Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій

Кафедра програмних систем і технологій

УДК 004.492

На правах рукопису

ВИПУСКНА КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Тема: "Розроблення програмного забезпечення логістичної ситеми доставки продуктів харчування покупцю"

Спеціальність – 121 "Інженерія програмного забезпечення"

ПОЯСНЮВАЛЬНА ЗАПИСКА
ВКБР.ІПЗ - 22.00.00.000 ІПЗ

Студент

ІПЗ-42 _____ /Олександр ЛІПКОВ/

Науковий керівник

к. ф.-м. н., доц._____ /Ольга СУПРУН/

Консультант
з питань нормоконтролю

_____ /Тамара ЧАПОВСЬКА/

Допускається до захисту
Завідувач кафедри

д.т.н., проф. _____/Олексій БИЧКОВ/

Київ – 2021

Рішенням Екзаменаційної комісії

випускна кваліфікаційна робота студента

_____

захищена з оцінкою

_____

Голова Екзаменаційної комісії

д. т. н., проф. Віктор ВИШНІВСЬКИЙ

Київський національний університет імені Тараса Шевченка

Факультет інформаційних технологій
Кафедра програмних систем і технологій
Освітньо-кваліфікаційний рівень бакалавр
Спеціальність 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖЕНО
Зав. кафедри програмних систем і технологій
_____(Олексій БИЧКОВ)

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Ліпкову Олександру Євгеновичу

1. Тема бакалаврської роботи “Розроблення програмного забезпечення логістичної системи доставки продуктів харчування покупцю”, керівник роботи Супрун Ольга Миколаївна затверджені на засіданні кафедри програмних систем і технологій, протокол №6 від від **“** 11 ” листопада 2020 р.

2. Строк подання студентом роботи «___» _____2021 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції і формальні моделі побудови та функціонування інформаційних та програмних технологій певного класу

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити):

1. Проаналізувати існуючі підходи для доставки продуктів
2. Огляд існуючих систем, аналіз задачі та пошук їх переваг та недоліків
3. Вибір засобів розробки системи
4. Розробка системи доставки продуктів харяування.

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

1. ERC Діаграма (рис.1.1, стор.15).
2.Діаграма порівняння популярних ОС (рис.1.2, стор.18).
3.Графік зростання кількості запитів пошуку React Native (рис.1.3, стор.20).
4.Схематичне зображення основних елементів технології React Native (рис.1.4, стор.21).
5.Алгоритм роботи кур’єра та ресторана в процесі виконання замовлення (рис. 2.1 стор.23)
6. Архітектура роботи серверної частини додатку (рис. 2.2 стор 27)
7.Переваги і недоліки використання технологій, подібних Expo та React Native CLI(рис. 2.3 стор 27)
8.Алгоритм роботи з системою контролю версій git(рис. 2.4 стор 28)
9.Принцип роботи шаблону проектування Flux(рис. 2.5 стор 30)
10.Архітектура додатку для доставки продуктів харчування(рис. 2.6 стор 33)
11.Список файлів для Model(рис. 2.7 стор 35)
12.Приклад коду файлу Model(рис. 2.8 стор 36)
13.Список файлів проекту додатку для замовника послуг доставки продуктів харчування (рис. 2.9 стор 37)
14.Приклад роботи з елементом Flatlist(рис. 2.10 стор 38)
15.Сторінка авторизації в системі замовлення продуктів харчування(рис. 3.1 стор 40)
16.Етап вводу даних до системи для реєстрації користувача (рис. 3.2 стор 41)
17.Список кодів стільникових опреаторів різних країн(рис. 3.3 стор 41)
18.Сторінка вводу номера телефону(рис. 3.4 стор 42)
19.Приклад повідомлення, яке надсилається користувачам для авторизації в системі замовлення продуктів харчування(рис. 3.5 стор 42)
20.Інтерфейс головної сторінки замовлення продуктів харування(рис. 3.6 стор 43)
21.Процесс введення даних про місцезнаходження та адресу в ситему (рис. 3.7 стор 44)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Розділ 1	Ольга СУПРУН		
Розділ 2	Ольга СУПРУН		
Розділ 3	Ольга СУПРУН		
Розділ 4	Ольга СУПРУН		

Дата видачі завдання 11 листопада 2020 р.

Керівник _____ (Ольга СУПРУН)

Завдання прийняв до виконання _____ (Олександр ЛІПКОВ)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів бакалаврської роботи	Термін виконання етапів роботи	Відмітка про виконання
1.	Уточнення постановки задачі	20.11.2020-30.11.2020	Виконано
2.	Аналіз літератури	01.12.2021-25.12.2021	Виконано
3.	Аналіз існуючих методів, концепцій та алгоритмів роботи сервісів доставки харчувння	01.01.2021-28.01.2021	Виконано

4.	Обґрунтування вибору рішення	29.01.2021-28.02.2021	Виконано
5.	Опис розробленого алгоритму	01.03.2021-28.03.2021	Виконано
6.	Розроблення програмного забезпечення	29.03.2021-30.04.2021	Виконано
7.	Тестування розробленого програмного забезпечення	01.05.2021-20.05.2021	Виконано
9.	Оформлення і друк пояснювальної записки	21.05.2021-31.05.2021	Виконано
10.	Оформлення презентації	01.06.2021-09.06.2021	Виконано
11.	Отримання рецензії	10.06.2021	
12.	Затвердження пояснювальної записки роботи завідувачем кафедри	12.06.2021	
13.	Захист дипломної роботи	22.06.2021	

Студент – бакалавр _____(Олександр ЛІПКОВ)

Керівник роботи _____ (Ольга СУПРУН)

АНОТАЦІЯ

Випускна кваліфікаційна бакалаврська робота: 60 с., 32 рис., 3 додат., 10 джерела.

Тема: Розроблення програмного забезпечення логістичної системи доставки продуктів харчування покупцю.

Об’єкт дослідження: система доставки продуктів харчування споживачу.

Предмет дослідження: технології автоматизації логістичних методів для впровадження в сервіс доставки продуктів харчування

Мета роботи: покращення процесу продажу та доставки їжі клієнту

Результати дослідження:

Досліджено та проаналізовано існуючі підходи та алгоритми реалізації сервісів логістичної доставки продуктів харчування покупцям.

Висновок:

В результаті досліджень було розроблено програмне забезпечення для доставки та замовлення продуктів харчування. Дане програмне забезпечення може бути легко масштабовано, та впроваджено в процес роботи реальної компанії.

ЛОГІСТИКА, ДОСТАВКА ПРОДУКТІВ, АЛГОРИТМ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОКУПКИ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДОСТАВКИ

АННОТАЦИЯ

Выпускная квалификационная бакалаврская работа: 68 с., 32 рис., 3 доп., 10 источников.

Тема: Разработка программного обеспечения логистической системы доставки продуктов питания покупателю

Объект исследования: система доставки продуктов питания покупателю

Предмет исследования: технологии автоматизации логистических методов для Внедрение в сервис доставки продуктов питания

Цель работы: улучшение процесса продаж и доставки еды клиенту

Результаты исследования: Исследованы и проанализированы существующие подходы и алгоритмы реализации сервисов логистической доставки продуктов питания покупателям.

Вывод:

В результате исследований было разработано программное обеспечение для доставки и заказа продуктов питания. Данное программное обеспечение может быть легко масштабируемо, и внедрено в процесс работы реальной компании.

ЛОГИСТИКА, ДОСТАВКА ПРОДУКТОВ, АЛГОРИТМ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПОКУПКИ, ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ДОСТАВКИ

SUMMERY

Graduation qualification bachelor's work: 60 p., 32 figures, 3 ext., 10 sources.

Topic: "Software development of a logistics system of food delivery to the buyer»

Object of research: the system of food delivery to the buyer.

