатків на платформі Android [8]. Принципи наступні:

- **Material Design.** Material Design є дизайн-мовою, рекомендованою Google для розробки додатків на Android. Вона надає рекомендації щодо використання кольорів, форм, тіней, анімацій та інших елементів інтерфейсу. У Material Design використовується концепція матеріалу, яка передбачає, що об'єкти на екрані повинні виглядати як фізичні шари, з дотриманням реальних фізичних властивостей. Об'єкти мають мати глибину, властивості тіні, реагувати на рухи та взаємодію користувача. Дотримання Material Design допоможе забезпечити консистентність та зручність використання додатка.
- **Responsive Layout.** Розробка інтерфейсу має бути адаптивною, щоб додаток добре виглядав і працював на різних розмірах екранів.
- **Навігація.** Забезпечення простої та зрозумілої навігації в додатка є важливим аспектом. Використання панелей навігації, бічних меню, закладок або схеми навігації зі стеком фрагментів сприяє зручності використання.
- **Елементи** керування**.** Необхідно використовувати стандартні елементи керування, такі як кнопки, текстові поля, списки, перемикачі, прапорці тощо. Додатково, можна використовувати кастомні елементи керування.
- **Кольори та типографіка.** Вибір кольорів і типографіки важливий для створення привабливого та зручного інтерфейсу. Рекомендується використовувати консистентні кольори та шрифти, які відповідають

бренду додатка, забезпечують хорошу читабельність та контрастність.

- **Анімація.** Додавання анімаційних ефектів може поліпшити взаємодію з додатком та зробити його більш привабливим. Використання переходів між екранами, анімацій прокрутки, плавного затемнення елементів та інших ефектів може покращити враження користувача.

Правильний графічний дизайн у мобільних додатках має велику важливість і впливає на загальний успіх додатка. Графічний інтерфейс є першим з чим зустрічається користувач, і він вирішує, чи залишиться він взаємодіяти з додатком або відмовиться від нього.

2 Дослідження технологій, які використовуються при розробці додатка

2.1 **Середовище розробки**

Коли проєкт вже повністю розпланований, залишається найцікавіша робота – написання коду. Будь-який програміст скаже вам, що в програмуванні більш за все важливо знати певну мову, уміти правильно написати логіку, бути терплячим. І звісно він буде правий, але є один аспект, про який мало говорять – середовище розробки. Чому це важливо?

Перш за все важливо те, що правильна середа розробки може дуже покращити продуктивність розробника. Вона може надавати зручні інструменти, підтримку автодоповнення коду, засоби відлагодження та інші функції, які допомагають прискорити процес розробки. Також дуже важливим аспектом, про який багато хто забуває – це візуалізація коду. Програмісти часто стикаються з такими ситуаціями, коли потрібно працювати над складним завданням/проєктом, і в якийсь момент настає відчуття, що «голова от-от вибухне». В такі моменти, звісно, потрібно відволіктися на щось інше, «перевести дух», але можна посприяти і тому, щоб ця проблема виникала трохи рідше. Хороша візуалізація коду допомагає одразу бачити чітку та зрозумілу структуру коду, що особливо дуже важливо в великих проєктах, де кількість рядків коду перевищує за тисячу.

Для розробки мобільного застосунку на мові Android мною були обрані два середовища розробки: Intellij Idea, та її дочірня IDE – Android Studio.

Intellij IDEA. Це середовище розробки було створено у 2001 році, та швидко набуло популярність, в першу чергу завдяки широкому набору інструментів для рефакторингу. На сьогоднішній день «Ідея» є одним з найпопулярніших середовищ розробки через низку причин.

Як уже було сказано, «Ідея» має широкі можливості **рефакторингу**. Завдяки широкому набору інструментів, програміст може легко автоматизувати рутинні завдання, такі як: перейменування змінних, вилучення методів, оптимізація імпортів тощо. Також потужним інструментом в IDEA є автодоповнення коду, що дуже пришвидчує процес написання. Ще однією перевагою цього середовища є великий набір плагінів, що дозволяють суттєво розширити функціональність «Ідеї». Ви можете встановлювати плагіни для підтримки конкретних технологій, розширення можливостей автодоповнення, інтеграції з іншими інструментами розробки та багато іншого.

Мови, що підтримує Intellij IDEA: Java, Kotlin, Scala, Groovy та інші.

Android Studio. Це середовище розробки було побудоване на базі продукту IntelliJ IDEA. Окрім телефонів та планшетів, на Android Studio можна розробляти ПЗ також і для телевізорів, розумних окулярів, та багатьох інших пристроїв.

Мабуть основною перевагою Android Studio можна назвати **вбудований емулятор**, що дозволяє запускати та тестувати додатки на різних віртуальних Android-пристроях без фізичних пристроїв. Це допомогає швидко перевіряти та налагоджувати додатки на різних конфігураціях пристроїв, що є дуже корисним. Також Android Studio має вбудований візуальний редактор розмітки, що дозволяє графічно створювати інтерфейси користувача додатків. Завдяки цьому можна перетягувати й відпускати елементи, налаштовувати їх властивості та

переглядати, як виглядає інтерфейс під час розробки, що є дуже зручним інструментом.

В цілому Android Studio є **потужним та найкращим** інструментом для розробки Android-додатків.

Чому не Visual Studio чи Eclipse? Звісно вибір середовища розробки є персональним для кожного програміста окремо, і питання «чому саме це, а не це» буде не дуже доречним. Мною було обране середовище розробки «Intellij IDEA» і я описав вище чому, але все ж я вирішив трошки розкрити питання, чим саме мені не подобаються аналоги «Visual Studio» та «Eclipse».

