МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система онлайн замовлень в закладах швидкого харчування»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студента групи ІНм – 91н **Мірошниченко В.І.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Мірошниченку Вячеславу Ігоревичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система онлайн замовлень в закладах швидкого харчування*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд технологій, що застосовуються для продажу товарів через Інтернет; 2) Постановка завдання й формування завдань дослідження; 3) Огляд технологій, що використовуються під час розробки додатків на ОС Android, Spring Framework, React; 4) Моделювання системи онлайн замовлень; 5) Розробка додатку; 6) Аналіз результатів.

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

_____

_____

_____

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд технологій, що застосовуються для продажу товарів через Інтернет		
2.	Постановка задачі та формування завдань дослідження.		
3.	Опис архітектури додатку		
4.	Розробка додатку з використанням GraphQL		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 68 стор., 27 рис., 4 таблиці, 1 додаток, 21 літературних джерел.

Об’єкт дослідження — Інформаційна система онлайн замовлень в закладах швидкого харчування.

Мета роботи — розробка додатку на ОС Android, Spring Framework, React, GraphQL для замовлення в закладах швидкого харчування.

Результати — проведений аналіз літератури, методів та інструментів, які дозволяють проводити замовлення різних товарі, обмін командами і даними між додатками, що написані на різних мовах програмування під різні платформи, вивчені особливості їх застосування у системі Android та backend, для різних типів користувачів. Після ознайомлення з існуючими рішеннями було розроблено алгоритм програми та її реалізація у вигляді додатку для мобільних пристроїв на ОС Android. Дана розробка дозволяє виконувати замовлення в закладах швидкого харчування будь-яким користувачам, а також дозволяє створювати власні сторінки для магазинів-користувачів. Додаток був реалізований за допомогою мов програмування Kotlin та JavaScript.

ІНФОРМАЦІЙНА СИСТЕМА ОНЛАЙН ЗАМОВЛЕНЬ В ЗАКЛАДАХ ШВИДКОГО ХАРЧУВАННЯ, INFORMATION SYSTEM FOR ONLINE ORDERING IN FAST FOOD RESTAURANTS, SPRING, GRAPHQL, POSTGRESQL

ЗМІСТ

ВСТУП 6

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 8

1.1 Дослідження актуальності проблеми 8

1.2 Аналіз аналогів та визначення наявних проблем 10

1.3 Постановка задачі 15

2 ОГЛЯД АРХІТЕКТУРИ ТА ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ 17

2.1 Огляд основних ідей REST та GraphQL архітектури 17

2.2 Вибір мов програмування 19

2.3 Вибір фреймворків для бекенда та мобільного додатку 20

2.4 Спосіб автентифікації користувачів 21

2.5 Вибір СУБД 25

2.6 Вибір платіжної системи 26

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ 30

3.1 Інформаційна модель 30

3.2 Програмна реалізація 31

3.3 Обробка помилок від сервера з системою типів GraphQL 34

3.4 Користувацький інтерфейс 37

ВИСНОВКИ 54

СПИСОК ЛІТЕРАТУРИ 55

ДОДАТКИ 57

Додаток А. Лістинг програмного коду 57

ВСТУП

Завдяки активному розвитку інформаційних технологій, протягом останніх десятиліть з величезною швидкістю зростає кількість даних, накопичених людством, котрі доступні в електронному вигляді. База цих даних з'явилася в результаті постійного використання світової мережі Інтернет, можливість користуватися якою є у більшості жителів усіх розвинених країн світу, котра значно спростила та пришвидшила доступ до інформаційних ресурсів навіть з найвіддаленіших точок земної кулі.

Хоча треба зважувати той факт, що більшість доступної інформації не несе жодної користі для пересічної особи, оскільки людський розум не в змозі знайти та обробити таку кількість неупорядкованих даних. Таким чином виникає проблема, коли з великої кількості доступних, але необроблених даних, необхідно вилучити та агрегувати потрібну інформацію для користувача.

Не дивно, що дану проблему уже вирішують досить давно і ми використовуємо пошукові системи кожен день. В загальних випадках проблему пошуку інформації можна сказати уже вирішили, але є узько-направлені області, де ця проблема є актуальною. Одною з таких областей є заклади швидкого харчування. В більшості випадків про них важко знайти інформацію в інтернеті, не говорячи вже про те щоб зробити замовлення онлайн. Звичайно, що дану інформацію потрібно надати користувачеві в якомога більш зручному форматі, а так як останнім часом спостерігається загальний тренд - перехід користувачів ПК на мобільні платформи, тому наголос буде ставитися на мобільні пристрої.

З іншого боку, дані заклади в основному не мають можливості купити собі програмне забезпечення котре буде збирати їм аналітику по всім користувачам, котрі фактично самі надають її, використовуючи додаток. Приклади таких даних:

- історія перегляду на конкретному закладі;
- час перебування на сторінці того чи іншого товару;
- загальна кількість відвідувань;

- інформація про те, звідки користувачі потрапляють на сторінку закладу(зі сторінок з рекламою, з пошукових сторінок);
- оцінка товару, коментарі та відгуки, які залишають користувачі;
- теплова карта активності користувачів (на якій частині сторінки найдовше залишається користувач – читає характеристики товару, чи переглядає його фото).

Маючи в своєму арсеналі відповідні засоби для аналізу отриманих даних, адміністрація закладів може виявляти певні закономірності, та прогнозувати поведінку покупців, наприклад після підвищення цін, проведення певної рекламної компанії чи анонсу знижок.

Таким чином, актуальною є розробка засобів, що дозволять не лише знаходити інформацію про заклади швидкого харчування для користувача, а й дасть можливість онлайн замовлення, що в свою чергу дасть можливість для закладів проводити більш справедливу конкуренту боротьбу, так як вся рейтингова інформація буде доступна для користувача і результат пошуку буде будуватися на основі цього рейтингу. Також для закладів буде доступна персональна аналітика.

Мінімалістичний дизайн мобільного додатку виконаний в стилі Material UI дозволить сконцентрувати увагу саме на потрібних товарах, аналізу їх характеристик зовнішнього вигляду та цін.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження актуальності проблеми

На сьогодні більшість людей наукової сфери українського суспільства стверджує, що сучасний український споживач давно сприяє розвитку постіндустріалізму. Вплив культурної глобалізації не можливо не помітити. Україна вже стала частиною бурхливого прогресу інформаційного суспільства, також відбувається поширення ідеології споживача. Тому практики споживання стають головним предметом вивчення науковців.

Тепер термін “суспільство споживання”, можна чути майже кожного дня так як масове споживацтво розвивається разом з капіталізмом дуже швидкими темпами, який супроводжується бурхливим технічним та економічним розвитком та такими соціальними змінами, як: ріст доходів, який суттєво змінює структуру споживання, розмивання класової структури, зменшення тривалості робочого дня індивідуалізація споживання.

В сучасному баченні Українське споживання є однією з головних сфер суспільства, оскільки є фундаментом для створення ідентичності, прояву соціального статусу, ідентичності, способу життя. Повсякденна практика показує що символічне споживання та закріплення людиною певного статусу в суспільстві для презентації, ідентифікації соціальної позиції за рахунок зовнішніх ефектів спожитих речей вже давно закріпилася. Під впливом інституціоналізації символічного споживання перебуває формування споживчих стандартів (житло, дозвілля, побутова, комп’ютерна та мультимедійна техніка, аксесуари та одяг).

Стрімкий розвиток Інтернет в Україні сприяє поширенню електронної комерції. Згідно з результатами досліджень (2009–2011 рр.), в Україні більша частина населення середнього класу вдається до е-комерції, з них: 40% – оплачували інформацію; 19% – здійснювали купівлю речей; 15% – оплачували розваги; 8% – проводили фінансові операції; по 7% – оплачували подорожі, туризм та культурні потреби.

Можна стверджувати що український ринок електронної комерції зріс на 400% за останні 5 років згідно підрахункам експертів у кінці 2014 р.

Оцінки компанії “GfK” показують, що кількість українців , які купують в Інтернеті, уже перевищила число в 2,8 млн осіб. В десятку найпопулярніших ресурсів, які показали найбільший попит у української аудиторієї, починаючи з 2013 р., входять інтернет-магазини Rozetka.com.ua і ресурс безкоштовних оголошень Olx.ua (Slando.ua).

Також, нещодавно Monobank провів досить цікаве дослідження (Таблиця 1.1), в котрому була порахована статистика по кількості покупок у різних категоріях. На даний момент Monobank обслуговує близько 3 000 000 клієнтів.

Таблиця 1.1 Кількість покупок у категорії

Категорії	%
Продукти та супермаркети	40.5
Кафе та ресторани	**13.5**
Подорожі	6.99
Поповнення мобільного	6.46
Інше	5.55
Краса та медицина	5.18
Розваги та спорт	4.74
Авто	4.48
Таксі	4.27
Комуналка та інтернет	3.43
Одяг та взуття	1.53
Ремонт	1.17
Кіно	0.88
Побутова техніка	0.52
Тварини	0.25
Бюджет та податки	0.21
Книги	0.19

Вся інформація вище говорить про те, що на даний час вирішення проблем пошуку кафе, ресторанів, закладів швидкого харчування та замовлень в них є досить актуально і кількість користувачів буде достатньо високою, так як середньостатистичний українець користується закладами харчування мінімум декілька разів на тиждень.

По суті, реалізований сервіс буде автоматизувати купівлю товарів в закладах харчування, що є досить проривним кроком для даного ринку, особливо в Україні, де інфраструктура для купівлі та замовлення даних товарів не досить добре влаштована. За приблизними підрахунками, реалізований сервіс буде зберігати до 1-2 годин часу для кожного користувачі за один тиждень.

1.2 Аналіз аналогів та визначення наявних проблем

Для початку було проведено дослідження маркетплейсів(Таблиця 1.2), котрі продають різні товари в інтернеті і були вибрані наступні аналоги:

1) Rozetka.com.ua — найбільший інтернет магазин України, також дає можливість іншим магазинам/постачальникам продавати свої товари;

2) Mister.am - інформаційна система онлайн замовлень їжі з доставкою;

3) Klunok.com.ua - мережа продовольчих крамниць, яка має службу доставки продуктів і оплату продуктів через інтернет.

Порівняльна характеристика буде проводитися за наступними характеристиками :

1) дизайн;

2) зручність використання;

3) коректна робота функціоналу;

4) система рейтингу(оцінки якості товару, та продавця);

5) адаптивність під різні розміру екрану, наявність мобільної версії;

6) опція порівняння товарів;

7) кнопки соціальних мереж, як для поширення інформації про товар, так і регістрації за допомогою соціальних мереж;

8) можливість додати інші продавців;

9) наявність онлайн чату.

Таблиця 1.2 – Порівняльна характеристика маркетплейсів

	Rozetka.com.ua	Mister.am	Klunok.com.ua
Дизайн	Добре продуманий. Кольори підібрані правильно, їх не забагато. Шрифти не розпливаються, добре зчитуються.	Все добре, як і на Rozetka, крім шрифтів, в деяких місцях вони зливаються з фоном.	Не вистачає кольорів для виділення головної інформації. Товари погано розділенні між собою. Також, наявні проблеми зі шрифтами
Зручність(1-10)	10	8 (мало фільтрів для пошуку потрібного товару, не має інформації про наявність товару)	7 (взагалі не зрозуміло як придбати товар, відсутня корзина)
Функціонал(1-10)	10	10	6 - Не коректно працює регістрація, не працює оплата через інтернет
Система рейтингу(+, -)	+- (відсутня система рейтингу продавця)	+- (відсутня система рейтингу товару, не можна залишати коментарі)	-
Адаптивність(+, -)	+	+	-
Порівняння тов.(+, -)	+	-	-
Соціальні мережі(+, -)	+	-	-
Інші продавці(+, -)	+- (у продавців не має власної сторінки)	+ (недоліком є неможливість замовлення страв	-

		з декількох ресторанів одночасно)	
Онлайн чат(+, -)	-	+	-

Незважаючи на недоліки ці сервіси дійсно допомагають обрати та зробити ту чи іншу покупку, явним лідером є Rozetka. Так як ці маркетплейси не зовсім виконують поставлене завдання, було вирішено провести ще одне дослідження і пошук мобільних додатків/сервісів котрі уже дозволяють замовляти і забирати самостійно товари з ресторанів. Найпопулярніші в Україні зараз Glovo та Rocket. Порівняльна характеристика представлена в Таблиці 1.3