Subject of research: technology automation of logistic methods for food delivery service.

Purpose: Improve the process of sales and food delivery to the client.

To achieve the goal it is necessary to solve the following tasks:

- Analyze existing approaches to food delivery, ways to implement them, identify the advantages and disadvantages; the main tools for software development;
- To design and software to implement a system of food delivery and ordering.

Results of the study: As a result of the research the software for delivery and ordering of food products was developed.

Conclusion:

As a result of the research was developed system of food delivery and ordering. This software can be easily scaled and implemented in the process of work of the real company.

LOGISTICS, GROCERY DELIVERY, ALGORITHM, PURCHASE SOFTWARE, DELIVERY SOFTWARE

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД - база даних.

API - application programming interface (Програмний інтерфейс додатку).

ПЗ - програмне забезпечення.

ОС - операційна система.

HTML - HyperText Markup Language (мова гіпертекстової розмітки).

JS - Java Script.

CLI - Command Line Interface (інтерфейс командного рядка).

URL - Uniform Resource Locator (уніфікований покажчик інформаційного ресурсу).

СМС - Short Message Service (служба коротких повідомлень).

HTTPS - HyperText Transfer Protocol Secure.

ВСТУП

Сформована світова економічна система, покликана в більшій чи меншій мірі поліпшити якість поставленого товару і сприяти розвитку світового виробництва, змушує різних виробників, модернізувати свою роботу і знаходить все нові і нові способи для підтримки своєї конкурентоспроможності. З постійним розвитком ринку, автоматизація та вдосконаленя не оминають і таку сферу як збут продуктів харчування та їх доставку кінцевому користувачу. В країнах з розвиненою ринковою економікою логістика – основа успішного функціонування суб'єктів господарювання, а формування логістичної системи забезпечує підвищення ефективності економічних процесів та зниження загальних витрат підприємств. В сучасному світі проблема автоматизації стоїть близько до процесів внутрішніх оптимізацій виробництва. Застосування досягнень сучасних технологій на практиці в більшості є унікальним способом для підвищення ефективності, а також способом зменшення витрат за рахунок усунення непотрібних елементів в процесі. Оскільки для сфери послуг автоматизація найбільш доступний, а часто і єдино можливий спосіб підвищення ефективності виробництва, то особливий інтерес автоматизація внутрішніх процесів представляє саме для підприємств цієї галузі, зокрема, компаній, що займаються транспортною логістикою.

Метою кваліфікаційної роботи є вдосконалення процесу продажу та доставки їжі клієнту. Для досягнення мети, в ході роботи будуть виконані такі етапи:

- проведення аналізу предметної області та вже створених логістичних систем;
- розробка концепції мобільного додатку;
- вибір технологій, які будуть використані в мобільному додатку ;
- створення мобільного додатку.

Результатом роботи є мобільний додаток з налагодженою логістичною системою, яка готова до безпосередньої роботи в реальній компанії. Робота реально має практичну цінність: подібний додаток дозволить оптимізувати роботу виробництва, мінімізувавши роль людини в процесі прийняття та обробки замовлення. Людина, як суб'єктивна система завжди становить потенційну

небезпеку для будь-якого процесу, мобільний додаток знижує ймовірність помилки в цілому і усуває помилку, пов'язану з людським фактором.

В рамках цієї роботи було виконано аналіз акуальності ринку, дані використані при підготовці доповіді та написанні тез «Вплив пандемії COVID-19 на ринок онлайн доставки продуктів харчування» на Міжнародній науково-технічній конференції «8-ма Східно-Европейська конференція Математичні та програмні технології Internet of Everything»

РОЗДІЛ 1
АНАЛІТИЧНА ЧАСТИНА

1.1 Призначення і галузь застосування

Основним завданням сервісу, для якого було створено програмне забезпечення, є автоматизація пошуку, продажу, та доставки готової їжі користувачу цього сервісу.

Розроблений модуль може працювати в різних варіантах:

1. Як самостійний додаток, в якому будуть опубліковані статичний товар, що ввів в систему менеджер. Прикладом може слугувати додаток для ресторану, в якому крім явного замовлення на місці, користувач міг би ознайомитися з меню заздалегідь, зробити замовлення та через деякий час отримати своє замовлення в точці, яку він зазначив.
2. Як частина сервісу, в якому товари публікуються динамічно такими ж користувачами (user-generated-content). Прикладом може слугувати будь яка система-маркетплейс.

Розглянемо детально процес підготовки до надання логістичного обслуговування і безпосереднього надання послуги.

Першим етапом є відправлення заявки клієнтом на товар. Клієнт обирає зі списку потрібний товар, обирає кількість, заповнює дані про своє місцезнаходження, обирає спосіб сплати та доставки.

Наступним етапом він здійснює платіж. Після цього створюється платіжна інформація, коли товар буде готовий, він може бути доставлений користувачу.

Ціль програмного забезпечення, яке створювалось, є процес доступу, покупки та доставки їжі в межах одного міста. Для розуміння основних бізнес-процесів роботи компанії, яка б потенційно могла використовувати розроблене ПЗ, була створена діаграма ланцюгів процессу (EPC).

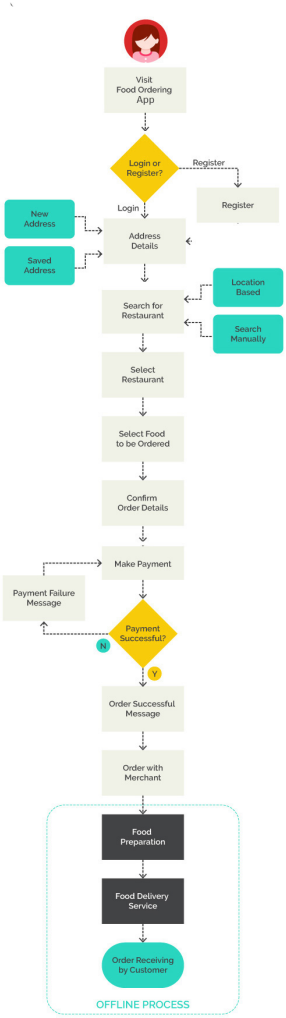

Рис. 1.1. EPC Діаграма

Як видно з рис.1.1, система управління замовленнями бере участь майже на кожному етапі виробництва замовлення. У загальному вигляді, щодо розглянутої предметної області, в системі є модель входження заявки, що надійшла від клієнта компанії, який бажає оформити замовлення.

У замовлення на протязі всього виробництва змінюється його статус, а саме:

- Waiting;
- in Progress;
- delivery;
- On the spot;

По готовності замовлення адміністратор відправляє дане замовлення з кур'єром, попутно закріпивши його в системі за певним кур'єром. Кур'єр при доставці замовлення за адресою повинен відзначити замовлення в системі, тим

самим змінивши його статус з «delivery» на «End». Це необхідно для ведення статистики часу доставки і інших важливих показників.

1.2. Обґрунтування необхідності впровадження програмного засобу

Робототa кур’єрської служби та служби доставки пов’язана з великою кількістю складних дій та рішень. Щоб зробити процес максимально надійним, потрібно мінімізувати втручання людини до роботи сервісу. Перехід на автоматизований режим дає можливість коректно відпрацьовувати періоди пікового навантаження на систему що в преспективі дає сервісу комерційний потенціал та ресурсоефективність. Також, як вказано в моїй публікації на даний момент сервіси такого типу являються дуже затребуваними у зв’язку з падемією Covid-19. У праці зазначено що платформа Deliveroo яка використовує аналогічний сервіс доставки продуктів харчування до розробленого змогла збільшити свої доходи на 54% в 2020 році[1-2]. Виходячи з цього факту можно стверджувати, що сервіси доставки продуктів харчування будуть розвиватися і далі, таким чином вони потребують постійного удосконалення та реалізації нових, самобутніх ідей.