Головною причиною, чому я не надаю перевагу Visual Studio є її суцільна «незграбність». Це середовище розробки «славиться» своєю ресурсозатратністю, для того щоб нормально працювати з ним потрібно мати неабияку систему (нервову теж). Особливо це проявляється, коли робота іде з великими проєктами, де завжди потрібно мати сильний ПК, а тут ще й через середовище розробки дуже затримується весь процес розробки. Visual Studio «полюбляє» підвисати та тормозити, навіть коли справа ще не доходить до компіляції. Також, до цієї «незграбності» можна віднести те, скільки вона важить. Для того, щоб завантажити Visual Studio з потрібними вам плагінами - потрібно виділити хоча б гігабайтів 20, і це надзвичайно багато, порівнюючи з аналогами. Також її недоліком є те, що на інших платформах, окрім Windows, функціонал та підтримка Visual Studio є досить обмеженим, хоча для нас ця проблема не так актуальна.

Eclipse має схожі недоліки. Він також потребує багато ресурсів, і якщо говорити в цілому: хоч це і Open Source проєкт – Eclipse все одно дуже відстає по зручності та функціоналу від «Ідеї».

2.2 **Java або Kotlin**

Коли мова йде про розробку мобільного додатка на Android, не менш важливим пунктом є вибір мови програмування. Вибір є серед двох мов програмування – Java та Kotlin, що, здавалося б, полегшує процес вибору, але все ж обрати мову може бути не найпростішим завданням. Я вирішив порівняти ці дві мови, та пояснити, чому була обрана саме моя.

Java. Java є однією з найпопулярніших мов програмування у світі. Вона була розроблена компанією Sun Microsystems (зараз власність компанії Oracle) і вперше випущена у 1995 році [9]. Так як ми порівнюємо всього дві мови, нижче будуть перераховані переваги мови Java над мовою Kotlin:

- **Переносимість.** Java базується на концепції "Write Once, Run Anywhere" (WOA), що означає, що програми, написані на Java, можуть запускатись на будь-якій платформі, яка підтримує віртуальну машину Java (JVM). Це робить Java особливо привабливою для розробки крос-платформових додатків.
- **Велика екосистема.** Java має велику кількість бібліотек і фреймворків, які значно спрощують розробку програм і прискорюють процес створення програмного забезпечення.
- **Об'єктно-орієнтована мова.** Java базується на об'єктно-орієнтованому підході, що дозволяє розробникам моделювати реальні об'єкти у своїх програмах. Це сприяє покращенню структури коду, полегшує його розуміння та підтримку.
- **Безпека.** Java має вбудовану систему безпеки, яка дозволяє контролювати доступ до ресурсів і захищати програми від потенційно небезпечного коду.

Kotlin. Ця мова програмування розробляється з 2010 року, і головна ціль, яку ставили перед собою розробники – створити мову, що буде лаконічніше ніж Java, і простіша ніж Scala [10].Чи вдалося їм це? Давайте перерахуємо переваги Kotlin над Java, і подивимось:

- **Короткий і зрозумілий синтаксис.** Kotlin має більш компактний і зручний синтаксис порівняно з Java, що дозволяє розробникам писати менше коду без втрати читабельності і збільшення продуктивності.
- **Нульова безпека.** Kotlin має систему типів, що дозволяє виявляти й уникати помилок, пов'язаних з нульовими значеннями (NullPointerException), які є одними з найпоширеніших помилок у Java.
- **Розширення функціональності.** Kotlin має багато вбудованих функцій, які спрощують розробку. Наприклад, підтримка функціонального програмування, розширення функціональності класів без наслідування, інтероперабельність з Java-кодом і багато іншого.
- **Підтримка в IDE.** Kotlin підтримується відомими розробниками IDE, такими як IntelliJ IDEA, Android Studio і Eclipse, що робить його зручним для використання.

Також, порівняння цих двох мов можна спостерігати на рисунку 2.1 з джерела [11]. Хоча Kotlin є більш сучасною та, напевно, зручнішою мовою програмування порівнянно з Java, все ж при розробці додатка була використана саме мова Java. Java залишається широко використовуваною мовою програмування з великою спільнотою розробників і великою кількістю наявних ресурсів, але головною причиною було те, що я знав саме її. :)

В майбутньому, я би все ж вивчив та спробував мову Kotlin, але на даний момент був зроблений саме такий вибір.

Attributes	Java	Kotlin
App Performance	High	Super High
Android Studio 3.0 Support	Partial	Excellent
Code Quality	Not-Optimized	Excellent
Market Presence	Excellent	Good
Adoption Cost	High	Low
App Security	Good	Excellent
Support for Complex Architecture	Excellent	Not Good

Рис. 2.1 – порівняння Java та Kotlin

2.3 **Вибір серверних технологій**

Spring. Spring - це фреймворк і контейнер керування об'єктами для розробки Java-додатків. Spring надає різноманітні компоненти та модулі, які спрощують розробку серверних додатків, забезпечують розширюваність і підтримують добру організацію коду.

В проєкті буде використовуватись саме **Spring boot** [12] – розширення Spring, яке спрощує створення автономних, готових до використання додатків та надає вбудовану конфігурацію та підтримку для багатьох вбудованих технологій, таких як сервери додатків і бази даних. Spring Boot дозволяє **швидко розпочати розробку** з меншою кількістю конфігурації, що є головною причиною використання саме його.