Таблиця 1.3 – Порівняльна характеристика Glovo та Rocket

	Glovo	Rocket
Оцінка в AppStore	4.8 (98500 користувачів)	4.6 (17 000 користувачів)
Оцінка в Play Market	4,3 (552 136 користувачів)	4,7(15 551 користувачів)
Місце в категорії в AppStore	2 в Lifestyle	2 в Food & Drink
Оплата	Готівка або карта, для додавання карти потрібно ввести 3 поля	Готівка або карта, для додавання карти потрібно ввести 5 полів
Підтримка	Не зручна, орієнтована на ще щоб користувач, як можна довше діставався до оператора	Можна використати вайбер, телеграм або зателефонувати. Митєве підключення до оператора
Точність часу виконання	Дуже приблизна, наприклад 30-40 хвилин, хоча замовлення може бути готовим через 10 хвилин	Дуже приблизна, наприклад 30-40 хвилин, хоча замовлення може бути готовим через 10 хвилин
Комісія для рестрорану від замовлення	Близько 30% + НДС	Близько 30% + НДС
Коректна робота функціоналу	Помилок не було помічено	Помилок не було помічено

Пошук найближчого закладу	Наявний в самому верху категорій, що досить зручно. Не зрозуміло за якою характеристикою були відсортовані заклади в самій категорії близьких до користувача закладів. Також не зрозуміло, яка відстань до закладу.	Наявний, але потрібно довго шукати серед інших категорій. Не зрозуміло за якою характеристикою були відсортовані заклади в самій категорії близьких до користувача закладів. Також не зрозуміло, яка відстань до закладу
Пошук закладу по карті	Відсутній	Відсутній
Масштабування	Підписується контракт з ресторанами, основна ставка йде на доставку. Інтегрування з системою обробки замовлень.	Підписується контракт з ресторанами, основна ставка йде на доставку. Інтегрування з системою обробки замовлень. Є багато вигідних знижок, тому ресторани більш охоче йдуть на партнерство.
Коректна обробка помилок	Додаток показує, що за помилка і направляє користувача по шляхам вирішення.	Була помилка, не можливість під’єднати карту до додатку, додаток просто не додав карту без повідомлень
Реферальна програма	Відсутня.	Можна запросити друга і отримати 70 грн знижку
Програма лояльності	Glovo Premium можна частково вважати, як програма лояльності, але по факту це модель оплати по підписці.	Відсутня.
Маркетинг	Зазвичай, це рекомендації. Компанія використовує так званий performance marketing, але не так активно як rozetka. Кількість витрачених коштів компанія не розголошує.	Дуже багато коштів в маркетинг, за неофіційною інформацією це сотні тисяч доларів кожен місяць. Є навіть інтеграції з відомими співаками та артистами.
Інформація про калорійність продуктів	Відсутня	Відсутня

Інформація про склад товарів	В основному відсутня	В основному відсутня
Вибір продукції	Основні пропозиції ресторану, зазвичай всі ті страви котрі можна зручно доставити.	Основні пропозиції ресторану, зазвичай всі ті страви котрі можна зручно доставити.
Система рейтингу (оцінки якості товару, та продавця, можливість для залишення коментарів)	Досить не точна, так як є тільки два значення при оцінці, також не показана кількість користувачів, котрі проставили рейтинг.	Оцінка по 5 бальній шкалі, є можливість залишити коментар. Наявна кількість користувачів, котрі проставили рейтинг.
Кількість “кроків” для замовлення (воронка продажу)	Довга регістрація, інше як і в Rocket.	Можна замовити навіть без регістрації, якщо оплата готівкою. Зареєструватися можна уже після того, як знайшов потрібний товар.

Зважаючи на всі характеристики вище, можна зробити наступні висновки по недолікам:

1) Жодна з компаній не піклується про точний час приготування замовлення, тому користувач по факту повинен зразу йти за своїм замовленням, якщо хоче забрати його гарячим.
2) Не має можливості дізнатися рейтинг конкретного товару.
3) Дуже висока комісія для закладів швидкого харчування, тому не великих кафе не має доступних в додатку. Сервіси це пояснюють високою ціною на те щоб отримати клієнта і ціною доставки(хоча доставка при самовивозі не потрібна).
4) Не можливо замовити кафу, чай, так як доставка їх складна і дуже чутлива до часу.
5) Оплата є в обох сервісах, але не має Google Pay, Apple Pay, щоб змогло значно пришвидшити замовлення і зменшити кількість помилок.
6) Погано оформлена інформація про товар, відсутній склад та калорійність.

7) Відсутній пошук закладів по карті.
8) Реферальна програма є в Rocket, але її досить складно знайти в додатку.
9) Програма лояльності майже відсутня.

Основною задачею після порівняння є врахування всіх недоліків цих сервісів і запобігання їх у реалізованій системі.

1.3 Постановка задачі

Метою роботи є розробка програмної системи, котра буде представляти собою мережу закладів швидкого харчування, на кшталт маркетплейсу де основними типами користувачів будуть:

1) заклади швидкого харчування, котрі обов'язково реєструються в системі з переліком товарів, які вони хочуть продавати;

2) покупці, котрі використовують систему для купівлі товарів, також можуть використовувати систему, як засіб для отримання актуальної інформації про товари того чи іншого магазину.

Система дозволить будь якому закладу створити свою власну сторінку з товарами, в якій можна буде вибрати товари з сумісного списку або добавити свої власні. Також, кожний магазин, буде мати змогу обслуговувати клієнтів через даний сервіс, тим самим скоротивши черги.

Після регістрації достатньої кількості користувачів, система дозволить порівнювати ціни на товари з різних закладів, повідомляти користувача про різноманітні зміни з цільовими характеристиками товарів. Також в майбутньому буду додано систему рейтингів, де кожен користувач зможе залишити свій коментар про проведену купівля і поставити оцінку закладу в котрому було проведено замовлення. Для досягнення поставленої мети сформульовані наступні задачі:

1) вибрати відповідні методи та технології для:
   1.1) авторизації та автентифікації користувачів;
   1.2) збереження інформації про заклади;

1.3) побудови статистики цін;

1.4) збереження інформації про товари;

1.5) створення серверної частини на базі Spring фреймворку;

1.6) створення графічного клієнту на базі React.

2) провести аналіз платіжних систем;

3) спроектувати передачу даних між компонентами системи;

4) спроектувати програмну систему та її частини;

5) розробити та протестувати систему в цілому.

2 ОГЛЯД АРХІТЕКТУРИ ТА ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ

2.1 Огляд основних ідей REST та GraphQL архітектури

GraphQL часто представляють, як революційно новий шлях осмислення API. Замість роботи з жорстко визначеними на сервері кінцевими точками (endpoints) ви можете за допомогою одного запиту отримати саме ті дані, які вам потрібні. І так - GraphQL гнучкий при впровадженні в організації, він робить спільну роботу команд мобільного додатку і backend розробки гладкою, як ніколи раніше. Однак на практиці обидві ці технології мають на увазі відправку HTTP-запиту і отримання якогось результату, і всередині GraphQL вбудовано безліч елементів з моделі REST.

Подібність:

1) Є поняття ресурсу, є можливість призначати ідентифікатори для ресурсів.

2) Ресурси можуть бути вилучені за допомогою GET-запиту URL-адреси по HTTP.

3) Відповідь на запит може повертати дані в форматі JSON.

4) Список кінцевих точок в REST API схожий на список полів в типах Query і Mutation, використовуваних в GraphQL API. Обидва вони є точками входу для доступу до даних.

5) Є можливість розрізняти запити до API на читання і на запис даних.

Різниця: в REST викликана користувачем кінцева точка (endpoint) - це і є сутність об'єкта. У GraphQL сутність об'єкта відокремлена від того, як саме цей об’єкт отримується.

6) Різниця: в GraphQL ви можете всередині одиночного запиту переміщатися від точки входу до пов'язаних даними, слідуючи зв'язків, певним в схемі. У REST для отримання пов'язаних ресурсів вам доведеться виконати запити до кількох кінцевих точок.

7) Кінцеві точки в REST і поля в GraphQL в кінцевому підсумку

викликають функції на сервері.

8) Як REST, так і GraphQL зазвичай покладаються на фреймворки і бібліотеки в частині рутинної роботи по організації мережевої взаємодії.

Відмінності:

1) В REST структура і обсяг ресурсу визначаються сервером. У GraphQL сервер визначає набір доступних ресурсів, а клієнт вказує необхідні йому дані прямо в запиті.

2) В GraphQL немає різниці між полями типу Query і полями будь-якого іншого типу, за винятком того, що в корені запиту доступний тільки тип query. Наприклад, у вас в запиті будь-яке поле може мати аргументи. У REST не існує поняття першого класу в разі вкладеного URL.

3) В REST ви визначаєте запис даних, змінюючи HTTP-метод запиту з GET на щось на зразок POST. У GraphQL ви міняєте ключове слово в запиті.

4) В REST кожен запит зазвичай викликає рівно одну функцію-обробник маршруту. У GraphQL один запит може викликати безліч функцій для побудови складної відповіді з безліччю вкладених ресурсів.

5) В REST ви будуєте форму відповіді самостійно. У GraphQL форма відповіді визначається бібліотекою, яка виконує GraphQL, для зіставлення формі запиту.

Опираючись на перечисленні вище характеристики, деякі з відмінностей говорять на користь GraphQL. Зокрема, дійсно корисно мати можливість реалізувати свій API, як набір невеликих функцій-розпізнавачей, а потім відправляти складні запити, які передбачуваним способом витягують безліч ресурсів за один раз. Це убезпечує розробника API від необхідності створювати безліч кінцевих точок з різною формою, а клієнту API дозволяє уникнути вилучення зайвих даних, які йому не потрібні.

З іншого боку, для GraphQL поки немає такої кількості інструментів і рішень по інтеграції, як для REST. Наприклад, не вийде за допомогою HTTP-кешування кешувати результати роботи GraphQL API так само легко, як це

робиться для результатів роботи REST API. Однак співтовариство наполегливо працює над покращенням інструментів та інфраструктури, а для кешування GraphQL можна використовувати такі інструменти, як Apollo Client і Relay. Зважаючи на все це для побудови сервісу було вибрано GraphQL.

2.2 Вибір мов програмування

Сервіс буде включати в себе бекенд та мобільний додаток. Для кожної частини програми потрібно обрати мову програмування.

Для побудови бекенду існує велика кількість мов програмування: Python, Ruby, PHP, JS , Elixir, Scala, Erlang, Clojure, Haskell, Goland, Rust, Java, C#, Kotlin. Провівши аналіз було вирішено реалізовувати мовою Kotlin. Kotlin (Котлін) — статично типізована мова програмування, що працює поверх JVM і розробляється компанією JetBrains. Основні особливості:

1) extension function;
2) high order function;
3) inter-operable with java;
4) null safety;
5) smart cast;
6) default and named arguments;
7) multi-value return from function;
8) data class;
9) open source.

Мобільний додаток був реалізований мовою JavaScript. З появою React Native, JavaScript став мультиплатформним і його можна буде використовувати не тільки на Android платформі, але й на iOS.

Основні особливості:

- math, number, string, array and object methods;
- binary and octal number types;
- default rest spread;
- for of comprehensions;

- symbols;
- tail calls;
- generators;
- map and set.

2.3 Вибір фреймворків для бекенда та мобільного додатку

Для мобільного додатку було вирішено використовувати React Native фреймворк так як, на даний час він є безперечним лідером по кількості використання.

React Native— відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників.

Основні особивості:

- declarative;
- virtual DOM;
- event handling;
- JSX;
- performance;
- component-based.

Для бекенда було вирішено використовувати Spring фреймворк так як, на даний час він є безперечним лідером по кількості використання.

Spring Framework — це програмний каркас (фреймворк) з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java.

Основні особливості Spring Framework можуть бути використані будь-яким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

Spring Framework складається з кількох модулів, які надають широкий спектр послуг:

1) Контейнер інверсії управління: конфігурація компонентів додатків і управління життєвим циклом об'єктів Java, здійснюється головним чином через інверсію управління

2) Аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні процедури

3) Доступ до даних: робота з реляційною системою управління базами даних на платформі Java з використанням JDBC і об'єктно-реляційні відображення та інструментів з NoSQL баз даних

4) Управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів

Модель-Вигляд-Управління (Model-View-Controller): програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful.

5) Автентифікація і авторизація: налаштовуванні процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою під проекту Spring Security (колишня система безпеки AcerI для Spring).

6) Віддалене керування: конфігураційний вплив і управління Java-об'єктами для місцевої (локальної) або віддаленої конфігурації через JMX

Тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів.

2.4 Спосіб автентифікації користувачів

Автентифікація на основі файлів cookie (Рисунок 2.1).

Автентифікація на основі файлів cookie була стандартним методом для перевірки автентичності користувачів протягом тривалого часу.

Автентифікація на основі файлів cookie є stateful. Це означає, що запис або сесія автентифікації повинні зберігатися як на сервері, так і на стороні клієнта.