1.3. Вимоги до програмного засобу

З урахуванням всіх факторів до програмного забезпечення були висунуті такі вимоги.

Створення двох незалежних клієнтських частин

1. Програмне забезпечення для вибору і покупки товару
2. Програмне забезпечення для доставки товару покупцю

Для програмного забезпечення 1 були поставлені наступні вимоги:

- мати інтуїтивно зрозумілий та зручний інтерфейс;
- кабінет покупця (реєстрація в сервісі, та авторизація);
- кросплатформеність;
- вільний вибір товарів, а саме їжі, по категоріям та цінам;
- оплата товару в додатку (готівкою або банківською картою);
- декілька видів доставки їжі, а сааме, самовивіз та доставка кур’єром.

1.3.1. Самовивіз

- Можливість отримати статус готовності їжі;
- Побудова оптимального маршруту в додатку;
- Побудова маршруту в сторонньому додатку;
- Можливість напряму зв'язатися з продавцем.

1.3.2. Доставка кур'єром

- Можливість напряму зв'язатися з продавцем;
- Можливість напряму зв'язатися з кур’єром;
- Повна інформація про статус готовності їжі;
- Онлайн переміщення кур'єра на карті.

Для програмного забезпечення 2 були поставлені такі вимоги:

- відображення всього списку замовлень;
- відображення кінцевої адреси замовлення на вбудованій карті;
- побудова маршруту в сторонньому навігаторі, який відкривається при натисканні на кнопку в додатку;
- можливість зв'язку з клієнтом.

1.4 Вибір цільової операційної системи

Проаналізувавши цільову аудиторію майбутнього сервісу, були зроблені висновки, що дане програмне забезпечення повинне бути реалізовано у вигляді мобільного додатку, було прибрано варіант реалізації, який би використовував технології для відображені в браузері, а саме веб-сервіс.

Основними причинами для цього були:

Недоліки мобільного сайту:

- менший рівень зручності, оскільки немає можливостей повною мірою задіяти всі функції смартфона (GPS, фото – і відео-камера);

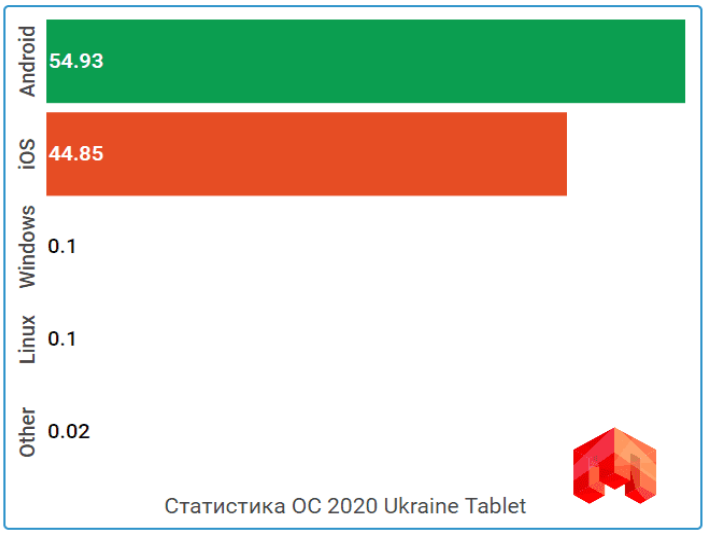

Рис. 1.2. Діаграма порівняння популярних ОС

- нижча в порівнянні з додатком швидкодія, оскільки швидкість завантаження залежить від інтернет-з'єднання;
- неможливість автономного доступу;
- не завжди хороший користувацький досвід, оскільки складно реалізувати один і той же інтерфейс на десктопних і мобільних пристроях

Переваги мобільного додатку:

- високий рівень інтерактивності, користувач може взаємодіяти з ним різними способами;
- можливість доступу до всього або більшої частини функціоналу пристрою, наприклад, акселлерометр, GPS-навігація, фотокамера і т. д .;
- мобільні додатки забезпечують кращий користувацький досвід, швидше завантажуються, і відповідають загальному UI конкретної ОС;
- завдяки можливості зберігати призначений для користувача контент, різні дані і проводити складні обчислення, забезпечується високий рівень персоналізації;
- ще одна важлива перевага мобільних додатків – їх можна використовувати в автономному режимі, без підключення до Інтернету.

Після того як було обрано формфактор пристроїв, на яких буде використовуватися дане програмне забезпечення, було поставлено наступне питання. Треба обрати для якої мобільної операційної системи буде розроблятися

дане програмне забезпечення. Для цього було проведений детальний аналіз цільової аудиторії та трендів на прикладі подібних сервісів.

Згідно з statcounter.com[4.] загальний рейтинг операційних систем в Україні, включаючи десктопи, мобільні, планшети та ігрові приставки, показує, що лідером є Windows, який встановлено на 59,81% пристроїв. Серед мобільних операційних систем першою за полярністю в Україні є Android який відсотково займає 82% ринку, операційна система iOS займає - 17% (майже кожний 6 мобільний пристрій). Статистика використання мобільних ОС на планшетних пристроях показує, що більш популярною ситемою є Android 54%, а IOS 44% Проаналізувавши всі зібрані дані, було прийнято рішення використовувати технології для створення кросплатформенних застосунків, так як загальна аудиторія яка користується смартфонами з ОС Android та IOS близька до рівноцінної. Таким чином, за рахунок створення програмного забезпечення з використанням кросплатформенних технологій буде досягнуто максимальне значення доступності застосунку для кінцевого користувача. Враховуючи вищезазначене, розглянемо схему роботи кросплатформенних додатків.

Кросплатформеність - це здатність ПЗ (в нашому випадку мобільних додатків) працювати на декількох платформах. Кросплатформена мобільна розробка дозволяє охопити дві операційні системи, iOS і Android, одним кодом. Вона не передбачає написання коду на рідній мові програмування, проте забезпечує майже нативний досвід завдяки інтерфейсу візуалізації з використанням власних елементів управління.

На поточний момент багато компаній використовують кросплатформені рішення, хтось вже всерйоз подумує перейти на них в найближчому майбутньому. Це не тільки вендори самих рішень, як, наприклад, Facebook зі своїм React Native, на якому працюють Instagram і Facebook, але й інші великі IT рішення, наприклад, на Flutter - Alibaba, Philips Hue, Hamilton, Tencent , Grab, Groupon та інші. У нашому випадку обирано React Native.

React Native – це високофункціональний інструмент розробки мобільних додатків, створений Facebook. Розробник може створювати програми для Android та iOS, використовуючи цю технологію. Як видно з діаграми на рис. 1.3, глобальний інтерес до React Native зростає.