Одна з суттєвих переваг Spring є сервіс Spring MVC, який дозволяє створювати контролери, що обробляють HTTP-запити, а також керувати моделями та представленнями (шаблони веб-сторінок). Варто відзначити,

що Spring MVC за своєю сутністю більше підходить для веб-застосунків, ніж для мобільних серверів.

Java EE (Java Platform, Enterprise Edition). Це офіційний стандарт Java, який більше підходить для розробки маштабних проєктів. Він надає широкий набір специфікацій та API, які допомагають розробникам побудувати масштабовані, надійні та розширювані додатки для підприємств.

Java EE має дві ключові особливості: багаторівневість – всі її застосунки багаторівневі, та вкладенність – всередині Java EE серверів знаходяться контейнери компонентів, всередені яких розміщені, власне, компоненти.

Загальна оцінка: Java EE – **потужний, але застарілий інструмент**.

Jakarta EE. Технологія Jakarta EE, до 2018 року відома як **Java EE**, допомагає поєднати старі та нові технології — скажімо, забезпечити поєднання старих та хмарних додатків у гібридному хмарному середовищі, і дозволяє застосункам ефективно взаємодіяти.

Jakarta EE можна роздивлятись як **міст між «старим» і «новим»** у тому сенсі, що це засіб для роботи з сучасними технологіями без відмови від інвестиций в існуючі додатки та інфраструктуру.

Також можна назвати застарілим інструментом.

Play Framework. Це відкрите програмне забезпечення, яке надає веб-фреймворк для розробки масштабованих та сучасних веб-додатків на мові Java і Scala. Він побудований на принципах "програвання" (play), що означає швидку реалізацію змін та безперервну розробку.

Основними рисами Play Framework є асинхронність та модель факторів, що дозволяє обробляти багато запитів одночасно без блокування потоків. Загалом - це популярний фреймворк, але він знову ж таки більше

підходить для веб-додатків, та є трохи складнішим у використанні ніж Spring.

Micronaut. Сучасний фреймворк, створений для розробки мікросервісів та серверних застосунків на Java. Головна задача, яку ставили перед собою розробники Micronaut – зробити його більш легковажним ніж Spring Boot, тим самим зробивши його біль швидким.

Одними з основних принципів розробників були: **оптимізувати по максимуму час старту, по максимуму оптимізувати пам’ять, яку використовує фреймворк та уникати проксі.**

В цілому фреймворк дуже схожий при використанні на Spring Boot, і більше нагадує його більш сучасне оновлення, але поки що Micronaut все ж поступається Spring Boot, що, можливо, лише питання часу.

Hibernate

Hibernate є фреймворком для об'єктно-реляційного відображення (ORM) в мові Java. Він надає зручний спосіб зберігання, зчитування, оновлення та видалення об'єктів з бази даних, використовуючи об'єктно-орієнтовані підходи. Hibernate спрощує роботу з базою даних, використовуючи маппінг об'єктів на таблиці бази даних і автоматично виконуючи SQL-операції [13].

Переваги Hibernate:

- Спрощена робота з базою даних
- Незалежність від бази даних
- Кешування
- Маппінг об'єктів
- Транзакційна безпеки

Недоліки Hibernate:

- Навчання.
- Продуктивність.
- Складні SQL-запити.

Hibernate є потужним і надійним фреймворком для об'єктно-реляційного відображення в мові Java. Він допомагає знизити складність роботи з базою даних і прискорює розробку. У цього фреймворку є декілька аналогів, таких як: EclipseLink, Spring Data, JPA MyBatis та TopLink, проте Hibernate є, на мою думку, краще за них завдяки великій екосистемі (популярності) та зручним механізмам для маппінгу об'єктів на таблиці бази даних, що дозволяє уникнути пряму та нудну роботу з SQL.

2.4 Вибір технологій, що поєднують клієнт з сервером

Retrofit

Retrofit - це одна з найпопулярніших бібліотек для роботи з мережевими запитами в Android-додатках. Вона надає зручний і простий у використанні спосіб виконання HTTP-запитів до веб-серверів і обробки їх відповідей. Retrofit був створений Square, американською компанією з розробки програмного забезпечення, і вперше був випущений у вересні 2013 року. Розробкою Retrofit займався Як Чеклі, який є одним зі співзасновників Square. [14]

У своєму випуску Retrofit відразу здобув популярність серед розробників Android-додатків. Бібліотека була створена з метою **спростити виконання мережевих запитів у додатках**, а також **забезпечити чистий і зрозумілий інтерфейс для взаємодії з веб-серверами**. Протягом років Retrofit продовжував розвиватися та

отримувати оновлення. З'являлися нові функції, покращення продуктивності та додаткові можливості для розробників в результаті чого Retrofit отримав широке визнання серед спільноти розробників завдяки своїй простоті використання і надійності.

Який принцип роботи Retrofit?

1. Спочатку ви визначаєте інтерфейс, в якому описуєте всі запити до сервера. Кожен метод інтерфейсу відповідає окремому запиту, а аннотації використовуються для налаштування цих запитів, таких як URL, метод запиту (GET, POST і т. д.) та параметри.
2. Потім ви створюєте екземпляр Retrofit, використовуючи Retrofit.Builder.
3. Після створення Retrofit ви отримуєте об'єкт вашого сервісу, використовуючи метод create() на екземплярі Retrofit. Цей об'єкт буде мати всі методи, які ви описали в інтерфейсі сервісу.
4. Тепер ви можете викликати методи вашого сервісу, як будь-які звичайні методи Java. Retrofit автоматично генерує код, щоб виконати HTTP-запити, передати параметри і обробити відповідь сервера.