Сервер повинен відстежувати активні сеанси в базі даних, тоді як на передньому кінці створюється файл cookie, який містить ідентифікатор сеансу. Потік традиційної автентифікації на основі файлів cookie:

1) Користувач вводить свої реєстраційні дані.
2) Сервер перевіряє правильність облікових даних і створює сеанс, який потім зберігається в базі даних.
3) У браузері користувачів розміщується файл cookie з ідентифікатором сеансу.
4) При наступних запитах ідентифікатор сеансу перевіряється відносно бази даних і, якщо дійсно, оброблений запит.
5) Після виходу користувача з програми сеанс знищується як на стороні клієнта, так і на стороні сервера.

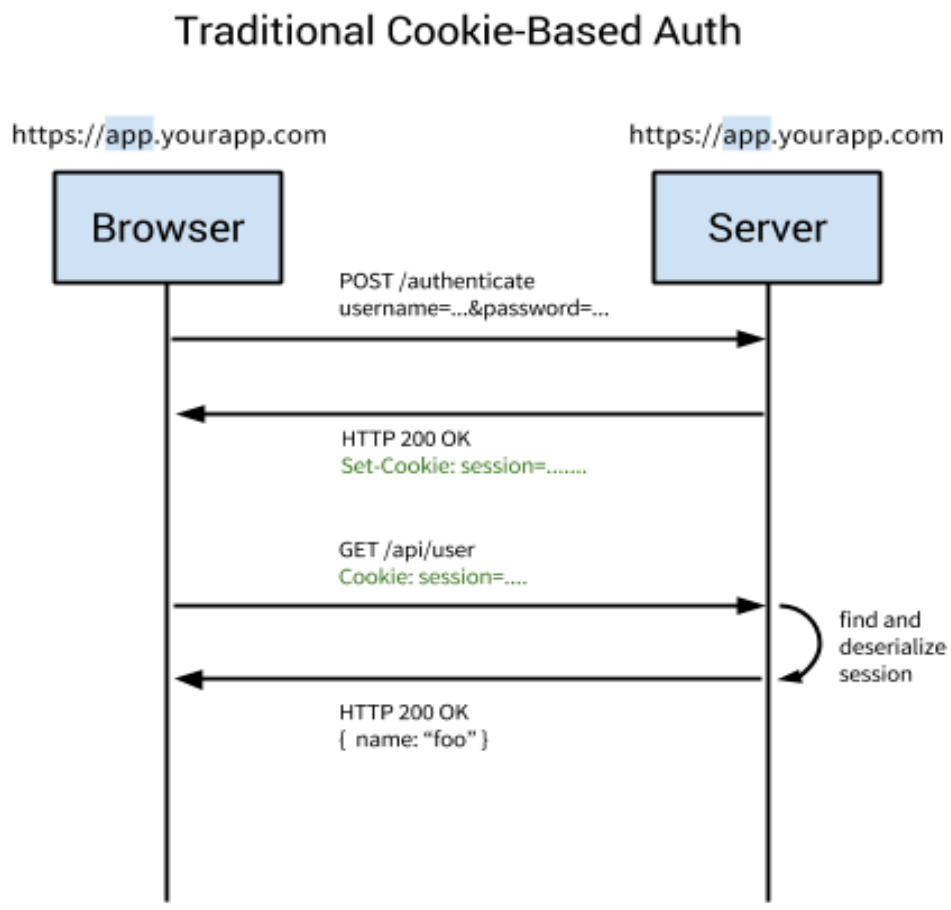

Рисунок 2.1 — Схематичне зображення аутентифікації на основі файлів cookie

Автентифікація на основі токенів (Рисунок 2.2).

Автентифікація на основі токенів за останні кілька років поширена завдяки зростанню кількості односторінкових додатків, веб-API і IoT. Коли ми говоримо про автентифікацію з токенами, ми зазвичай говоримо про аутентифікацію за допомогою JSON Web Tokens (JWT).Хоча існують різні способи реалізації токенів, JWT стали стандартом дефакто.

Автентифікація, що базується на токенах є stateless. Сервер не веде запису про те, які користувачі ввійшли в систему або яким було випущено JWT. Натомість кожен запит до сервера супроводжується токеном, який сервер використовує для перевірки автентичності запиту. Токен, як правило, надсилається як заголовок додаткової авторизації у вигляді Bearer {JWT} , але додатково може надсилатися в тілі запиту POST або навіть як параметр запиту. Як працює цей потік:

1) користувач вводить свої реєстраційні дані;
2) сервер перевіряє правильність облікових даних і повертає підписаний токен;
3) цей токен зберігається на стороні клієнта, найчастіше в локальному сховищі - але може зберігатися і в сховищі сеансу, або в файлі cookie;
4) подальші запити на сервер включають цей токен як додатковий заголовок авторизації або через один з інших методів, згаданих вище;
5) сервер декодує JWT і, якщо токен є допустимим, обробляє запит;
6) після того, як користувач вийшов з системи, токен знищено на стороні клієнта, взаємодія з сервером не потрібна.

При використанні cookie бекенд повинен виконувати пошук за традиційною SQL-базі або NoSQL-альтернативі, і обмін даними напевно триває довше, ніж розшифровка токена. Крім того, так як можна зберігати всередині JWT додаткові дані на зразок користувальницьких дозволів, то можна заощадити і додаткові звернення пошукові запити на отримання та обробку даних.

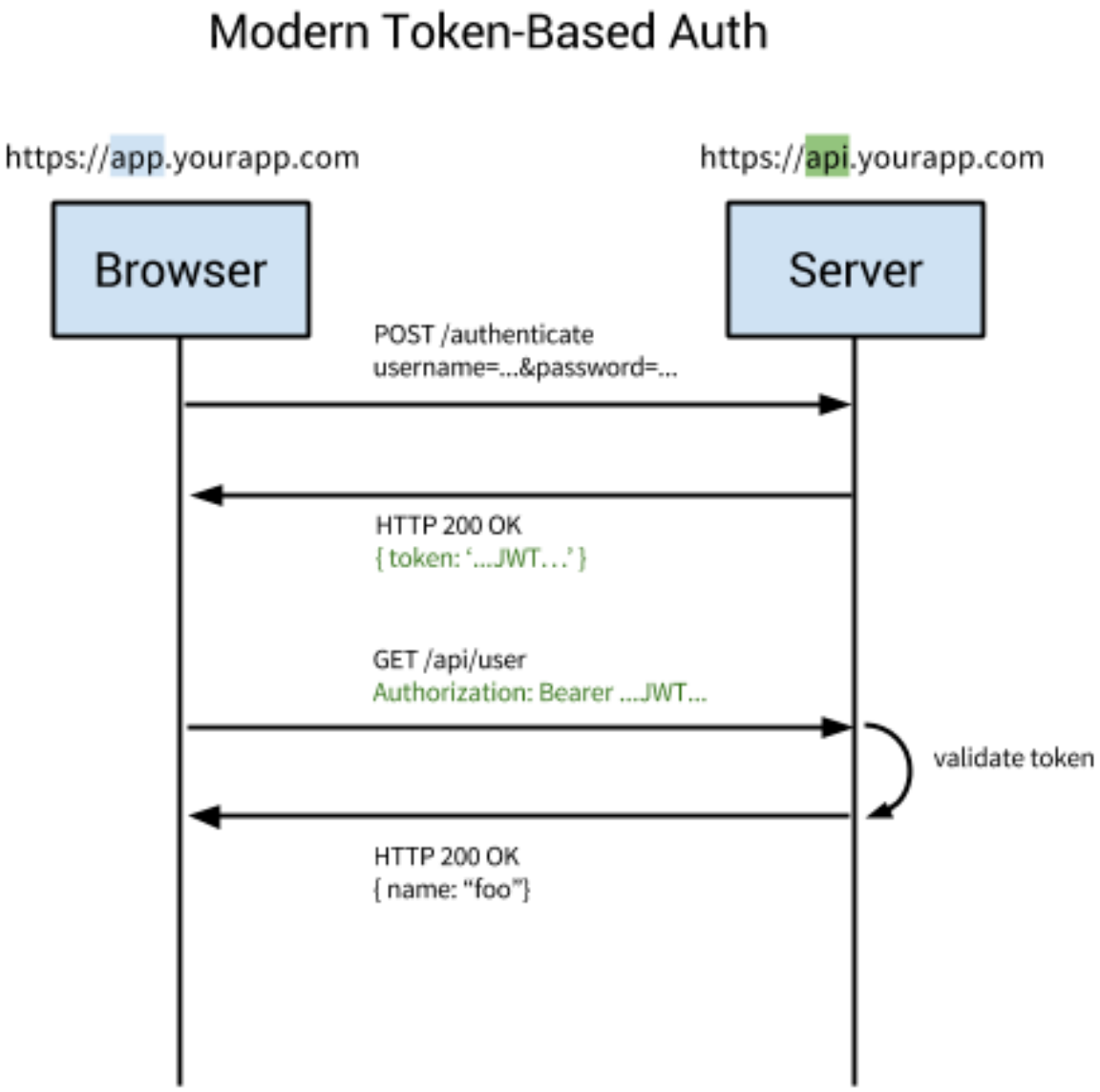

Рисунок 2.2 — Схематичне зображення аутентифікації на основі токенів

Переваги автентифікації на основі токенів.

Найбільшою перевагою використання токенів над cookies є той факт, що автентифікація токена є stateless. Бекенд не потребує ведення запису токенів. Кожен токен є автономним, містить всі дані, необхідні для перевірки його дійсності.

Тоді єдина робота сервера - це підписання маркерів на успішний запит на вхід і перевірка правильності вхідних маркерів. Насправді, сервер навіть не повинен підписувати маркери. Послуги третьої сторони, такі як Auth0, можуть

обробляти видачу лексем, і тоді серверу потрібно лише перевірити правильність токена.

Файли cookie добре працюють з окремими доменами та субдоменами, але коли справа стосується керування файлами cookie в різних доменах, це призводить до ряду проблем. На відміну від цього, підхід, заснований на токені з підтримкою CORS, робить тривіальним розкриття API для різних служб і доменів. Оскільки JWT необхідний і перевіряється з кожним викликом на зворотну сторону, доки існує допустимий маркер, запити можуть оброблятися.

Також сучасні API не тільки взаємодіють із браузером. Належним чином написаний один API може обслуговувати як браузер, так і різні мобільні платформи, такі як iOS і Android. Різні мобільні платформи та файли cookie не мають досконалої реалізації з вирішенням всіх проблем при взаємодії. Хоча це можливо, існує багато обмежень і міркувань щодо використання файлів cookie з мобільними платформами. З іншого боку, токени набагато простіше реалізувати як на iOS, так і на Android. Токени також легше реалізувати для програм і служб Internet of Things, які не мають концепції зберігання файлів cookie.

2.5 Вибір СУБД

Для побудови і підтримки бази даних було використано PostgreSQL СУБД. PostgreSQL — об'єктно-реляційна система керування базами даних (СКБД). Є альтернативою як комерційним СКБД (Oracle Database, Microsoft SQL Server, IBM DB2 та інші), так і СКБД з відкритим кодом (MySQL, Firebird, SQLite).

Порівняно з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється якоюсь однією компанією, її розробка можлива завдяки співпраці багатьох людей та компаній, які хочуть використовувати цю СКБД та впроваджувати у неї найновіші досягнення.

PostgreSQL не просто реляційна, а об'єктно-реляційна СУБД. Це дає йому деякі переваги над іншими SQL базами даних з відкритим вихідним кодом, такими як MySQL, MariaDB і Firebird.

Фундаментальна характеристика об'єктно-реляційної бази даних - це підтримка об'єктів і їх поведінки, включаючи типи даних, функції, операції, домени і індекси. Це робить PostgreSQL неймовірно гнучким і надійним. Серед іншого, він вміє створювати, зберігати та видавати складні структури даних. У деяких прикладах нижче ви побачите вкладені і складові конструкції, які не підтримуються стандартними РСУБД.

Існує великий список типів даних, які підтримує PostgeSQL. Крім числових, з плаваючою точкою, текстових, булевих і інших очікуваних типів даних (а також безлічі їх варіацій), PostgreSQL може похвалитися підтримкою uuid, грошового, що перераховується, геометричного, бінарного типів, мережевих адрес, бітових рядків, текстового пошуку, xml, json , масивів, композитних типів і діапазонів, а також деяких внутрішніх типів для ідентифікації об'єктів. Справедливості заради варто сказати, що MySQL, MariaDB і Firebird теж мають деякі з цих типів даних, але тільки PostgreSQL підтримує їх всі.

2.6 Вибір платіжної системи

Основні гравці ринку онлайн-платежів України.

1. Portmone

3 млн користувачів вже користуються послугами компанії. Тільки за першу частину 2018 року було проведено більше ніж 20 млн транзакцій. Найпопулярніші операції - оплата комунальних послуг, поповнення мобільного та переказ грошей з картки на картку.

У компанії є можливість оплачувати рахунки за банківськими реквізитами, є електронна доставка клієнтам рахунків від провайдерів послуг.