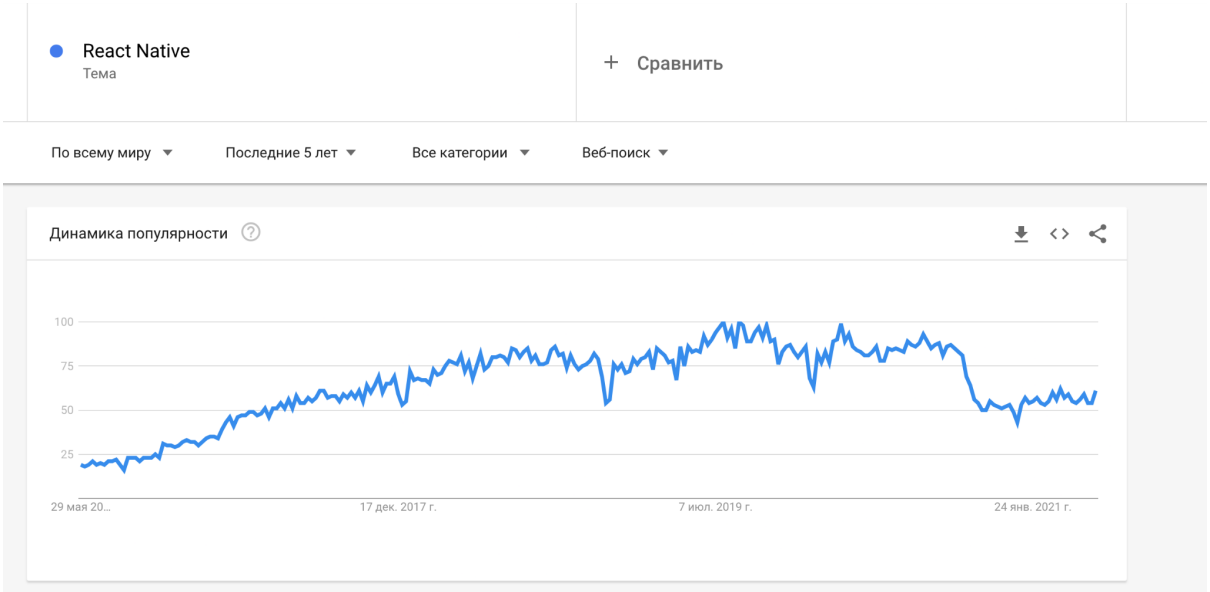

Рис. 1.3. Графік зростання кількості запитів пошуку React Native

Головною причиною вибору є швидкість створення додатків. React Native надає компоненти для тексту, зображень, екранної клавітури, списків, які можна прокручувати, панелі прогресу, анімації, буфера обміну, посилан тощо. На рис. 1.4 можна побачити основні елементи які надає React Native. Ці компоненти значно пришвидшують процес розробки додатків, а функція Hot Reloading також економить багато часу, оскільки дозволяє перезавантажувати додаток, не перекомпонуючи весь код. React Native надає можливість використовувати різноманітні бібліотеки, такі як Redux (менеджер станк додатку) та Awesome React Native (список компонентів та елементів) значно зменшить час на розробку додатку. Інструменти розробки, як Nuclid для написання коду, Yoga для побудови макетів, Sentry для моніторингу помилок і збоїв, і React Native для розробників для налагодження процесу розробки React Native за рахунок своєї відкритості та можливостей робить процес розробки значно простішим.

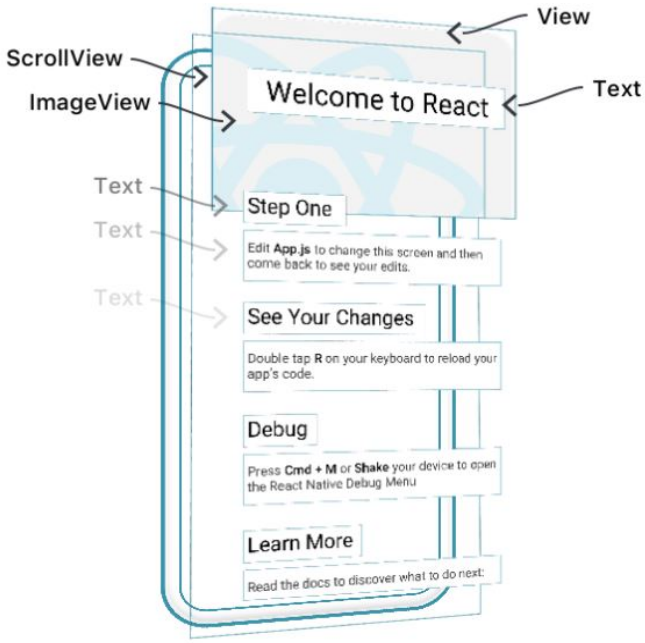

Рис. 1.4. Схематичне зображення основних елементів технології React Native

РОЗДІЛ 2
ТЕХНІЧНІ ХАРАКТЕРИСТИКИ

2.1. Постановка завдання на розробку системи

2.1.1 Опис алгоритму

Для реалізації завдання потрібно було розробити відповідні сервіси, які виконували функцію рекомендування товарів користувачу, покупки товарів, його доставки, та інформування користувача про статус виконання його замовлення, всі модулі розподілені між серверною частиною та самим додатком.

2.1.2. Опис логістичної моделі

Для реалізації логістичної оптимізованої системи доставки продуктів харчування були виведені основні критерії, які будуть задовольняти всіх суб'єктів, що входять в процес приготування, доставки та замовлення.

1. Мінімальне втручання в усі процес роботи даного додатку, система повинна бути автоматизована та незалежна.
2. Мінімізування втрат в процесі доставки, на етапі приготування та при передачі замовлення покупцю.
3. Реальні для виконання часові рамки. Кур’єр не повинен сильно поспішати, та замовник повинен отримати замовлення вчасно.
4. Користування системою для всіх учасників повинно бути інтуїтивним.

За допомогою таких вимог модель стає конкурентоспроможною для роботи в великих містах з великим попитом в період пікових навантажень.

Було прийнято рішення організувати роботу курєрів таким чином:

1. Кур’єру при активації програми доставки надається радіус в якому він буде отримувати замовлення. Радіус поділяється на два значення “порогове” та “абсолютне”. Нехай поргове - N1, абсолютне N2. N2 - буде розраховуватися на базі всіх курєрів в місті, це константа, яку вводить оператор, і вона статична для всіх виконавців. N1 - динамічна величина, яка змінюється в залежності від кількості кур’єрів в виділеному місці.

Таким чином, пріоритетний для вибору буде кур'єр, який знаходиться ближче до місця забору продукту, але у випадку не заповненості системи і відсутності виконавців на місцевості обирається кур’єр з відстанню від точки >N1.

2. Алгоритм привязки виконавця до замовлення можна побачити на рис.2.1.

Після підтвердження рестораном статус прийняття замовлення система знаходить кур’єра та передає йому координати закладу. В цей час починається етап приготування замовлення. Кур’єр отримує замовлення та везе на точку видачі, яка відображається у нього в додатку.

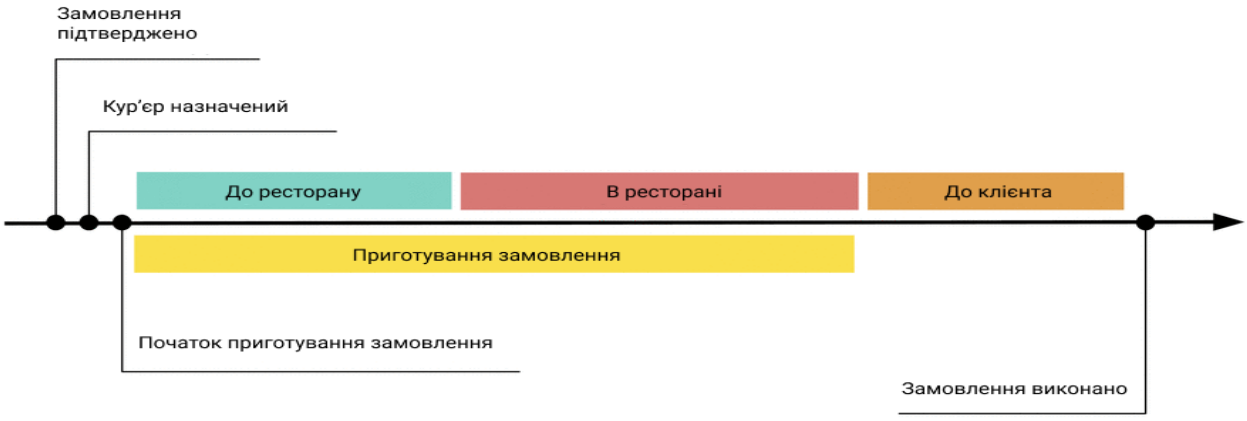

Рис. 2.1. Алгоритм роботи кур’єра та ресторана в процесі виконання замовлення

2.1.3. Опис і обґрунтування вибору складу технічних та програмних засобів

2.1.3.1. IDE

Для реалізації цієї задачі була обрана IDE WebStorm. WebStorm є створеною командою Jet Brains спеціалізованою версією PhpStorm,. WebStorm постачається з перед-встановленими плагінами JavaScript наприклад Node.js. WebStorm підтримує мови JavaScript, CoffeeScript, TypeScript та Dart. WebStorm забезпечує автодоповнення коду, його аналіх у процесі виконання, зручне переміщення по

проекту, рефакторинг, масштабний та зручний віджет для роботи з системами контролю версій (git). Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (у тому числі, рефакторинг коду JavaScript, що міститься в різних файлах і теках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладений Javascript) — в таких конструкціях підтримується коректний рефакторинг та конректна робота лінтера.