Які переваги та недоліки Retrofit?

+ **Спрощена розробка.** Retrofit пропонує високорівневий інтерфейс для опису запитів до сервера за допомогою аннотацій. Це дозволяє розробникам зосередитися на логіці додатка, не турбуючись про деталі роботи з мережею. Все, що потрібно зробити, це описати структуру запитів у вигляді інтерфейсу, а Retrofit автоматично згенерує необхідний код.

+ **Інтеграція з конвертерами.** Retrofit підтримує використання конвертерів для автоматичного перетворення даних в формати, зрозумілі для додатка. Він підтримує різні конвертери, такі як Gson, Jackson, Moshi і SimpleXML, що дозволяє легко робити парсинг та серіалізацію об'єктів.

+ **Підтримка асинхронних запитів.** Retrofit надає можливість виконувати мережеві запити асинхронно за допомогою обробників (callback) або використовуючи RxJava або Kotlin Coroutines, що дозволяє уникнути блокування основного потоку і забезпечити швидку та ефективну роботу додатка.

- **Висока кількість налаштувань.** Іноді використання Retrofit може вимагати деяких додаткових налаштувань, особливо при використанні складніших функцій, таких як інтерсептори або заголовки запитів.

- **Обмежена підтримка інших протоколів.** Хоча Retrofit добре підходить для використання з RESTful API, він може бути менш підходящим для інших типів мережевих протоколів, таких як SOAP або GraphQL, що не є мінусом у нашому випадку.

3 Процес розробки

3.1 **Аналіз користувацьких потреб для зручного користування додатком**

Основами зручного користувацького інтерфейсу є **простота, функціональність, «дружелюбність» інтерфейсу, варіативність** та ін. Всі ці пункти можна побачити у будь-якому додатка для замовлення їжі відомих компаній. Забезпечення цих функцій може покращити задоволення користувачів і створити їм зручність та ефективність у процесі замовлення їжі через мобільний додаток, тож цьому повинно бути приділена особлива увага.

Головне меню. Головне меню – це обличчя нашого застосунку. Це перше, що бачить користувач відкривши додаток, тож цьому варто приділити особливу увагу. В головному меню, користувач одразу має зрозуміти з чим має справу, та в нього повинно виникнути почуття інтересу. У нашому застосунку, в головному меню будуть реалізовані спеціальні пропозиції, категорії, та популярні продукти.

Кошик. В цьому пункті користувач зможе переглянути все, що він вже замовив у додатка. Найважливіше, що треба реалізувати при продумуванні корзини – її зрозумілість, щоб користувач одразу міг бачити, що він замовив та у якій кількості.

Історія покупок. В цьому аспекті меню користувач має бачити, що він замовив раніше. Ідеальною реалізацією цього пункту є чіткий та зрозумілий список попередніх замовлень, натиснувши на які, можна побачити більш детальну інформацію.

Профіль. Також важливий пункт, у якому користувач зможе відредагувати дані про себе. Має бути інтуїтивно зрозумілим та зручним.

3.2 Розробка графічного інтерфейсу та перших функцій додатка

Будь-який Android додаток починається зі створення першого «Activity».

За тематикою нашого застосунку були створені перші Activity: MainActivity та LoginActivity. Графічний інтерфейс для кожної Activity описується за допомогою макетів. Важливо відзначити, що макети – xml файли, використовуються в усьому нашому проєкті. Саме ці макети описують графічний інтерфейс нашого додатка, але вони не завдають ніякої логіки. Уся логіка розміщена в шарі MVVM.

LoginActivity говорить сам за себе – він створений для здійснення регістрації та входу в застосунок користувачем. Варто відзначити, що LoginActivity є дуже важливим етапом, адже це перше, що бачить користувач, і саме на цій стадії він «вирішує» чи залишитись у нашому застосунку.

Головною сторінкою є MainActivity, на якій може бути що завгодно, але завдяки попередньому аналізу користувацьких потреб було повністю продумана, що має бути на головній сторінці. Для прототипу, було вирішено на головній сторінці створити список усіх продуктів, які користувач зможе замовити у нашому додатку (рис. 3.1).

Також, в MainActivity, використовуються фрагменти. За своєю сутністю, «фрагменти» є дуже схожим елементом на «Activity», проте в Activity може бути декілька фрагментів, а фрагменти не можуть окремо існувати без Activity. Тому саме це й застосовується: наша MainActivity містить в собі чотири фрагменти, кожен з яких виконує свою функцію: головна сторінка, кошик, історія придбань та профіль.