2. Plategka

Plategka була заснована в 2010 році. Девіз компанії - зручність, зрозумілість і безпека, її вже оцінили понад 20 тис користувачів, 300 компаній-партнерів і сотні тисяч клієнтів. Користувачі мають доступ до більше ніж 300 сервісів: переклади з карти на карту, оплата комунальних послуг, оплата

дитячих садків та мобільного зв'язку, покупка квитків на транспорт і бронь готелів і т.д.

3. iPay.ua

Що місяця понад 400 тис клієнтів користуються та сплачують послуги на сайті iPay.ua. За всі роки роботи (компанія була заснована в 2008 році) було проведено понад 25 млн транзакцій. За останні десять років компанія стала одним з лідерів ринку онлайн-платежів. iPay.ua мав за мету популяризувати оплату через інтернет серед населення України, а також забезпечити максимальну безпеку онлайн-розрахунків.

4. PayPong

Компанія Paypong з'явилася на ринку в 2014 році, і з цього моменту вона активно розвивається, удосконалює роботу свого сервісу і робить доступним онлайн кредитування на території країни. Відгуки людей про роботу мікро фінансової організації характеризують її, як стабільну, надійну і чесну фірму, яка може надати фінансову підтримку у важкий момент. Мінімальна сума позики становить сто гривень, максимальна - 3 тисячі 700 гривень. Термін кредитування - від одного до тридцяти днів.

5. Electrum Оператор електронних фінансових і платіжних інструментів. Рік заснування – 2016.

6. Tachcard

Компанія Tachcard була заснована в 2016 році (дочірня компанія City World Group LCC.). Головна спеціалізація – створення комплексних, інноваційних і високотехнологічних продуктів і рішень для корпоративного бізнесу. Існує мобільний додаток який дає змогу здійснювати грошові перекази з картки на картку, оплата найпопулярніших послуг (поповнення мобільного, оплата комунальних, страховки, погашення кредиту і т.д.).

7. Uplata

Своє існування компанія розпочала в 2016 році вона дає змогу оплати понад 1500 послуг по всій Україні. Що до особовості компанії, вона є не тільки

сервісом онлайн-платежів, але ще і платіжною картою. За ствердженням представників компанії, Uplata планує створити систему, яка зможе замінити в повсякденному житті класичний банк.

8. UAPAY

В 2016 Україні з'явився новий онлайн-сервіс миттєвих грошових переказів Uapay. Сервіс доступний без реєстрації і для будь-яких платформ: десктопів, планшетів, смартфонів. Переказати гроші в Uapay с карти на карту можна не тільки з зазначенням реквізитів одержувача, але також за номером телефону або email. Також в Uapay є цікава функція, яка дозволяє зайняти грошей по інтернету. Спектр послуг який дозволяє оплачувати компанія (більше 1600) по всій Україні. Крім того, UAPAY надає послуги еквайрингу інтернет-торговцям.

9. LeoBot

LeoBot, створений та запущений в 2017 році - продукт компаній LeoGaming і Mastercard. Це бот працює в соціальній мережі Facebook, на платформі безпечних платежів Masterpass. Всі операції виконуються після авторизації користувача в гаманці або після створення нового облікового запису. За допомогою чат-бота можна швидко переводити гроші з картки на картку, поповнити мобільний рахунок або акаунт онлайн-ігор, а також оплатити інтернет.

10. EasyPay

Один з лідерів серед сервісів онлайн-платежів в Україні, роботу почав в середині 2007 році. Сайт компанії дає можливість сплати більше 2500 послуг (інтернет, мобільний зв'язок, комунальні платежі, ТВ), а також перекази з карти на карту, за реквізитами та між електронними гаманцями. В минулому році EasyPay і Forward Bank презентували платіжну карту яка є спільною - кредитка, яка дозволяє здійснювати онлайн-розрахунки без комісії. Ліміт на цій карті - до 100 тис грн, а також користувачам доступна можливість отримувати кешбек на всі торгові операції.

Після порівняння за комісією систем онлайн платежів (Таблиця 2.1) було вирішено використовувати систему LeoBot, так як система не знімає жодної комісії при переведенні коштів.

Таблиця 2.1 – Порівняння комісій систем онлайн платежів

Сервіс	Комісія
Portmone	2 грн
Plategka	1,41 грн (для карт Райффайзен Банка Аваль); 2,23 грн (для карт Ощадбанка); 2,65 грн (інші банки)
iPay.ua	3,60 грн
PayPong	3 грн
Electrum	2,55 грн
Tachcard	3 грн
Uplata	3 грн (карта любого банка); 0,50 грн (Uplata)
UAPAY	3 грн
LeoBot	Без комісії
EasyPay	2 грн

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Продумуючи структуру як всієї системи, так і кожної з її компонент, було прийнято рішення створити систему з наступною структурою: Android app, Backend, PostgreSQL (Рисунок 3.1).

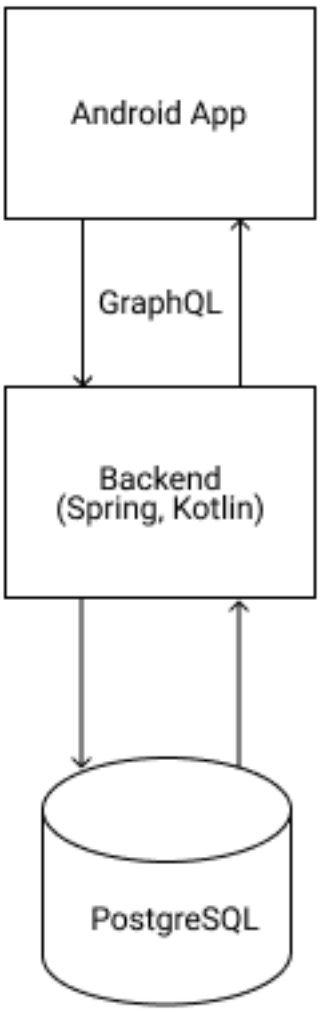

Рисунок 3.1 – Схема взаємодії компонент системи

Наступним кроком було побудовано ERD діаграму(Рисунок 3.2), котра включай в себе всі основні сутності сервісу, а також відношення між ними на рівні бази даних.

Основні сутності:

1) Shops
2) Products
3) Users
4) Purchases
5) Product categories
6) Users Groups
7) Departments
8) Shops Users

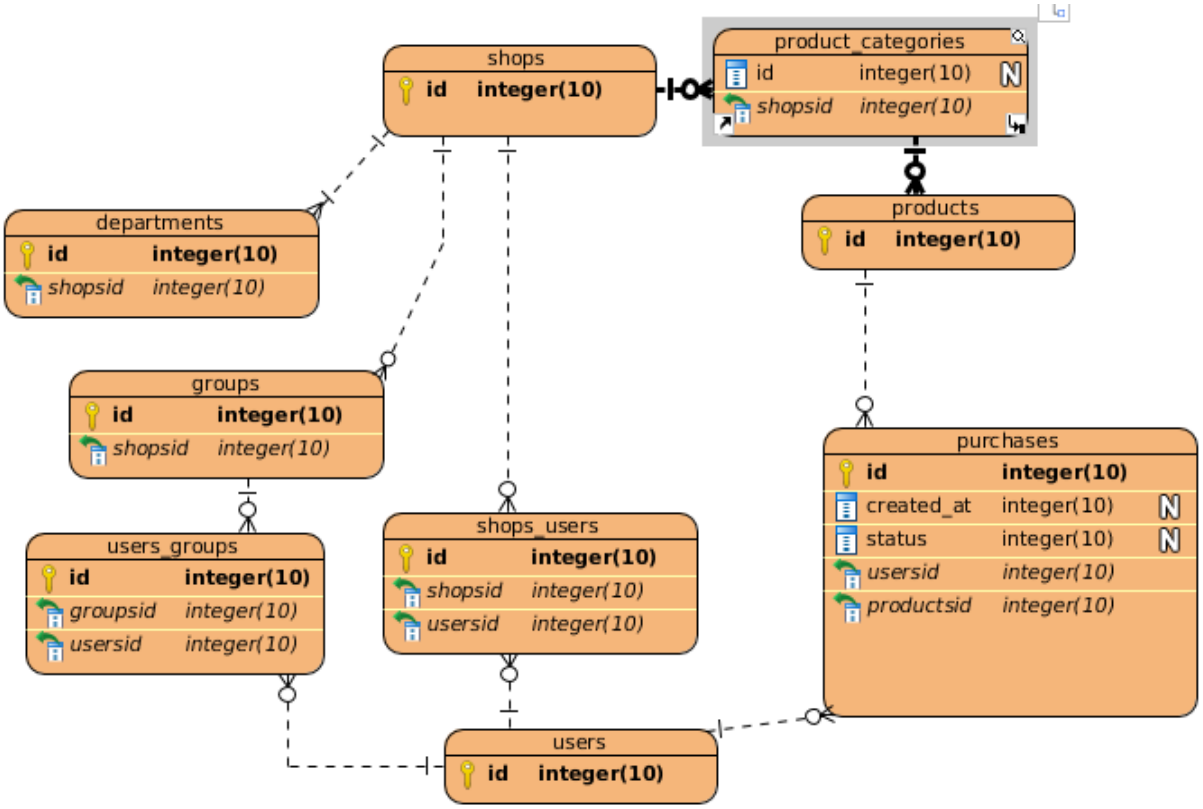

Рисунок 3.2 – ERD діаграма

3.2 Програмна реалізація

Для програмної реалізації бекенду спочатку було вирішено створити схему GraphQL котра описує взаємодію між бекендом та іншими сервісами.

```
schema {
  query: EntryPoint
  mutation: Mutations
}
```

```
type EntryPoint {
  hello: String @deprecated(reason: "just playing around!")
  cart(id: ID!): Cart
}

type Cart {
  id: ID!
  items: [Item!]!
  subTotal: Float!
}

type Item {
  quantity: Int!
  product: Product!
  total: BigDecimal!
}

type Product {
  id: ID!
  title: String!
  price: BigDecimal!
  description: String
  sku: String!
  images(limit: Int = 0): [String!]!
}

type Mutations {
  addProductToCart(cartId: ID!, productId: ID!, quantity: Int = 0): Cart
}
```

В даній схемі описані основні типи сутностей та дані котрі можна отримати з них. Детальний приклад запиту можна побачити на Рисунку 3.3, виконаний за допомогою GraphQL Playground

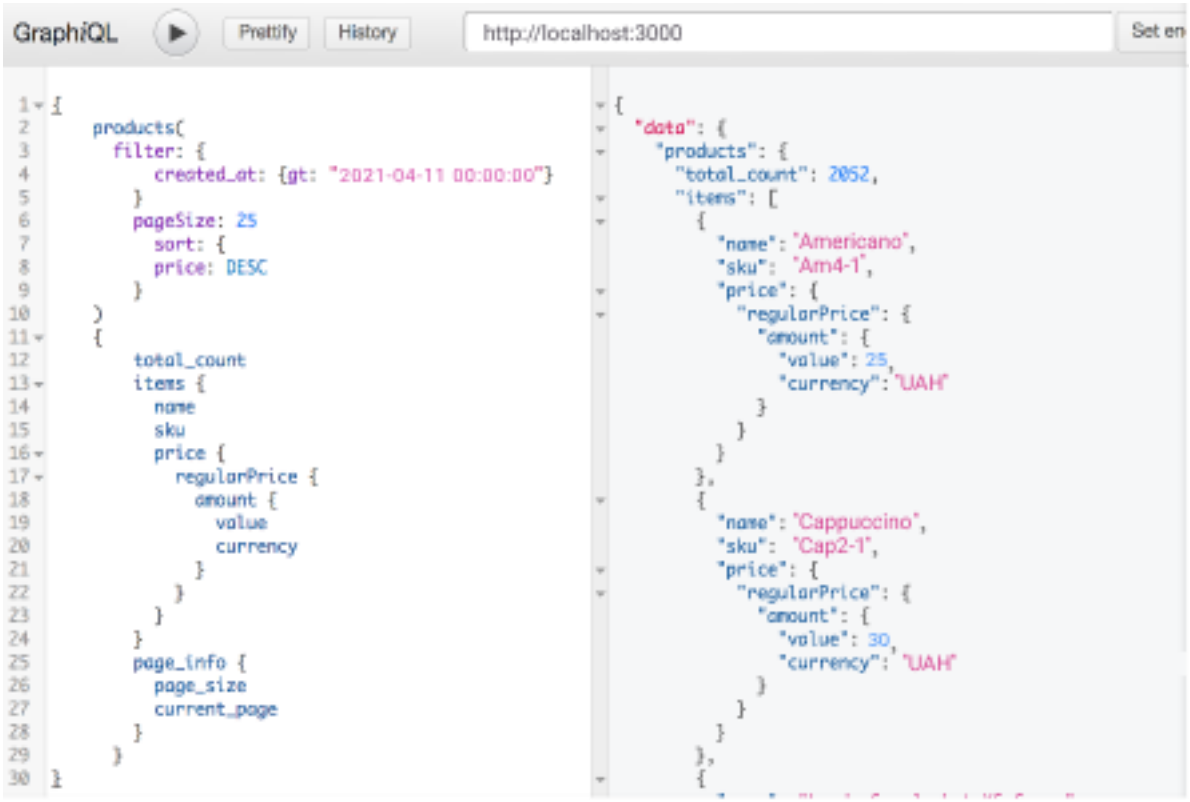

Рисунок 3.3 – Приклад запиту за допомогою GraphQL

Щоб дана схема загрузилася на сервері потрібно реалізувати відповідний парсер, котрий зчитає з файлу схему та виконає валідації на правильність типів. Для цього була написана наступна функція:

```
init {
   val schema = GraphQLSchemaGenerator()
            .withOperationsFromSingleton(cartGraph)
            .withOperationsFromSingleton(productQuery)
            .withValueMapperFactory(JacksonValueMapperFactory())
            .generate()
   graphQL = GraphQL.newGraphQL(schema)
            .queryExecutionStrategy(BatchedExecutionStrategy())
              .instrumentation(ChainedInstrumentation(listOf(
                    MaxQueryComplexityInstrumentation(200),
                    MaxQueryDepthInstrumentation(20)
            )))
            .build()
}
```

Для того щоб даний запит виконувався на сервері потрібно реалізувати так звані Resolvers, котрі відповідають за відображення кожного поля, котре в казано в схемі.