Основними причинами вибору цієї IDE була його тісна інтерграція з фреймворками JavaScript, також дуже важливим було те що в WebStorm є вбудований відладчик тобто можливо було відкрити точки останова в вихідному коді, переглянути значення в стеку, простежити за значеннями і використовувати інтерактивну консоль. Також перевагою було інтеграція з системами контролю версій. Була можливість використовувати простий універсальний інтерфейс для роботи з Git, GitHub, Mercurial і іншими системами контролю версій. Робити комміти, переглядати внесені зміни і вирішувати конфлікти можна було прямо в IDE.

Також один із основних факторів була підтримка роботи JSX який використовується в React Native для створення компонентів. Код підсвічується коректно та також є можливість проглянути документацію зразу на сторінці редактора.

2.1.3.2. Мови програмування

Для реалізації BackEnd частини було обрано середу виконання JavaScript – Node.js. Node.js - це серверна платформа для роботи з JavaScript через двигун V8. JavaScript виконує дію для стороннього клієнта, а вузол - на сервері.Node.js має можливість працювати з великою кількістю зовнішних бібліотек через вбудований менеджер біліотек npm, також має можливість роботи в режимі веб-сервера, та може викликати системні команди з середовища JavaScript.

Деякі провідні світові компанії використовують Node у виробництві, наприклад Netflix, Paypal, Walmart та Uber. Node часто використовується для створення резервних служб, що спілкуються з клієнтськими програмами. Ці програми отримують та надсилають дані за допомогою сервісу під назвою API. API служить інтерфейсом між різними програмами, щоб вони могли спілкуватися один з одним. Веб-додаток і мобільний додаток нижче можуть використовувати один і той же API для зберігання даних, надсилання електронної пошти, надсилання повідомлень або ініціювання робочих процесів на сервері.

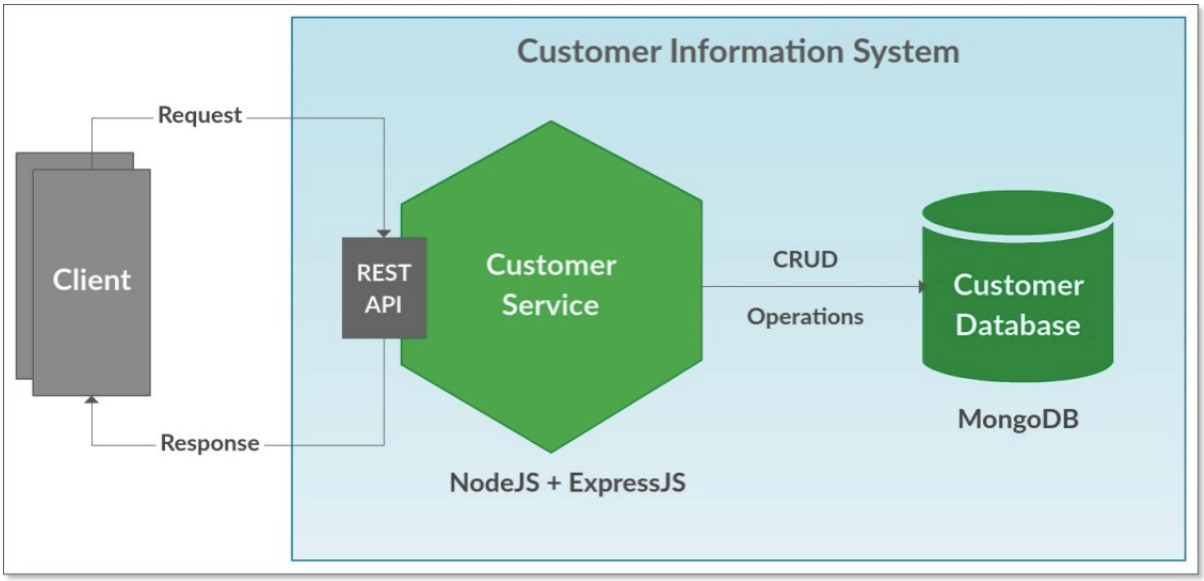

Рис. 2.2. Архітектура роботи серверної частини додатку

2.1.3.3. Додаток

Для створення додатку був обраний фреймворк для створення кросплатформенних застосунків React Native.

React-Native — це крос-платформний мобільний фреймворк, який дозволяє створювати програми, використовуючи лише JavaScript. А втім, на відміну від інших технологій, «мобільний веб-додаток» коли мова огортається компонентом який транслює програму у веб відображеня. Створений код на базі React Native

компілюється в мобільний додаток, який не відрізнятиметься від iOS додатка, побудованого з використанням Objective-C, Swift або ж Android додатка з використанням Java.Такі можливості остаточно дають React-Native безкомпромісні переваги якгібридних мобільних додатків так і у випадку нативних.

Плюси React Native:

1. Кросплатформеність. Під час розробки на React Native розробник створює одразу дві версії додатку, для ОС Anndroid з нативною мовою Android і також IOS на Objective-C
2. Швидко і дешево. Як наслідок першого пункту, не потрібно підтримувати дві платформи. Створю чи додаток з повною підтримкою однієї ОС автоматично ви отримуєте додаток який буде працювати і на іншій ОС, для повного функціювання потрібно тільки налаштувати конфігурації роботи з нативними функціями операційної системи в коді
3. Велике співтовариство. Безліч спільнот в попклярних каналах зв’язку зможуть легко допомогти вам з будь-яким питанням стосовно роботи з цією технологією.
4. Багатий вибір доступних компонентів і готових рішень. Головна умова завдання - швидкий запуск, тому не було часу писати бібліотеки і рішення з нуля. Готові компоненти і рішення нам в цьому допомогли, як і в переході від ReactJS до React Native.

Expo Це шар абстракцій - набір інструментів, бібліотек і сервісів для швидкого запуску. Це деякий API, який «з коробки» дає доступ до можливостей пристрою: до камери, геолокації, push-повідомленнями. В Expo є інструменти налагодження і тестування програми. Алгоритм простий: встановлюємо на телефон додаток Expo Client, підключаємо телефон з комп'ютером в одну локальну мережу, заходимо в Expo на телефоні, відкриваємо що розробляється. Всі зміни в коді відображаються моментально. Це допомагає тестувати і працювати в команді.

Зліт на Expo швидкий і стрімкий. З Expo не потрібні Xcode і Android Studio. Пишемо в нашому середовищі розробки, збираємо додаток за допомогою сервісу Expo, підписуємо і оновлюємо на льоту. Але є і недоліки. Не можна додавати зроблені власноруч нативні модулі. Наприклад, система дмтсанційих push-повідомлень працювати стабільно не буде якраз з причини браку підтримки модулів. Expo - шар абстракцій над React Native. Не можна оновитися на нову версію фреймворку, поки не оновиться Expo.