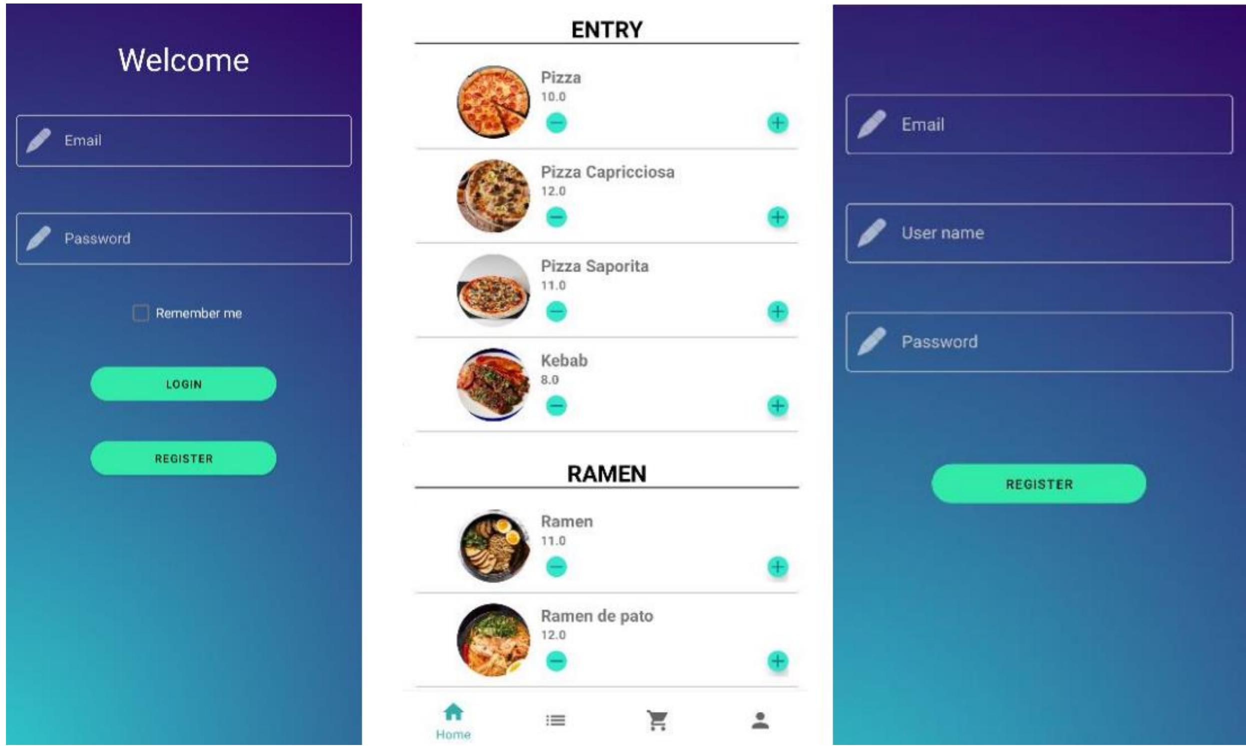

Рис. 3.1 - Перший прототип додатка

Після створення прототипа можна створювати сервер під потреби нашого клієнта.

3.3 **Розробка сервера під потреби Android-клієнта**

Знаючи, завдяки попередньому аналізу, потреби клієнта, можна переходити до створення сервера.

Як було зазначено вище, був обраний Spring boot як фреймворк у додатка, який буде допомагати у розробці. В першу чергу, пам’ятаючи потреби клієнта, потрібно зберігати певні дані про користувачів: E-mail, логін та пароль, ім’я, адреси, номер мобільного телефону, місто, тощо. Також необхідно зберігати дані про продукти, які будуть доступні для замовлення.

Також, для зручності клієнта, необхідно зберігати повну інформацію про персональну історію замовлень кожного користувача. В застосунку реалізована окрема кнопка, натиснув котру користувач зможе легко переглянути свої попередні замовлення.

Саме тому створюється **база даних** для зберігання всієї цієї інформації. Так як в найближчих планах застосунку немає роботи з величезною кількістю даних – було вирішено використовувати зручну бібліотеку **Hibernate.** Можливості, переваги та недоліки цієї бібліотеки були вже описані у пункті 2.3. Варто лиш відзначити, що найбільше у чому вона допомагає при розробці – це створення бази даних та у дуже легкому доступі до неї: можна уникнути комплексні нативні SQL-запити, та писати замість них більш прості рішення, які надає Hibernate.

Далі у роботу вступає Spring Boot. Найбільше у чому він нам допомагає – уникнути налаштування конфігурації сервера, та одразу перейти до створення логіки додатка. Схема роботи Spring Boot зображена на рисунку 3.2 з джерела [15].

Рис. 3.2 – Логічна складова проєкту

Логіка додатка.

На даному етапі вже створена база даних та моделі DAO у проєкті. Далі переходимо до найважливішої частини будь-якого сервера, що використовує архітектурний стиль REST - створення контролерів.

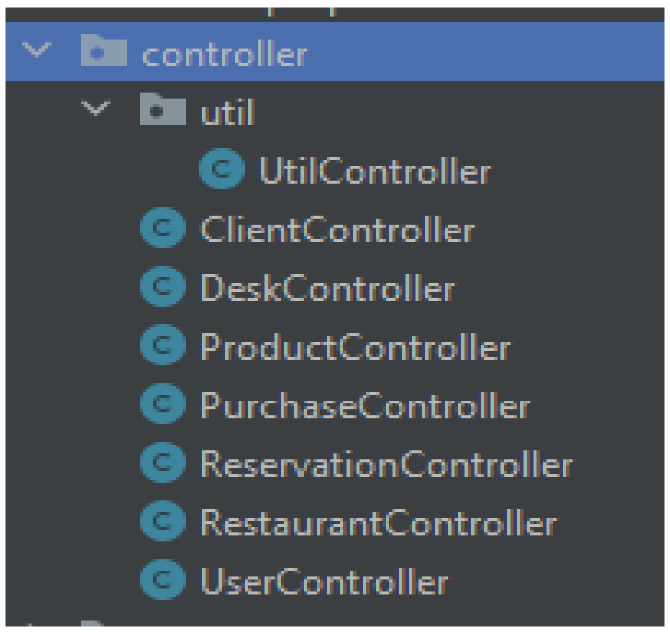

Рис. 3.3 – Контролери у проєкті

Кожен з контролерів на рисунком 3.3 має свою конкретну задачу. Наприклад "ClientController" виконує роль по передачі даних усіх наших користувачів, таких як: реєстрація, логін, паролі у зашифрованому вигляді та ін. "ProductController" передає усю інформацію про товари, які користувач може замовити. "PurchaseController" використовується для того, щоб користувач міг замовити їжу, отримати усі дані про історію замовлень, відредагувати дані замовлення і тд. Отже, всі контролери виконують певну важливу роль, та є невідмінною частиною застосунку.