Приклад реалізації Resolver для відображення продукта:

```
@Bean
fun productResolver() = object : GraphQLResolver<Product> {
fun images(product: Product, limit: Int) =
          product.images.take(
                  if (limit > 0) limit else product.images.size)
 }
```

Як уже було сказано вище, для побудови фроненда було використано React. Основною функціональною одиницею в React є компонента. Концептуально, компоненти схожі на JavaScript-функції. Вони приймають довільні дані (звані props) і повертають React-елементи, які описують те, що повинно з'явитися на екрані. Реалізація компоненти для відображення інформації про користувача:

```
export default compose(
  connect((_, props) => ({
    initialValues: {
      name: props.user.get('name'),
      instagram_url: props.user.get('instagram_url'),
      fb_url: props.user.get('fb_url'),
    },
})
```

3.3 Обробка помилок від сервера з системою типів GraphQL

GraphQL сам по собі не пропонує ні яких механізмів для обробки даних. На офіційному сайті пропонується додати окреме поле errors з типом MutationError котрий включай в себе path та message, в котрому передаються дані про помилку. Для прикладу, мутація для додавання нової одиниці товару буде виглядати так:

```
mutation($itemId: ID!, $itemAttributes: ItemAttributes!) {
  updatePost(id: $itemId, attributes: $itemAttributes) {
    item {
      title
      price
      description
    }
```

```
    errors {
      path
      message
    }
  }
}
```

Такий варіант підходить для простих помилок. Проблеми котрі виникають при такому підході:

1. Усі помилки трактуються однаково. Помилки потрапляють до масиву помилок, незалежно від того, про яку помилку вони говорять.

2. Важко зрозуміти, звідки виникла помилка. Нашим прикладом був простий запит, але в більш складних запитах важче дізнатися, звідки помилка, особливо якщо це, скажімо, список елементів.

3. Клієнту важко знати, про які помилки потрібно піклуватися. Клієнти отримують усі помилки в масиві помилок, тому клієнти навіть не можуть запитувати помилки. Це означає, що клієнти не знають, які справи навіть мають бути розглянуті, не кажучи вже про те, які з них важливі, які ми можемо ігнорувати тощо.

Далі було проведено дослідження, в результаті котрих було вирішено всі помилки від сервера розділити на дві групи:

– валідаційні - помилки викликані неможливістю обробити передані призначені для користувача дані через їх не коректності;

– системні - помилки викликані порушеною роботою backend.

При будь-якому зверненні до сервера і відсутності системної помилки, клієнту повинен повернутися результат операції, який може бути успішним або неуспішним. Клієнт в свою чергу запитує поля як успішної відповіді так і не успішного, а визначає тип результату по системному полю __typename. Реалізація типу результату операції в API повинна бути зроблена з урахуванням взаємовиключення появи і даних і помилки у відповіді.

Варіації відповідей на прикладі мутації відкриття рахунку:

```
mutation OpenAccount {
   openAccount(input:{
         platform: ...,
```

```
            groupSetId:"...",
            groupCurrencyId:"...",
            leverage: ...,
            password:"..."
        }) {
            result {
                __typename
                ... on AccountOpenSuccess {
                    account {
                        id
                        ...
                    }
                }
                ... on AccountOpenFail {
                    errors {
                        message
                        path
                    }
                }
            }
        }
    }
```

Відповідь без помилок:

```
{
    "data": {
        "result": {
            "__typename": "AccountOpenSuccess",
            "account": {
                "id": "",
                ...
            }
        }
    }
}
```

Відповідь с помилкою валідації поля:

```
{
    "data": {
        "result": {
            "__typename": "AccountOpenFail",
            "errors": [
                {
                   "message": "...",
                   "path": "groupCurrencyId"
                }
            ]
        }
    }
```

```
}
```

Відповідь с помилкою валідації поля:

```
{
    "errors": [
        {
           "message": "...",
           "code": "..."
        }
    ]
}
```

Для того щоб реалізувати дану схему на сервері було використано union - це абстрактні типи GraphQL, які дозволяють полю схеми повертати один із декількох типів об'єктів. Завдяки цьому в поле результату є можливість відправляти, як дані після успішного виконання, так і дані про помилку, при цьому вказавши правильний тип. Для прикладу, опис типу для регістрації користувача:

```
union UserRegisterResult =
  UserRegisterResultSuccess
  | UserRegisterInvalidInputError
  | CountryBlockedError

type Mutation {
  userRegister(input: UserRegisterInput!): UserRegisterResult!
}
```

3.4 Користувацький інтерфейс

Основною частиною для користувача будь-якого мобільного додатку є інтерфейс, адже саме він формує враження про використання програми. Спочатку, перед тим як була розпочата програмна реалізація, було створено прототип, це дало змогу більш ефективно використати ресурси і з самого початку відкинути хибні ідеї, котрі погано відображалися на користувацькому досвіді використання додатку.

Одну з найважливіших ролей при проектуванні інтерфейсу відігравав UX(дизайн взаємодії з користувачем). В цілому, дизайн повинен бути сучасним, лаконічним без використання застарілих засобів взаємодії з користувачем.

На Рисунку 3.4 зображений перший екран котрий бачить користувач, його ще називають домашній екран.

Рисунок 3.4 – Домашній екран

Далі на Рисунку 3.5 показано екран, котрий просить користувача про дозвіл користування даними про локацію або ж вести в ручну дані(Рисунок 3.6).

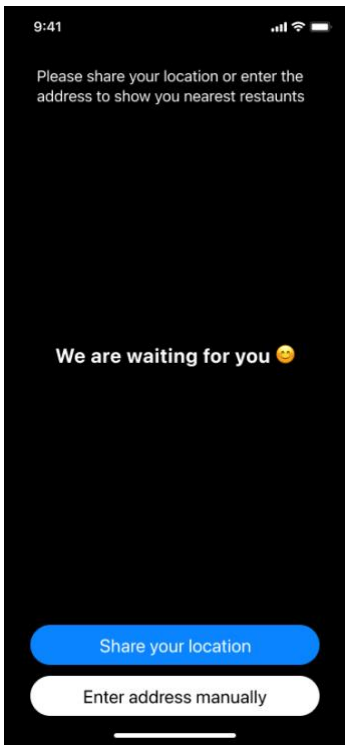

Рисунок 3.5 – Запит даних про локацію

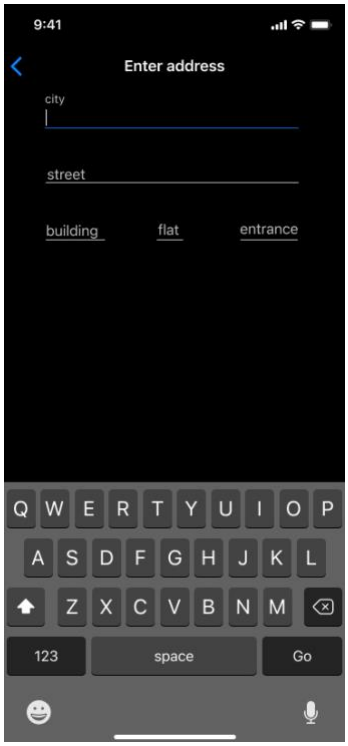

Рисунок 3.6 – Введення локації вручну

Після того, як користувач ввів локацію, додаток просить ввести країну і номер телефону для регістрації(Рисунок 3.2.7), а потім просить підтвердити його(Рисунок 3.2.8).

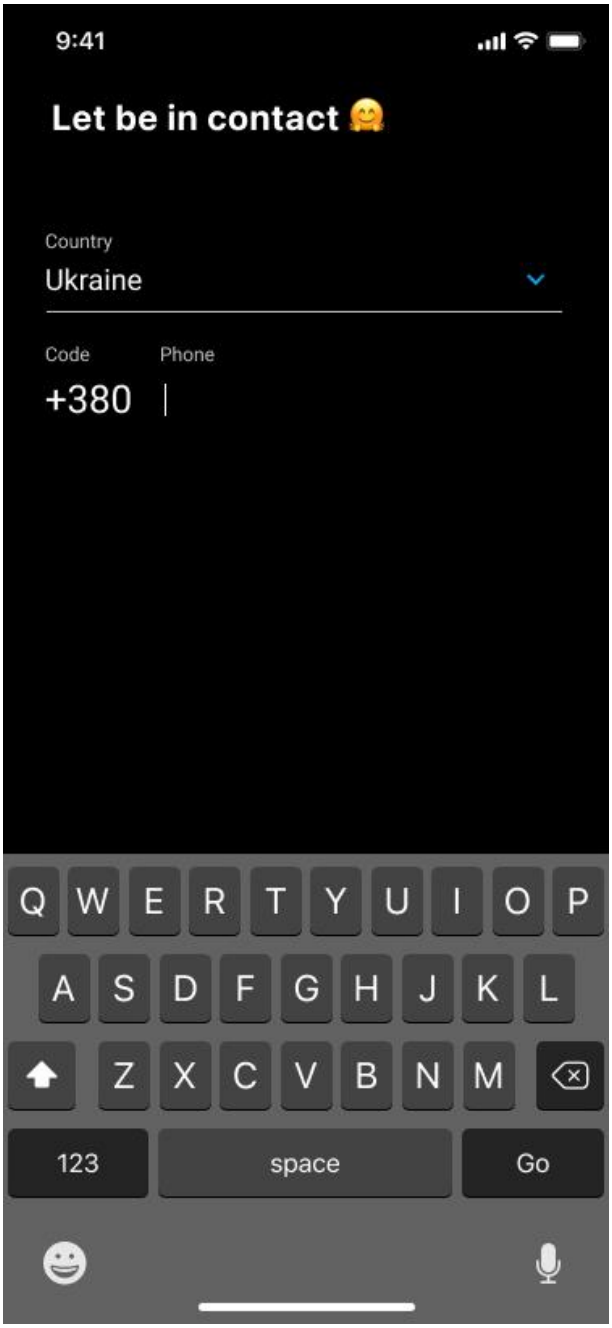

Рисунок 3.7 – Регістрація

Рисунок 3.8 – Підтвердження номера телефону

Після успішної регістрації, відкривається основний екран(Рисунок 3.9) на якому зображено основні заклади легкого харчування з рейтингом, відстанню і іншою додатковою інформацією. Є основний функціонал для навігації по додатку: пошук за допомогою текстового воду(Рисунок 3.10-11), пошук на карті(Рисунок 3.11) і основний вибір ресторанів на головному екрані.