React Native CLI. Його перевага в можливості кастомізації, додавання нативних модулів. Але для підпису і збірки додатку (і публікації) потрібен Xcode або Android Studio. Щоб протестувати додаток, необхідно зробити збірку і завантажити на телефон, наприклад, за допомогою TestFlight для iOS або прямий установки з ПК на телефон. Стандартна схема, але незручно і довго.

Особенности	React Native CLI	Expo
Нативные модули (Java / Objective-C)	✚	▬
Размер приложения “Hello, world!”	5 Мб	25 Мб
Не требуется XCode / Android Studio	▬	✚
Шрифты и иконки из коробки	▬	✚
JS API: камера, местоположение, нотификации, и др.	▬	✚
Сторонние библиотеки, содержащие нативный код	✚	▬
Удобное тестирование на устройстве	Сложнее	Легче

Рис. 2.3. Переваги і недоліки використання технологій, подібних Expo та React Native CLI

2.1.3.4. Бібліотеки та модулі

Express

Це веб-фреймворк, який дозволяє структурувати веб-додаток для обробки декількох різних запитів http за певною URL-адресою. Express - це бібліотека для Node.js, створена для того, щоб значно спростити розробку веб-сайтів, веб-додатків та API.

Важливою частиною проекта, а особливо проекта на node.js є NPM.

Npm, менеджер пакутів Node, це дві речі: насамперед, це онлайн-сховище для публікації проектів Node.js з відкритим кодом; по-друге, це утиліта командного рядка для взаємодії зі згаданим сховищем, яка допомагає в установці пакунків, керуванні версіями та управління залежностями. Безліч бібліотек і програм Node.js публікуються в npm, і багато інших додаються щодня. За допомогою npm менджера пакунків можна встановити будь яку бібліотеку за допомогою єдиного рядка. У випадку, коли для проекту потрібна будь-яка новітня технологія, її можливо встановити з сховища npm. Npm дуже простий у використанні: потрібно запустити npm install async, і вказаний модуль, який необхідно встановити, він буде інстальваний у поточному каталозі в директорії ./node_modules/. Після встановлення у папку node_modules можливо використовувати на всі інтальовані бібліотеки виклик «()», так само як і на вбудовані модулі.

2.1.3.5 Робота з git

У великих проектах з численною командою розробників необхідна система контролю версій, яка дозволить декільком розробникам працювати паралельно над одним проектом, збереженим у віддаленому сховищі. Віддалене сховище також потрібно для збереження даних в разі їх втрати на фізичному носії. Переваги Git над іншими системами контролю версій в тому, що більшість операцій доступні локально, без підключення до мережі. Це дає можливість працювати над проектом в файлової системі без підключення до мережі, а при можливості підключення до мережі можна легко інтегруватися з віддаленим репозиторієм.

У Git є три стани, в якому можуть знаходитися файли:

- зафіксоване (committed) - означає, що файл вже збережено в локальної базі;
- змінений (modified) - означає, що файл зазнав змін,які ще не були зафіксовані;
- підготовлене (staged) - змінені файли, відмічені для включення в наступний коміт.

Отже, у проекту Git є три основні секції:

1. Git-директорія - це те місце, де Git зберігає метадані та базу об'єктів вашого проекту. Це найважливіша частина Git, і це та частина, яка копіюється при клонуванні сховища з іншого комп'ютера.
2. Робоча директорія є «знімком» версії проекту. Файли розпаковуються зі стислої бази даних в Git-директорії і розташовуються на диску, для того щоб їх можна було змінювати і використовувати.
3. Область підготовлених файлів - це файл, зазвичай розташовується у вашій Git-директорії, в ньому міститься інформація про те, які зміни потраплять в наступний Комміт.

Схематично цикл зміни файлів представлений на рис. 2.4.

Рис. 2.4. Алгоритм роботи з системою контролю версій git

2.2.Проектування додатку

2.2.1 Шаблон проектування Flux

Як було описано раніше, використання React Native не накладають ніяких обмежень на структуру проекту і роботу з JavaScript. Таким чином, не було жодних обмежень у виборі архітектурного підходу для додатків які розроблялися в період виконання роботи. Тож для данного завдання була обрана архітектура Flux. Його стректура представлена на рис. 2.5.

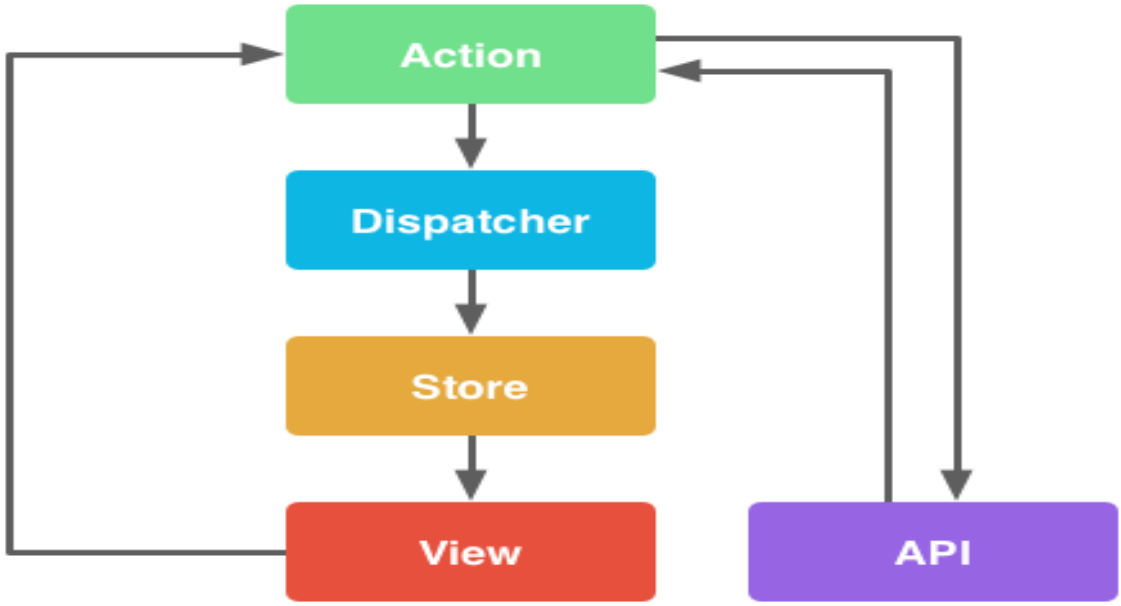

Рис. 2.5. Принцип роботи шаблону проектування Flux

Таке рішення було обґрунтовано:

- вже наявному доствіду роботи з таким архітектурним шаблоном;

- Наявності великої кількості інформації у відкритих джерелах, також досить дружна аудиторія розробників яка зможе завжди допомогти в процесі розробки використовуючи Flux

- простотою і ефективністю даного підходу. При його використанні проект легко масштабувати, причому його використання підходити як для малих проектів, так і для великих комерційних.