Для того, щоб усі контролери працювали саме так, як цього потребує наш користувач існує певний рівень між контролерами та базою даних: **сервіси.**

Сервіс – це рівень, у якому виконується вся логіка сервера. Тут відбуваються такі процеси, як: зберігання даних у нашій базі, перетворення усіх даних на необхідну нам модель, яку відправлятиме контролер, верифікація користувача через логін або реєстрацію та інше.

3.4 **Завершення процесу розробки**

Тепер, коли готовий сервер та прототип нашого додатка – можна приступити до фінальної стадії розробки, а саме до створення тих функцій, яких не було в прототипі, а також до завершення розробки графічного інтерфейсу у тому вигляді, яким його уявляли на етапі планування.

На фінальному етапі, у фрагменті головного меню, додаються такі речі, як: список напопулярніших продуктів, категорії, пошук потрібних товарів та банер, на якому можна побачити спеціальні пропозиції нашого ресторана. Остаточний вигляд головного меню можна побачити на рисунку 3.5. Також повністю реалізується функцію логіну та реєстрації (рис. 3.4), так як є вже повністю готовий сервер з усією необхідною логікою. До всього цього треба додати невеличку функцію – збереження даних. Завдяки цій функції, користувачу не потрібно буде вводити щоразу логін та пароль під час входу в застосунок.

Для реалізації безпеки користувача додатка в проєкті було створено криптографічний алгоритм шифрування паролів «PBKDF2WithHmacSHA1». Це однонаправлений шифрувальшик хеш-функцій, який неможливо розшифрувати (навіть розробнику алгоритму) з зашифрованого вигляду. Приклад реалізації у вигляді коду можна побачити у наступному додатка А.