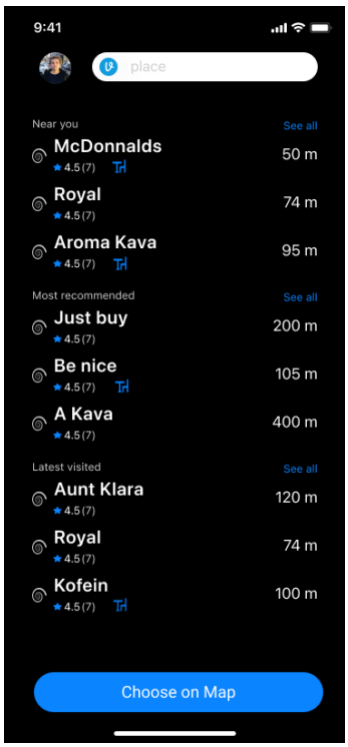

Рисунок 3.9 – Основний екран

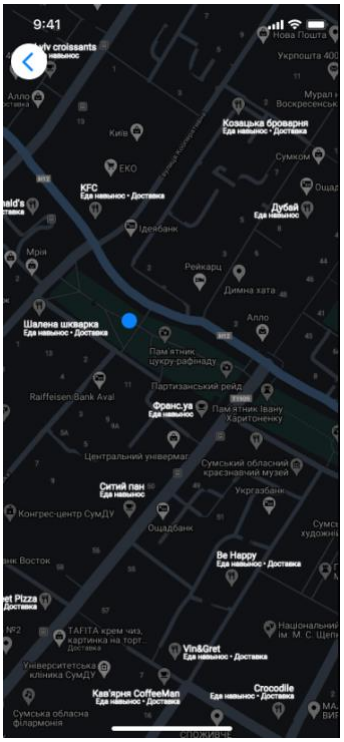

Рисунок 3.10 – Пошук по карті

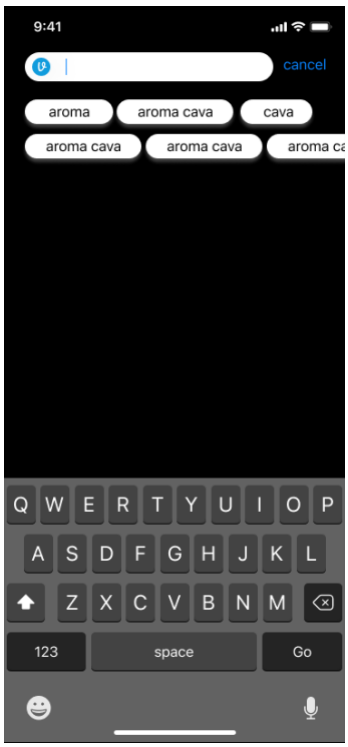

Рисунок 3.11 – Пошук за допомогою текстового воду

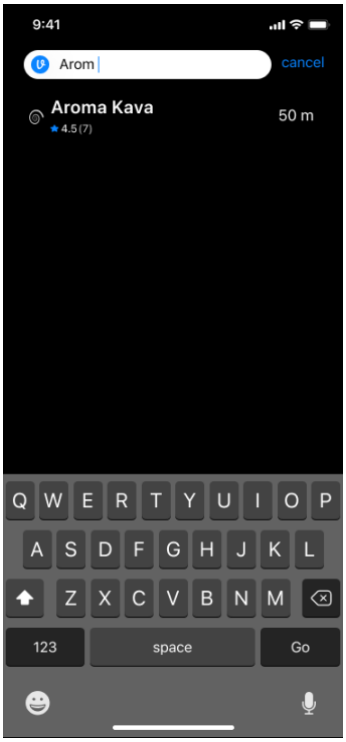

Рисунок 3.12 – Пошук після введення даних

Для прикладу була вибрана Aroma Kava і показана сторінка з основними товарами(Рисунок 3.2.13) і сторінка після вибору першого товару(Рисунок 3.2.14).

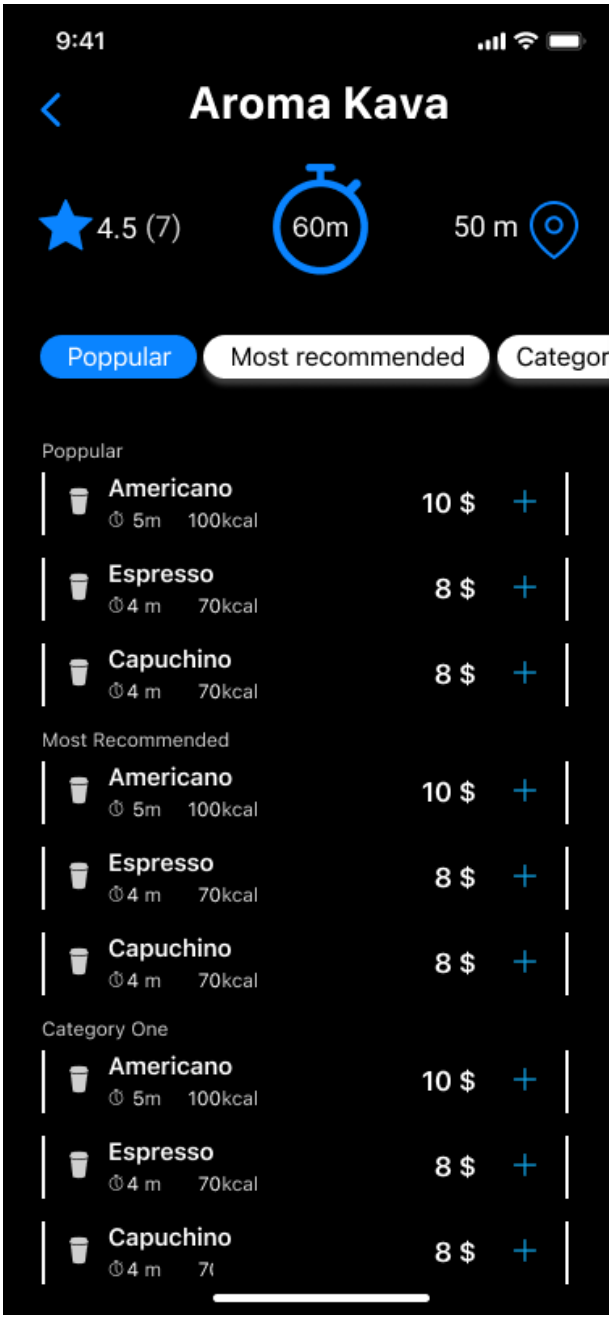

Рисунок 3.13 – Сторінка закладу

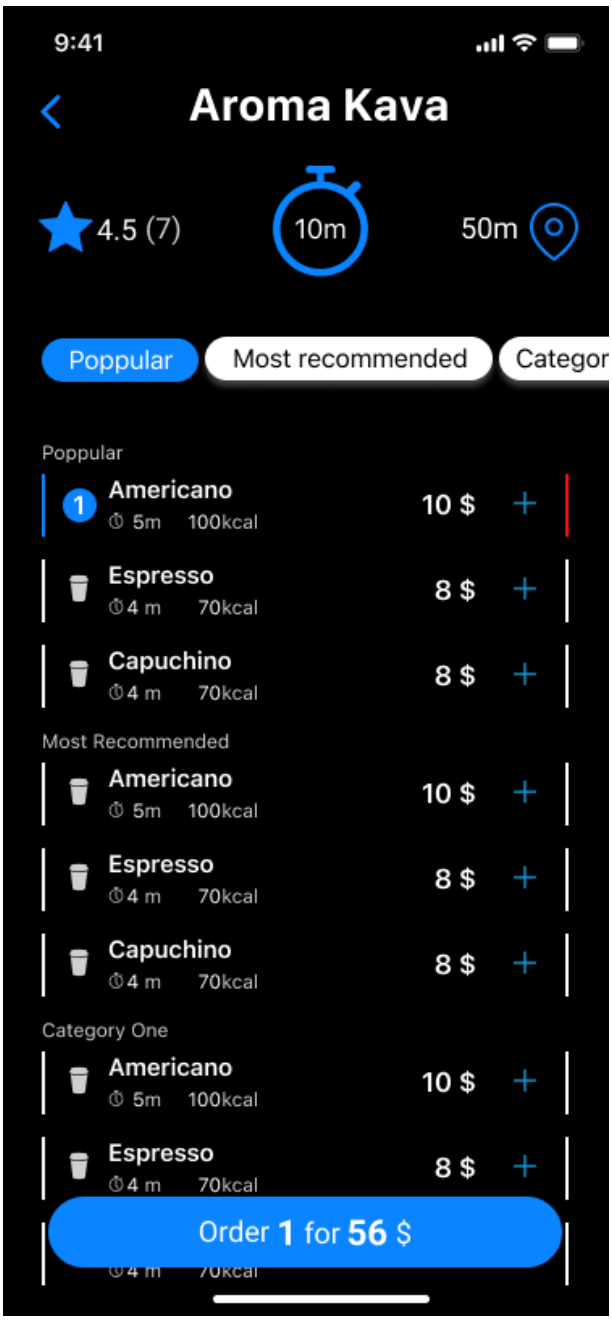

Рисунок 3.14 – Вибір товару

Після вибору товару можна зразу перейти до оформлення замовлення(Рисунок 3.15). На цій сторінці можна перейти на сторінку вибору методу оплати(Рисунок 3.16) і на ній можна додати новий метод оплати(Рисунок 3.17).

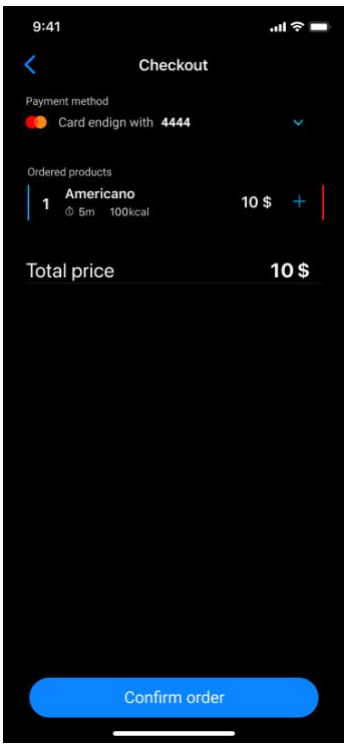

Рисунок 3.15 – Оформлення замовлення

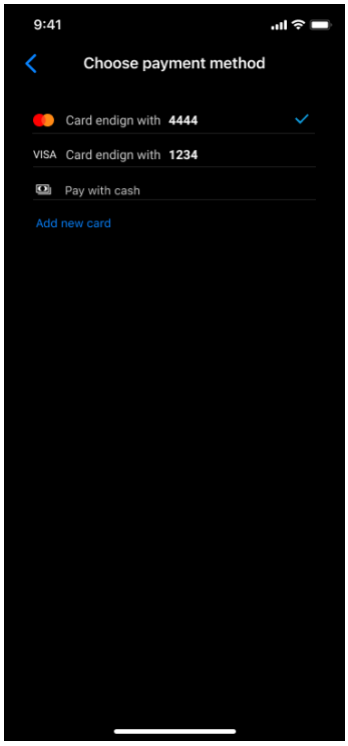

Рисунок 3.16 – Вибір методу оплати

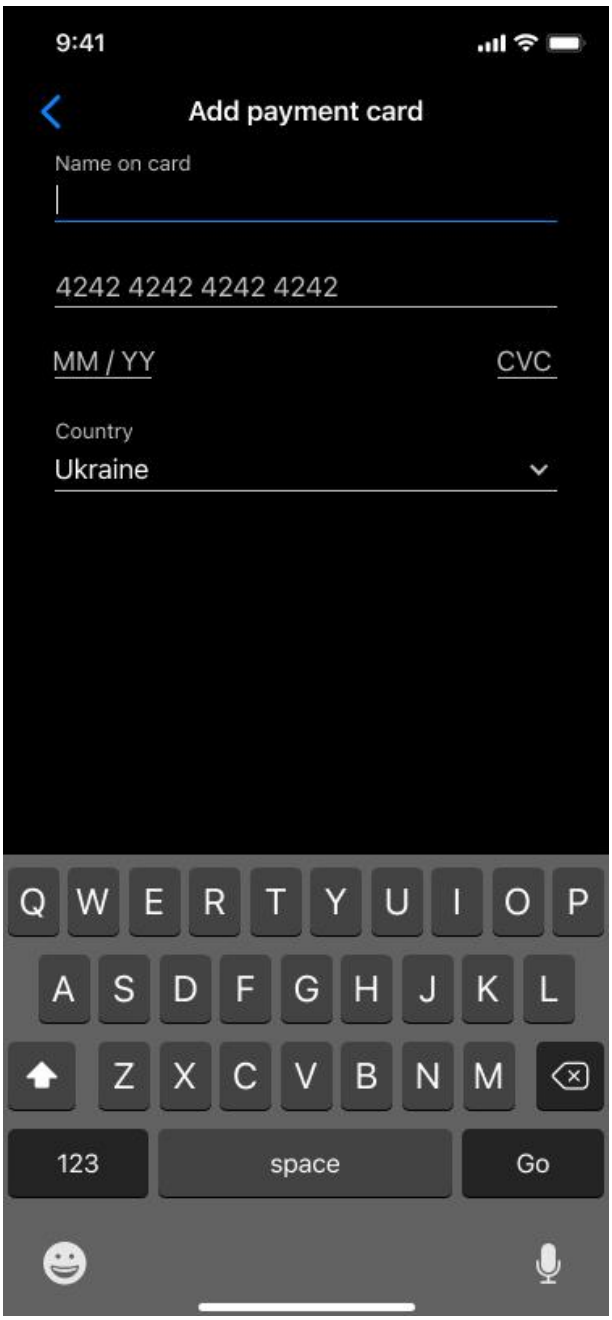

Рисунок 3.17 – Додавання нового методу оплати

На Рисунку 3.18 зображена сторінка відстеження замовлення. На ній можна: перейти на карту де саме знаходиться заклад(Рисунок 3.2.9), поділитися замовленням с кимось з контактів(Рисунок 3.19), відмінити замовлення(Рисунок 3.20), зв’язатися з службою підтримки через месенджери.