Flux-архітектура - архітектурний підхід або набір шаблонів програмування. Використовується для коректної побудови користувацького інтерфейсу мобільних додатків. Цей підхід зарекомендував себе у роботі в зв’язці з React Native і його основна особливість побудова на односпрямованих потоках. Основною відмінною рисою Flux є одностороння спрямованість передачі даних між компонентами Flux-архітектури. Архітектура Flux створює ліміти на потік даних, зокрема, виключаючи можливість поновлення стану компонентів рекурсивно. Такий архітектурний підхід робить потік даних лінійним в результаті його лекго передбачити, що значно допомоагає у розробці та у майбутньому також допоможе у пошуку та вирішеннб помилок. Flux-архітектура складається з трьох компонентів:

- уявлення - компонент, який реалізує видачу інформації користувачеві. У клієнтських додатках може бути представлений сторінкою в веб-браузері (або в мобільному додатку) або відокремленим модулем на ній. У уявлення є доступ до читанняінформації зі сховища, але немає можливості безпосередньо змінювати її: додавати нові дані, редагувати або видаляти існуючі. як приклад уявлення, можна привести сторінку зі списком доступних замовлень для користувача в додатку, що розробляється. Вся інформація на замовлення зберігається в сховищі, в поданні вона лише перетворюється для зручного відображення в інтерфейсі користувача;
- сховище - об'єкт, в якому зберігаються всі дані, використовувані в додатку. Дані з сховище можна зчитувати, але не можна змінювати безпосередньо. Для їх зміни існують дії, які можна відправляти в сховище з поданні та інших модулів проекту. І вже в залежності від типу дії і даних, зазначених в ньому, сховище саме визначить, які зміни потрібно провести з даними;
- дію - простий об'єкт. Насправді, може мати будь-яку структуру, але загально прийнято має два поля: поле, що визначає тип дії, і поле, що зберігає дані для цього дії. Тип дії представляється рядком, в залежності від типу буде викликаний потрібний обробник в сховище. Дані для дії представлені об'єктом з довільною структурою. При такій розбивці сховище ніяк не залежить від уявлення, тому його можна легко масштабувати: додавати нові поля для опису сутностей, які

використовуються в проекті, і нові обробники для дій. Оскільки всі дані розташовані в сховище, а в уявленнях вони лише перетворюються, то відсутня необхідність відслідковувати і обробляти зміни вироблені з ними - в уявлення завжди будуть передані актуальні дані. Через те, що уявлення не може безпосередньо змінювати дані в сховище, чи не буде конфліктів при додавання нових інтерфейсів (Подань) до проекту і зміна існуючих. Простота масштабування, низька складність і висока швидкість розробки роблять даний архітектурний підхід дуже вдалим і поширеним серед проектів клієнтських додатків.

2.2.2 Архітектура програмного продукту

За час виконання роботи було створено 3 повноцінні програми:

- серверна частина (Back-End);
- мобільний додаток покупця (Android та IOS);
- мобільний додаток курьера.

Користувацькі програми, що розроблені, є клієнт-серверними. У випадку додатку покупця для своєї роботи потрібно отримувати дані про товари з віддаленого сервера. В додатку для кур'єра для коректної роботи також повинен бути постійний зв'язок з сервером для отримання інформації. Загальна схема роботи клієнт серверних додатків представлена на рис. 2.6.

Спочатку клієнт надсилає серверу http-запит, далі, якщо сервер може розпізнати цей запит, він при необхідності звертається до бази даних, яка поверне серверу запит. Сервер перетворює дані в необхідний для клієнта формат JSON і відправляє йому.

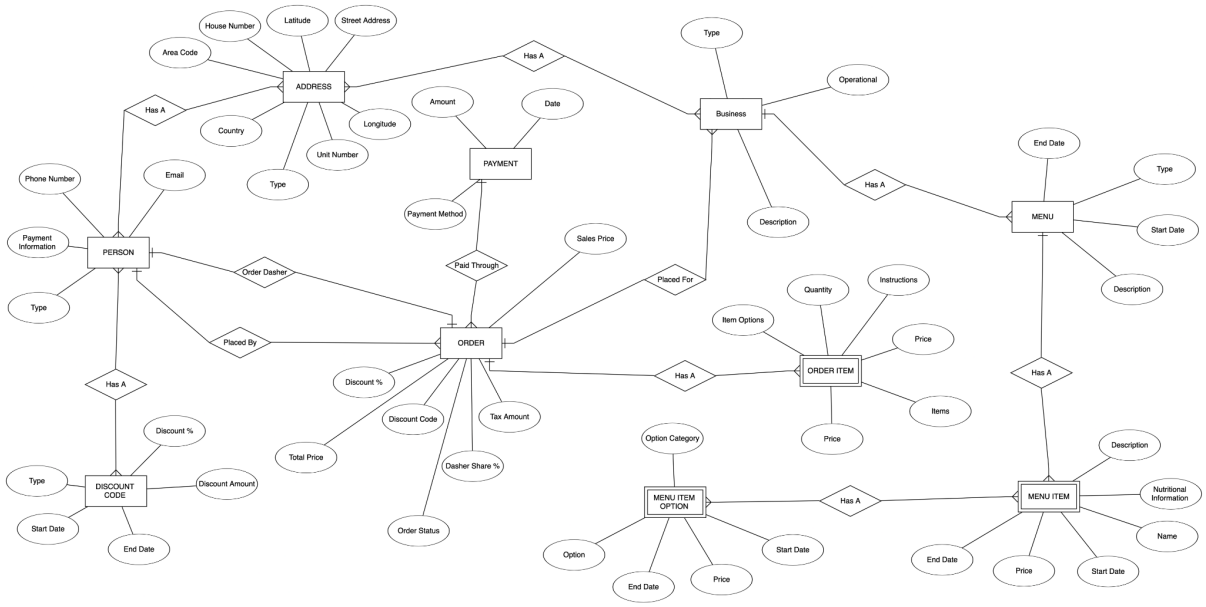

Рис. 2.6. Архітектура додатку для доставки продуктів харчування

Для роботи програми з API серверу був реалізований класс. Всі запити до API сервера виконуються через протокол https. Серверна частина виконується на віддаленому сервері, де був виконаний Port-forwarding для роботи захищеного підключення. Також було куплене доменне ім’я, на який зсилався сервер.

Крім звернення до API сервера, дані на мобільний пристрій можуть надійти з push-повідомленням. Для цього був використаний сервіс Google FireBase. Через APi цього сервісу можливо відправляти дані push-повідолень в Listener, який в свою чергу, виведе це повідомлення на екран навіть коли додаток знаходиться не в запущеному стані. У повідомленні міститься тип даних, щоб на мобільному пристрої можна було зрозуміти, як їх обробити, наприклад orderPublished, і самі дані. Для роботи з даними був реалізований клас NotificationManager.

Для роботи з локальними даними, які потрібно зберігати навіть після перезапуску додатку, використовується бібліотека AsyncStorage. У випадку додатків, які розроблялись, AsyncStorage був потрібний для зберігання токену для аутентифікації, обраного міста, адреси, дані користувачів та інші.

Інтерфейс користувача додатку замовлення їжі був реалізований через класи:

- BuyerHotFood - сторінка перегляду продуктів, які готові до відправлення;
- BuyerLeftUserTool - сторінка вибору функцій та налаштувань профілю користувача;
- BuyerProductBuyView - сторінка оформлення замовлення;
- BuyerMainView - основна сторінка, де відображаються вся їжа, яка пропонується покупцеві;
- BuyerOrderProfile – сторінка, де наявний статус всіх замолених страв;
- BuyerMaps - сторінка для відображення карт користувачу;
- SIgnIn - сторінка авторизації;
- SignUp - сторінка реєстрації.

2.2.2.1. Архітектура і реалізація серверної частини

В процесі розробки було прийнято рішення умовно розділити методи на декілька частин:

- Account;
- Delivery;
- Notification;
- Payments;
- Products;
- Chat.

Account має в собі Api для обробки всіх процесів для початкової ініціалізації користувача, тобто додавання початкової адреси, редагування профілю користувача, збереження зображень та інше.

Delivery відповідає на запити, пов'язані з доставкою продуктів користувачу.

Notification виконує завдання обробки запитів від інших процесів які хочуть відправити на додаток push або internal повідомлення

Payments працює для документування та обробки всіх платежів, він безпосередньо працює з сервісом Stripe.