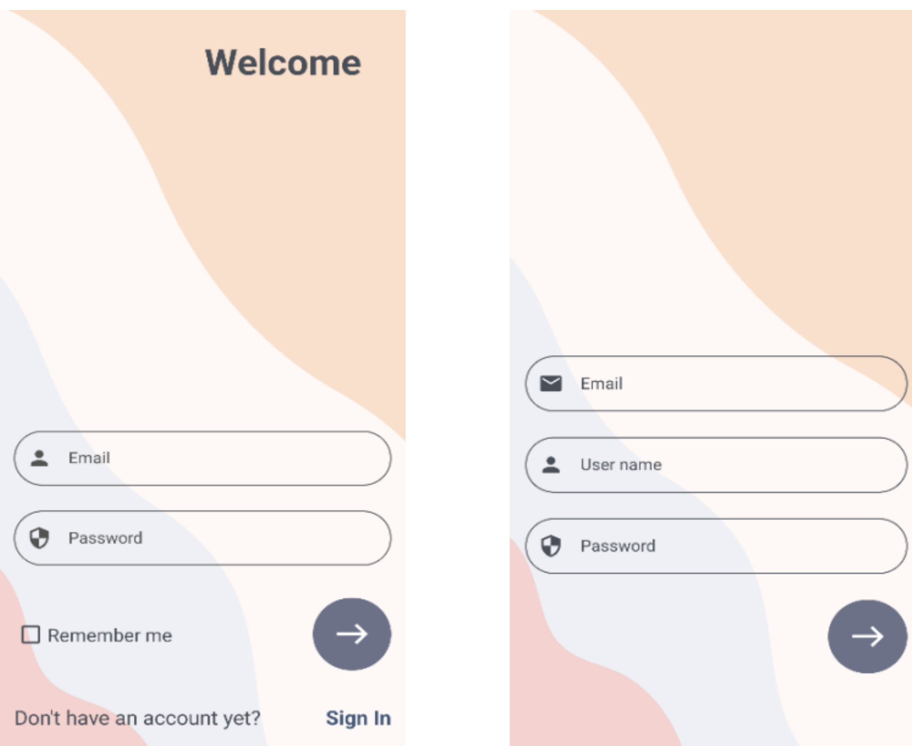

Рис. 3.4 – Інтерфейс логіну

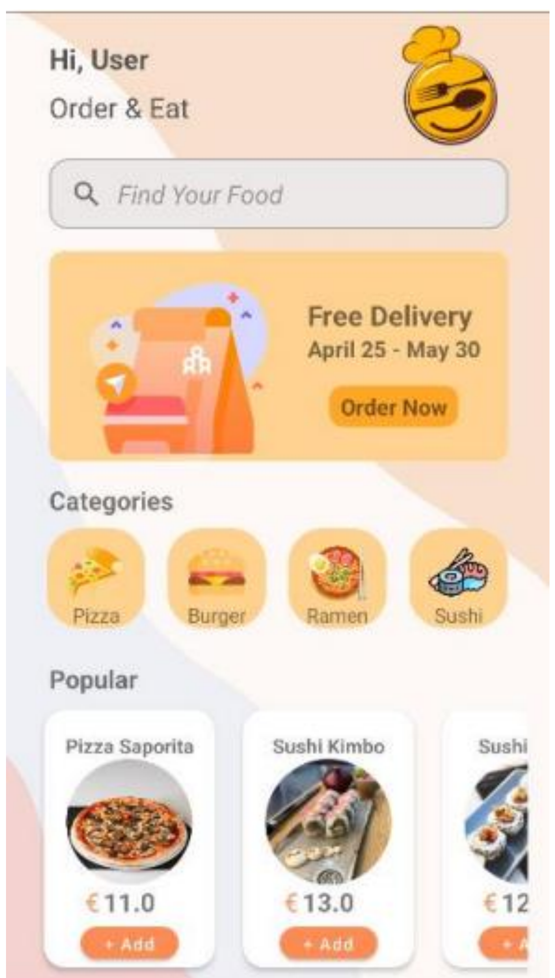

Рис. 3.5 – Головне меню застосунка

Наостанок залишається зробити персональне місце для наших користувачів – сторінка профілю (рис. 3.6). В ній користувач зможе вести більш детальну інформацію про себе, відредагувати дані та вийти зі свого акаунту. Як було сказано в аналізі – профіль має бути інтуїтивно зрозумілим нашому користувачу та зручним. Зайшовши в цей підпункт, юзер одразу має бачити куди йому треба натиснути, щоб увести зміни.

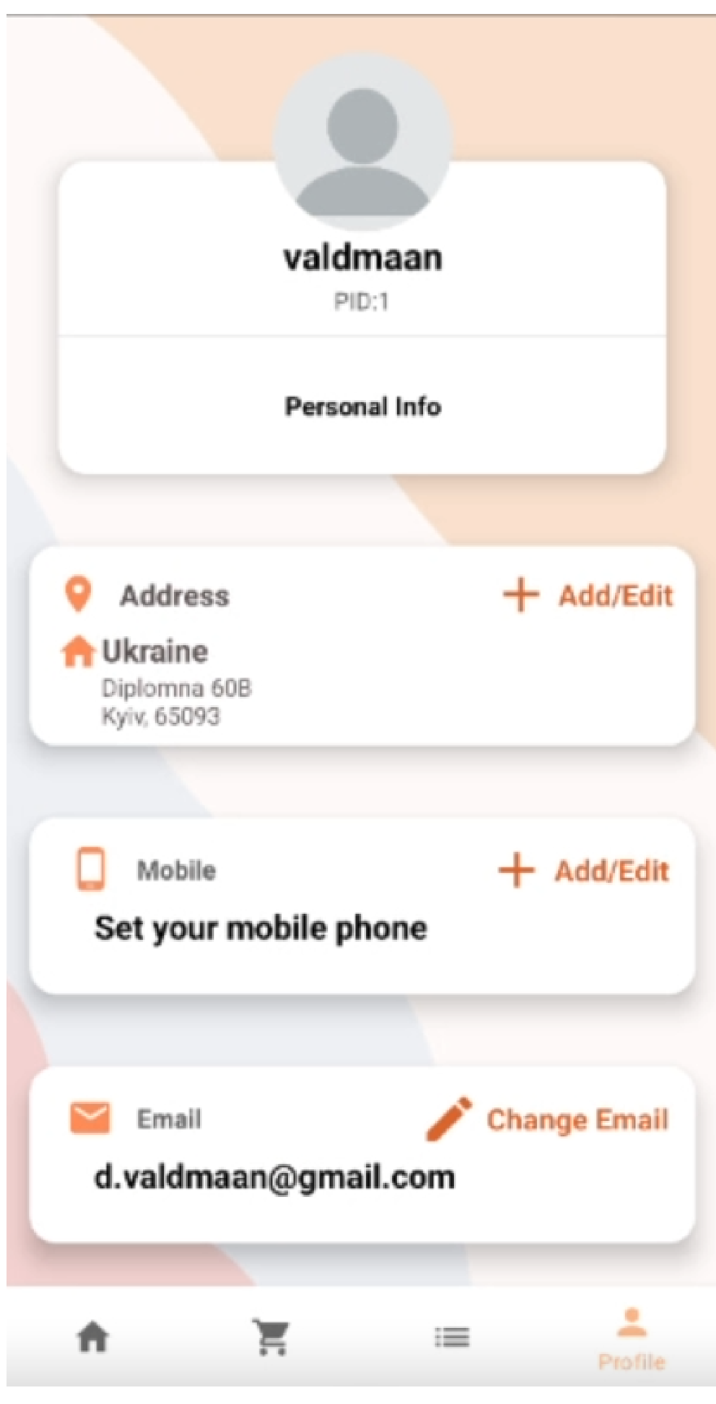

Рис. 3.6 – Персональна сторінка

3.5 Подальший розвиток додатка

Показана версія додатка, звісно ж, не є остаточною. Остаточної версії напевно не буде ніколи: технології розвиваються, прогресують, тому завжди буде можливість для змін на краще.