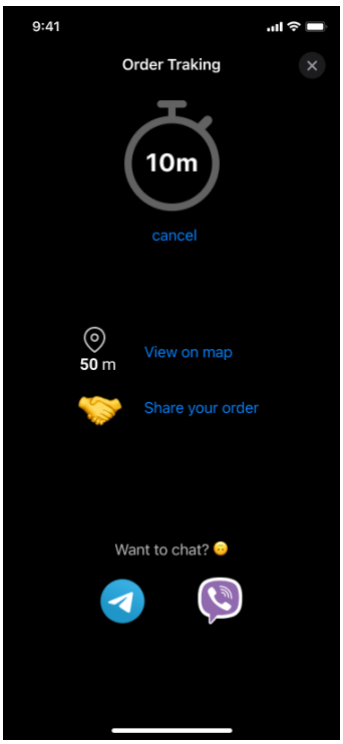

Рисунок 3.18 – Відстеження замовлення

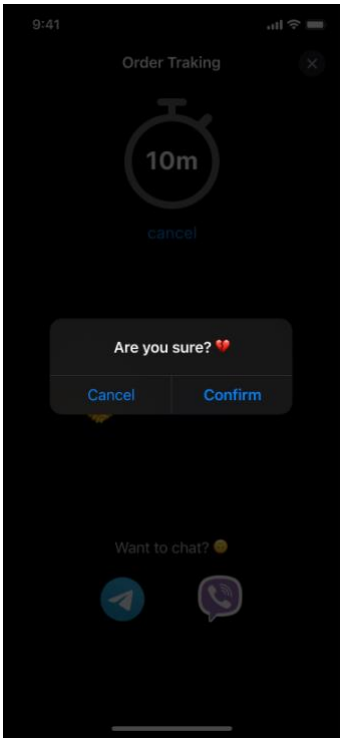

Рисунок 3.19 – Відміна замовлення

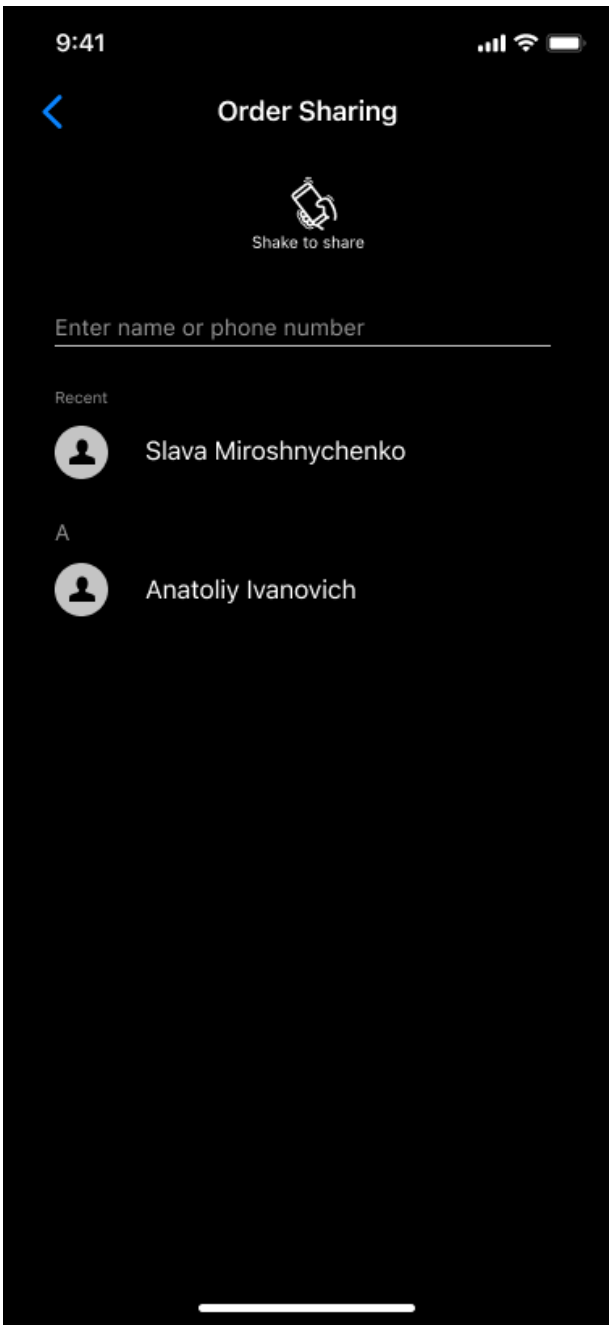

Рисунок 3.20 – Пошук контактів для відправлення замовлення

Після того як замовлення було підготовлення закладом, сторінка відстеження змінює свій вид(Рисунок 3.21). Замість відміни товару, тепер з’явилась можливість показати QR код з замовленням(Рисунок 3.22)

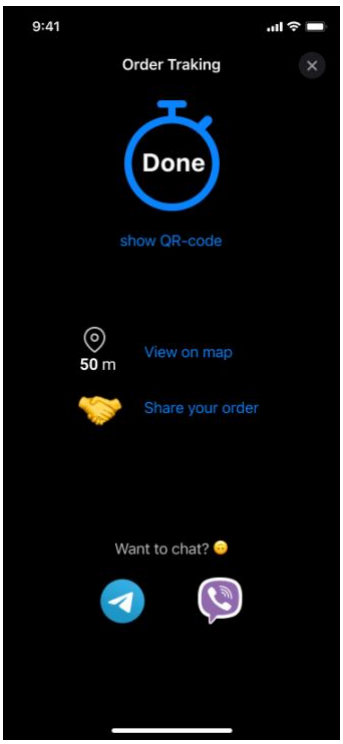

Рисунок 3.21 – Відстеження замовлення після його виконання

Рисунок 3.22 – QR код замовлення

Коли буде натиснута кнопка, що замовлення було отримане, відкриється сторінка з можливість оцінити заклад і залишити коментар (Рисунок 3.23).

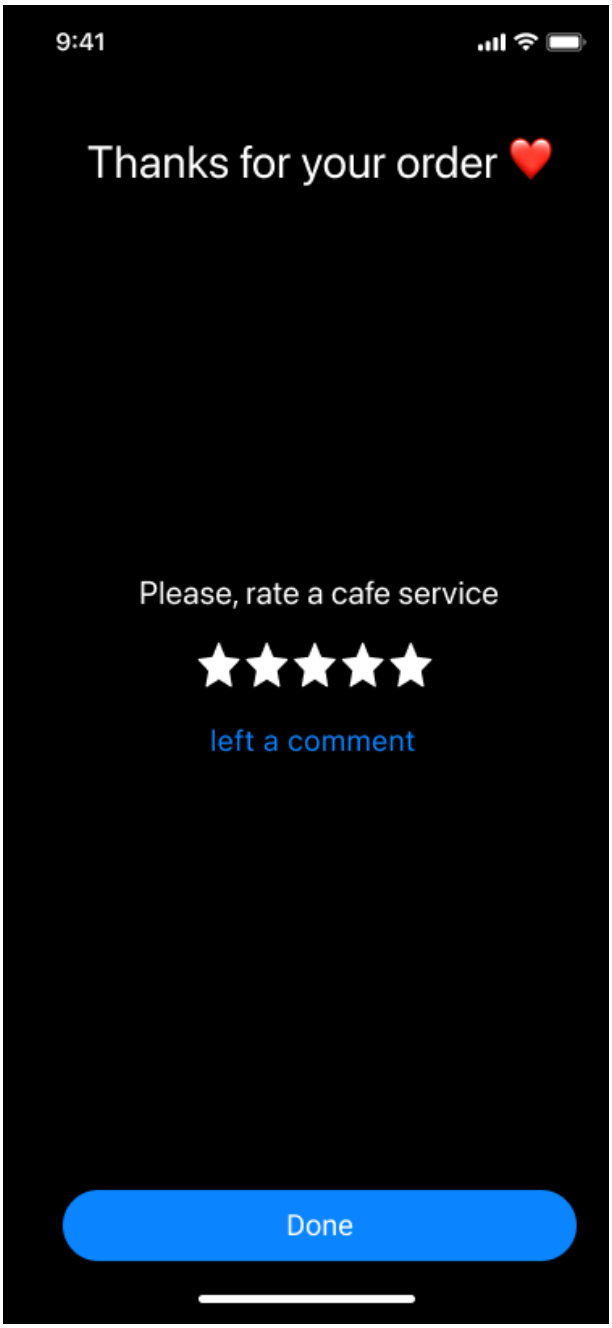

Рисунок 3.23 – Сторінка оцінки закладу

Також на основному екрані можна перейти на налаштування користувача(Рисунок 3.24), а з цієї сторінки на сам профіль користувача (Рисунок 3.25)

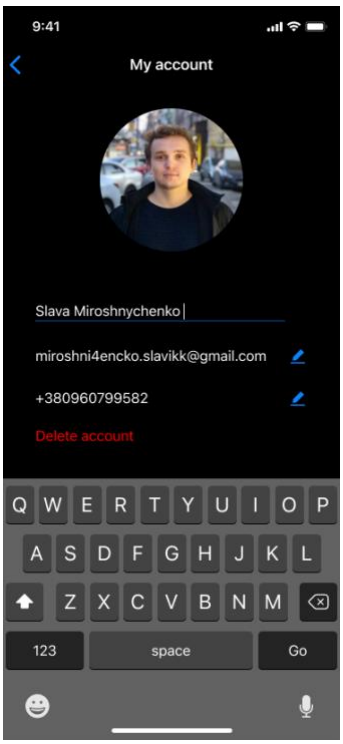

Рисунок 3.24 – Профіль користувача

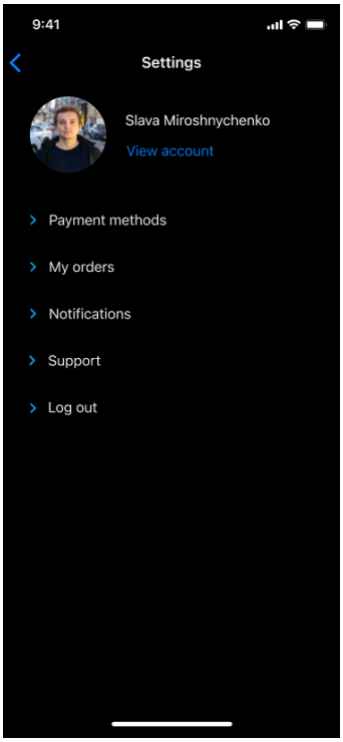

Рисунок 3.25 – Налаштування користувача

В результаті було отримано швидкий, продуманий до деталей інерфейс, який зручно використовувати для замовлення в закладах швидкого харчування. Максимальна увага була приділена простоті та зрозумілості у навігації та пошуку закладів для користувача, а також відображення точного часу приготування замовлення. Системе рейтингу та коментарі викликають довіру у користувача до додатку, як наслідок є постійне користування.

ВИСНОВКИ

За результатами аналізу наявних програмних продуктів, що надають можливість замовити товари в закладах швидкого харчування, було виявлено ряд проблем у їх функціональності та комфорті роботи з ними.

Було вирішено створити власну програму, яка в автоматизованому режимі дозволяли закладам швидкого харчування обслуговувати покупців.

У результаті було продумано методи та технології, які були реалізовані для ефективної роботи програмного продукту. компоненти якого взаємодіють між собою через GraphQL. Backend реалізовано за допомогою Spring Framework, а мобільну версію на базі Android та за допомогою React.

Також, проаналізовано та вибрано авторизацію та аутентифікацію для користувачів в виді токенів. Інформацію про товари було вирішено зберігати у json форматі так, як характеристики можуть бути різні і в табличному виді зберігати їх не доцільно. Також, було проведено аналіз великої кількості платіжних систем і обрана найкраща.

Під розробки системи керування були використанні різноманітні технології та знання, зокрема

- основи комп’ютерних мереж, телекомунікаційних систем;
- мова Kotlin;
- мова Java та мережеві технології Java;
- команди Unix-подібних систем;
- веб-технології: HTML-розмітка, JSON-формат, regexp;
- підтримка програмного забезпечення: використання системи контролю версій, системи автоматичної збірки, уникнення антипатернів.

Була вирішена проблема з коректною обробкою помилок від сервера за допомогою системи типів GraphQL.

Розроблений додаток задовольняє всім вимогам, поставленим на етапі постановки завдання.

СПИСОК ЛІТЕРАТУРИ

1. Мірошниченко В.І. Інформаційна система онлайн замовлень побутових товарів [Текст]: робота на здобуття кваліфікаційного рівня бакалавр; спец.: 122 - комп`ютерні науки В.І. Мірошниченко; кер. Б.О. Кузіков. - Суми: СумДУ, 2019. - 59 с.

2. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс] – Режим доступу: https://www.postgresql.org/

3. GraphQL Foundation [Електронний ресурс] – Режим доступу: https://foundation.graphql.org/

4. PostgreSQL [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/PostgreSQL

5. GraphQL Playground Visually explore Apollo Server [Електронний ресурс] – Режим доступу: https://www.apollographql.com/docs/apollo-server/testing/graphql-playground/

6. Bast H, Weber I. Повна пошукова система: інтерактивна, ефективна та інтеграційна IP та БД. Третя дворічна конференція з інноваційних систем даних. Сан Франциско VLDB Journal. 2007. 88–95.

7. Б. Маклафлин. Изучаем Ajax. — СПб.: Питер, 2007. — ISBN 978-5-91180-322-3.

8. Стивен Хольцнер. Ajax Библия программиста. — М.: Диалектика, 2009. — С. 553. — ISBN 978-5-8459-1502-3.

9. Дейв Крейн, Бер Бибо, Джордон Сонневельд. Ajax на практике. — М.: Вильямс, 2007. — ISBN 978-5-8459-1327-2.

10. Дэниел Вулстон. Ajax и платформа .NET 2.0 для профессионалов. — М.: Вильямс, 2007. — С. 464. — ISBN 1-59059-670-6.

11. Дейв Крейн, Эрик Паскарелло, Даррен Джеймс. AJAX в действии: технология — Asynchronous JavaScript and XML. —М.: Вильямс, 2006. — С. 640. — ISBN 1-932394-61-3.

12. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И.
Основы web-технологий Интернет-университет информационных технологий - ИНТУИТ.ру, 2003

13. Будилов В.А. Практические занятия по HTML. Краткий курс
Наука и техника, 2001

14. Крамер Э. HTML: наглядный курс web-дизайна
Диалектика, 2001

15. JavaScript: The Definitive Guide S.Spainhour, R.Eckstein
Webmaster in a Nutshell. 2nd Edition O'Reilly, 1999

16. Паттерсон Л. Использование HTML 4.0. Ясно, кратко, надежно
Диалектика, 1999

17. Муссиано Ч., Кеннеди Б. HTML и XHTML. Подробное руководство
Символ-Плюс, 2002

18. Пэтчетт К., Райт М. CGI/Perl: создание программ для Web
BHV-Киев, 2000

19. Коржинский С.Н. Настольная книга Web-мастера: эффективное применение HTML, CSS и JavaScript КноРус, 2000

20. David Flanagan. JavaScript: The Definitive Guide 3rd Edition. O'Reilly & Associates, Inc. ISBN 1-56592-392-8

21. Ташков П.А.: Веб-мастеринг: HTML, CSS, JavaScript, PHP, CMS, графика, раскрутка. - СПб.: Питер, 2020

ДОДАТКИ

Додаток А. Лістинг програмного коду

```
@Configuration
class RepositoriesConfiguration {

    @Bean
    fun commandLineRunner(cartRepository: CartRepository,
                          productServiceRestClient: ProductServiceRestClient) =
            CommandLineRunner { insertInitialData(cartRepository,
productServiceRestClient) }

    private fun insertInitialData(cartRepository: CartRepository,
                                  productServiceRestClient: ProductServiceRestClient) {
        val product1 =
productServiceRestClient.fetchProduct("59eb83c0040fa80b29938e3f")
        val product2 =
productServiceRestClient.fetchProduct("59eb83c0040fa80b29938e40")
        val product3 =
productServiceRestClient.fetchProduct("59eb88bd040fa8125aa9c400")

        val cart = Cart()
        cart.addProduct(product1.id, product1.price, 1)
        cart.addProduct(product2.id, product2.price, 2)
        cart.addProduct(product3.id, product3.price, 1)
        cartRepository.save(cart)
    }
}
@Configuration
class RestClientsConfiguration {
```

```
        @Bean
        fun productServiceRestClient() =
ProductServiceRestClient("http://localhost:9090")
    }

    class ProductServiceRestClient(private val baseUrl: String) {

        fun fetchProduct(productId: String): Product =
"$baseUrl/products/$productId".httpGet()
                .responseObject(jacksonDeserializerOf<Product>())
                .third.get()

    }


    @Entity
    data class Cart(

            @Id
            @GeneratedValue
            var id: Long? = null,

            @ElementCollection(fetch = FetchType.EAGER)
            val items: MutableList<Item> = mutableListOf()) {

        fun getSubTotal(): BigDecimal = items
                .map { it.total }
                .reduce { obj, incSum -> obj.add(incSum) }
```

```
        fun addProduct(id: String, price: BigDecimal, quantity: Int) {
            val item = items.stream()
                    .filter { p -> p.productId == id }
                    .findFirst()
                    .orElseGet {
                        val newItem = Item(id, 0, BigDecimal.ZERO)
                        items.add(newItem)
                        newItem
                    }
            item.quantity = item.quantity + quantity
            item.total = price.multiply(BigDecimal.valueOf(item.quantity.toLong()))
        }

    }

    @Component
    class CartService(private val cartRepository: CartRepository, private val
productServiceRestClient: ProductServiceRestClient) {

        fun findCart(cartId: Long): Cart? =
cartRepository.findById(cartId).orElse(null)

        fun addProductToCart(cartId: Long, productId: String, quantity: Int): Cart {
            val cart = findCart(cartId)!!
            val product = productServiceRestClient.fetchProduct(productId)
            cart.addProduct(product.id, product.price, quantity)
            return cartRepository.save(cart)
        }
```

```
}

@Embeddable
data class Item(

    @Column(nullable = false)
    var productId: String = "",

    @Column(nullable = false)
    var quantity: Int = 0,

    @Column(nullable = false)
    var total: BigDecimal = BigDecimal.ZERO) {

    internal constructor(productId: String, productPrice: BigDecimal, quantity:
Int) : this() {
        this.productId = productId
        this.quantity = quantity
        this.total =
BigDecimal.valueOf(quantity.toLong()).multiply(productPrice)
    }

}


@Component
class CartGraph(private val cartService: CartService) {
```

```
        @GraphQLQuery(name = "cart")
        fun cart(@GraphQLArgument(name = "id") id: Long): Cart? {
            log.info("fetching cart with id={}", id)
            return cartService.findCart(id)
        }

        @GraphQLMutation(name = "addProductToCart")
        fun addProductToCart(@GraphQLArgument(name = "cartId") cartId: Long,
                             @GraphQLArgument(name = "productId") productId: String,
                             @GraphQLArgument(name = "quantity", defaultValue = "1")
quantity: Int): Cart {
            log.info("adding {} product(s) with id={} to cart with id={}", quantity,
productId, cartId)
            return cartService.addProductToCart(cartId, productId, quantity)
        }

        companion object {
            var log: Logger = getLogger(CartGraph::class.java)
        }

    }

    @RestController
    class GraphQLController(cartGraph: CartGraph, productQuery: ProductGraph)
{

        private val graphQL: GraphQL

        init {
```

```
            val schema = GraphQLSchemaGenerator()
                    .withOperationsFromSingleton(cartGraph)
                    .withOperationsFromSingleton(productQuery)
                    .withValueMapperFactory(JacksonValueMapperFactory())
                    .generate()
            graphQL = GraphQL.newGraphQL(schema)
                    .queryExecutionStrategy(BatchedExecutionStrategy())
                    .instrumentation(ChainedInstrumentation(listOf(
                            MaxQueryComplexityInstrumentation(200),
                            MaxQueryDepthInstrumentation(20)
                    )))
                    .build()
        }

        @PostMapping(value = ["/graphql"], consumes =
[APPLICATION_JSON_UTF8_VALUE], produces =
[APPLICATION_JSON_UTF8_VALUE])
        @ResponseBody
        fun execute(@RequestBody request: Map<String, Any>): ExecutionResult =
                graphQL.execute(ExecutionInput.newExecutionInput()
                        .query(request["query"] as String)
                        .operationName((request["operationName"] ?: "") as String)
                        .build())

    }

    @Component
    class ProductGraph {
```

```
    @GraphQLQuery(name = "product")
    @Batched
    fun products(@GraphQLContext items: List<Item>,
                 @GraphQLEnvironment fields: Set<String>): List<Product> =
        "http://localhost.charlesproxy.com:9090/products".httpGet(listOf(
            "ids" to items.joinToString(",") { it.productId },
            "include" to fields.joinToString(","))
        ).responseObject(jacksonDeserializerOf<Products>())
            .third.get()
            .products

    @GraphQLQuery(name = "images")
    fun images(@GraphQLContext product: Product,
               @GraphQLArgument(name = "limit", defaultValue = "0") limit: Int):
List<String> =
        product.images.take(
            if (limit > 0) limit else product.images.size)

}


@Configuration
class GraphQLConfig(private val cartService: CartService) {

    @Bean
    fun query() = object : GraphQLQueryResolver {
        fun hello() = "Hello, Unicorns!"

        fun cart(id: Long) = cartService.findCart(id)
```

```
    }

    @Bean
    fun itemResolver() = object : GraphQLResolver<Item> {
        fun product(item: Item): Product =
                "http://localhost:9090/products/${item.productId}".httpGet()
                        .responseObject(jacksonDeserializerOf<Product>())
                        .third.get()
    }

    @Bean
    fun productResolver() = object : GraphQLResolver<Product> {
        fun images(product: Product, limit: Int) =
                product.images.take(
                        if (limit > 0) limit else product.images.size)
    }

    @Bean
    fun mutations() = object : GraphQLMutationResolver {
        fun addProductToCart(cartId: Long, productId: String, quantity: Int) =
                cartService.addProductToCart(cartId, productId, quantity)
    }

}

@DataJpaTest
class RepositoriesTests @Autowired constructor(
    val entityManager: TestEntityManager,
    val userRepository: UserRepository,
```

```
        val articleRepository: ArticleRepository) {

        @Test
        fun `When findByIdOrNull then return Article`() {
            val juergen = User("springjuergen", "Juergen", "Hoeller")
            entityManager.persist(juergen)
            val article = Article("Spring Framework 5.0 goes GA", "Dear Spring
community ...", "Lorem ipsum", juergen)
            entityManager.persist(article)
            entityManager.flush()
            val found = articleRepository.findByIdOrNull(article.id!!)
            assertThat(found).isEqualTo(article)
        }

        @Test
        fun `When findByLogin then return User`() {
            val juergen = User("springjuergen", "Juergen", "Hoeller")
            entityManager.persist(juergen)
            entityManager.flush()
            val user = userRepository.findByLogin(juergen.login)
            assertThat(user).isEqualTo(juergen)
        }
    }


    @RestController
    class CartController(private val cartService: CartService, private val
productServiceRestClient: ProductServiceRestClient) {
```

```
        @RequestMapping("/carts/{id}")
        @ResponseBody
        operator fun get(@PathVariable id: Long,
                         @RequestParam(value = "projection", required = false)
projection: String?): ResponseEntity<Any> {
            val cart = cartService.findCart(id)
            if (cart == null) {
                return ResponseEntity.notFound().build<Any>()
            } else {
                return ResponseEntity.ok(if ("with-products" == projection) {
                    getProjectionWithProducts(cart)
                } else cart)
            }
        }

        private fun getProjectionWithProducts(source: Cart):
CartWithProductsProjection {
            val cart = CartWithProductsProjection()
            cart.id = source.id!!
            cart.items = toItemsWithProducts(source.items)
            cart.subTotal = source.getSubTotal()
            return cart
        }

        private fun toItemsWithProducts(source: List<Item>):
List<CartWithProductsProjection.Item> =
                source.map { sourceItem ->
                    val item = CartWithProductsProjection.Item()
```

```
                item.product =
toProductProjection(productServiceRestClient.fetchProduct(sourceItem.productId))
                item.quantity = sourceItem.quantity
                item.total = sourceItem.total
                item
            }

        private fun toProductProjection(source: Product):
CartWithProductsProjection.Item.ProductProjection {
            val product = CartWithProductsProjection.Item.ProductProjection()
            product.id = source.id
            product.title = source.title
            product.price = source.price
            product.sku = source.sku
            product.image = source.images[0]
            return product
        }

    }

    data class CartWithProductsProjection(var id: Long = -1L, var items:
List<Item> = mutableListOf(),
                                          var subTotal: BigDecimal = BigDecimal.ZERO) {

        data class Item(var product: ProductProjection = ProductProjection(), var
quantity: Int = 0,
                        var total: BigDecimal = BigDecimal.ZERO) {
```

```
        data class ProductProjection(var id: String = "", var title: String = "", var
price: BigDecimal = BigDecimal.ZERO,
                                     var image: String = "", var sku: String = "")
    }
}

fun main(args: Array<String>) {
    SpringApplication.run(CartServiceApp::class.java, *args)
}

@SpringBootApplication
class CartServiceApp
```