Ось приблизні зміни, у найближчому майбутньому:

Patch 1.1 Список оновлень:

- Буде додано автоматичну пропозицію заповнити адресу, якщо користувач її не має;
- Буде додано пошук товарів за першим словом у головному меню;

Patch 1.2 Список оновлень:

- Буде додано автоматичну зміну статусу замовлення;
- Буде додано функцію редагування адреси після здійснення замовлення;

Patch 1.3 Список оновлень:

- Буде додано перелік країн та міст при заповненні інформації на персональній сторінці;
- Буде додано функцію автодозаповнення при написані введених раніше персональних даних;

Patch 1.4 Список оновлень:

- Планується покращення пункту «Історія замовлень»: детальна інформація про замовлення показується після натискання на замовлення;
- Буде створено алгоритм автоматичного визначення смаків користувача, який аналізує його попередні замовлення у додатка;

Patch 1.5 Список оновлень:

- Буде додано функцію «Залишити відгук про страву»;

Висновки

Метою цієї роботи була розробка фулстек застосунку для замовлення їжі на платформі Android. Для виконання цієї задачі:

- Досліджено ринок застосунків для замовлення їжі;
- Проаналізовано додатки відомих компаній на предмет дизайну;
- Підібрано найзручніше середовище розробки;
- Обрано мову написання коду;
- Досліджено інформацію про серверні технології: Spring, Hibernate, Micronaut та ін.
- Досліджено інформацію про бібліотеку Retrofit;
- Проаналізовано потреби користувача при користуванні додатком;
- Досліджено процес розробки Android додатків з клієнт-серверною архітектури.

Завдяки попередньому плануванню та чіткому виконанню завдань, було розроблено функціональний застосунок на платформі Android для замовлення їжі. Додаток містить у собі низку різних технологій, а також має гарно продуманий графічний інтерфейс. Ми змогли дізнатись, як великі компанії, такі як Glovo, Uber Eats та інші створють застосунки на багатомільйонну аудиторію, та розробити прототип такого застосунку. Створений додаток можна використовувати в комерційних цілях, попередньо додавши декілька дрібних функцій та реалізувавшик росплатформовість. В цілому, ми маємо гарний результат виконання нашої роботи.

Перелік джерел посилання

1. Restaurant Mobile App Development Guide: All You Need to Know [Електронний ресурс] - Режим доступу до ресурсу: https://www.fatbit.com/fab/restaurant-app-development/
2. What is Client-Server Architecture? Everything You Should Know [Електронний ресурс] - Режим доступу до ресурсу: https://www.simplilearn.com/what-is-client-server-architecture-article
3. Введение в GraphQL [Електронний ресурс] - Режим доступу до ресурсу: https://dou.ua/lenta/articles/working-with-qraphql/
4. What is REST API? [Електронний ресурс] - Режим доступу до ресурсу: https://www.redhat.com/en/topics/api/what-is-a-rest-api
5. Comparing SOAP vs REST vs GraphQL vs RPC API | AltexSoft [Електронний ресурс] - Режим доступу до ресурсу: https://www.altexsoft.com/blog/soap-vs-rest-vs-graphql-vs-rpc/
6. MVC, MVP and MVVM – Tomy's Blog [Електронний ресурс] - Режим доступу до ресурсу: https://tomyrhymond.wordpress.com/2011/09/16/mvc-mvp-and-mvvm/
7. Порівняння MVP, MVC, MVVM [Електронний ресурс] - Режим доступу до ресурсу: https://www.dz-techs.com/ru/mvc-mvp-mvvm-which-choose
8. Рекомендації по створенню дизайну від Google [Електронний ресурс] - Режим доступу до ресурсу: https://developers.google.com/assistant/interactivecanvas/design?hl=ru
9. Java language [Електронний ресурс] - Режим доступу до ресурсу: https://www.java.com/ru/
10. Comparsion to Java [Електронний ресурс] - Режим доступу до ресурсу: https://kotlinlang.org/docs/comparison-to-java.html

11. Kotlin vs Java: what is the best way to teach Android as a developer? [Електронний ресурс] - Режим доступу до ресурсу: https://lampalampa.net/en/java-vs-kotlin-which-is-better-for-android-apps-development/
12. Spring boot reference documentation [Електронний ресурс] - Режим доступу до ресурсу: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/
13. Документація Hibernate [Електронний ресурс] - Режим доступу до ресурсу: https://hibernate.org/orm/documentation/6.2/
14. Опис Retrofit [Електронний ресурс] - Режим доступу до ресурсу: https://guides.codepath.com/android/consuming-apis-with-retrofit
15. Standard Project Structure for Spring Boot Projects [Електронний ресурс] - Режим доступу до ресурсу: https://www.javaguides.net/2019/01/standard-project-structure-for-spring-boot-projects.html
16. How to create an app on Android in 2022 [Електронний ресурс] - Режим доступу до ресурсу: https://www.designware.io/blog/design-android-app
17. Брайсон Пейн – легкий спосіб вивчити Java, 2019.
18. What are web-services? [Електронний ресурс] - Режим доступу до ресурсу: https://www.cleo.com/blog/knowledge-base-web-services
19. MySQL documentation [Електронний ресурс] - Режим доступу до ресурсу: https://dev.mysql.com/doc/

Додаток А

Фрагмент коду алгоритму шифрування паролів

```
package com.example.restaurant.encryptor;

import…

public class Encryptor {

   public static String generateStrongPasswordHash(String password)
         throws NoSuchAlgorithmException, InvalidKeySpecException {
      int iterations = 1000;
      String sr = ("SHA1PRNG");
      char[] chars = password.toCharArray();
      byte[] salt = sr.getBytes();

      PBEKeySpec spec = new PBEKeySpec(chars, salt, iterations, 64 * 8);
      SecretKeyFactory skf =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");

      byte[] hash = skf.generateSecret(spec).getEncoded();
      return iterations + ":" + toHex(salt) + ":" + toHex(hash);
   }
```

```
    private static String toHex(byte[] array) {
        BigInteger bi = new BigInteger(1, array);
        String hex = bi.toString(16);

        int paddingLength = (array.length * 2) - hex.length();
        if (paddingLength > 0) {
            return String.format("%0" + paddingLength + "d", 0) + hex;
        } else {
            return hex;
        }
    }
}
```