Stripe - американська технологічна компанія, що розробляє рішення для прийому і обробки електронних платежів. Основна причина вибору подібного сервісу - простота використання, оскільки у даного сервісу є велика документація та досить зручні OpenApi.

Розділ Payments виконує функцію додавання банківських карт до бази даних, виконання транзакцій та створення “Invoices”.

Chat - обробник повідомлень. В цьому розділі також знаходиться метод, який виконує поєднання двох клієнтів в одну сесію.

Products - виконує велику кількість функцій для створення, редагування, рекомендації товарів користувачу.

Також для коректної роботи серверної частини були створені Model.

За допомогою Model ініціалізується колекції в БД. Кожен файл Model відповідає за свою колекцію. В собі мають поля require. За допомогою них бібліотека mongoose перевіряє об’єкти на обов'язковість отримання інформації.Також мають поля default, за допомогою цього поля можна вказати дані, які будуть записані в БД за замовчуванням у випадку не отримання даних від користувача.

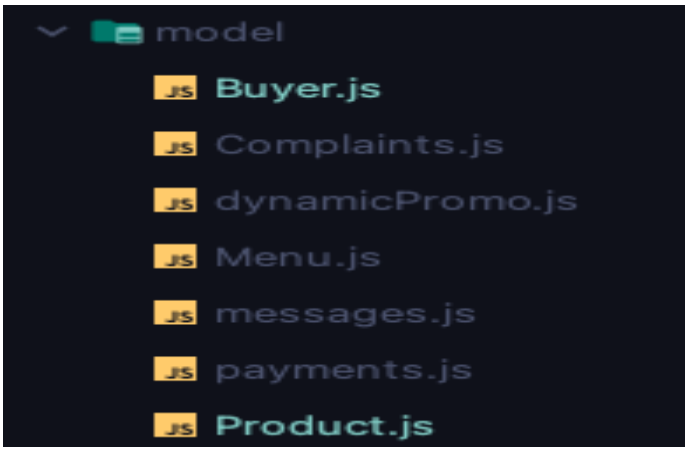

Рис. 2.7. Список файлів для Model

```
const userSchema = new mongoose.Schema({
    name: {
        type: String,
        required: true,
        min: 3,
        max:200
    },
    surname: {
        type: String,
        required: true,
        min: 3,
        max:200
     },
    email: {
        type: String,
        required: true,
        min: 6,
        max:200
    },
    password: {
        type: String,
        required: true,
        min: 6,
        max:1000
    },
    phone: {
        type: String,
        required: true,
        min: 6,
        max:200
    },
    date: {
        type: Date,
        default: Date.now
    },
    iconImageURL: {
        type: String,
        default: "https://res.cloudinary.com/mealaway/image/upload/"
    },
    payInfoStripe:[],
    historyOfProduct:[],
    historyOfPay:[],
    addressInfo: [],
    dateOfLastEnter: {
      type: Date,
      default: Date.now
    },
    notificationInfo:[],

});
```

Рис. 2.8. Приклад коду файлу Model

Для створення доставки процесу доставки продукту був проведений пошук алгоритмів для пошуку найближчого кур’єра. Після проведеного аналізу та пошуку було прийнято рішення використовувати алгоритм Дейкстрі. Він має найменшу складність: O (V * E). Це дозволить системі бути більш гнучкою і продуктивною. Що, в свою чергу, збільшить ефективність роботи кур’єрської служби.

Для збереження координат кур`єра використовувався один з сервісів FireBase. Таким чином, період відправки запитів на сервер був лімітований затримкою в 2 хвилини. На протязі цього часу клієнтський додаток збирав маршрут в масив та відправляв на FireBase. З такою ж затримкою запити на отримання координат відправляв і додаток покупця. За допомогою такого способу отримуємо трансляцію координат з одного пристрою на інший.

2.2.2.2. Архітектура і реалізація додатків

Після дослідження потреб ринку були прийнято рішення про створення додатків на базі кросплатформної утиліти React Native. Таким чином, всі програми, які були створені за час написання роботи мають можливість роботи на декількох мобільних ОС. Порівнюючи з нативними мовами програмування у кросплатформної реалізації є деякі відмінності. Структура додатку для покупця зображена на рис.2.9.

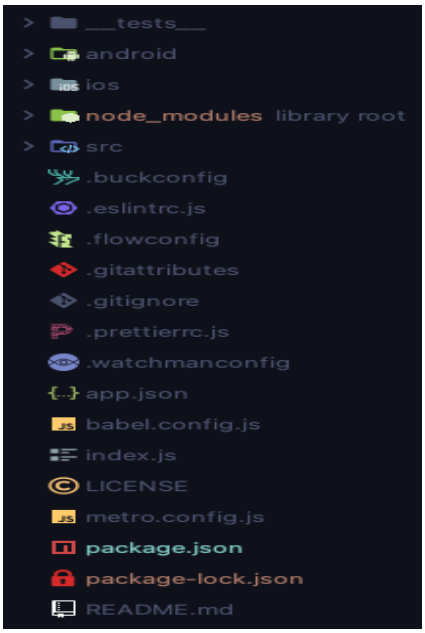

Рис. 2.9. Список файлів проекту для замовника послуг

На зображенні можна побачити дві директорії “Android” та “IOS”. В цих директоріях знаходяться системні файли які являються пріоритетними для кінцевої ОС.

В ios директорії знаходиться директорія PODS та документ Podfile. Podfile це конфігураційний файл для менеджера залежностей Cocoapods. Cocoapods використовується в проектах Swift та Objective-C. Його основна функція це включення сторонніх бібліотек, фреймворків в кінцевий продукт.

В React Native для створення сторінок використовується розширення мови JavaScript - JSX. За допомогою цього семантичної мови можна створювати

елементи, які далі будуть рендеритися на самому пристрої. Основний компонент, який використовується в системі це ScrollView. Цей компонент надається стандартною бібліотекою react-native. За допомогою цього компента будуються майже всі сторінки в додатку. Його основне застосування - розширення області екрана за межі фізичного розміру екрану пристрою. Таким чином можливо помістити всю потрібну інформацію в сторінку. Також популярним компонентом, який використовується в додатку є FlatList. Він наслідує частину Props від ScrollView, за допомогою чого теж має можливість “прокручування”. Його основною функцією є те, що він може розділити масив даних на окремі частини і по черзі згенерувати елементи з цього списку.

```
<FlatList
    data={foods}
    horizontal
    keyExtractor={(item) => `${item._id}`}
    renderItem={this.renderItem}
    style={{backgroundColor: COLORS.lightGray4}}
    showsVerticalScrollIndicator={false}
    showsHorizontalScrollIndicator={false}
    contentContainerStyle={{
        paddingHorizontal: SIZES.padding * 2,
        paddingBottom: 30,
    }}

/>
```

Рис. 2.10. Приклад роботи з елементом Flatlist

data - об’єкт, який приймає повний масив даних, які плтрібно розбити по ітераціям.

keyExtarctor - функція, яка отримує в якості першого аргументу елемент з даних (в цьому зворотного виклику він називався x), а другий аргумент-індекс елемента в масиві даних (в даному випадку називається i).

renderItem- функція, що приймає об’єкт і відображає його на екрані.

РОЗДІЛ 3
ОПИС РОБОТИ ПРОГРАМ

3.1. Інсталяція програмного забезпечення на пристрій

Для того, щоб встановити додаток на пристрій, необхідно мати смартфон під управлінням операційної системи Android версії не нижче 5.0 або пристрій з ОС IOS версії не нижчою, ніж 9.0, здатний виходити в мережу інтернет, а також повинна бути підтримка системи позиціонування GPS. Інсталяція відбувається запуском apk файлу на пристрої у випадку Android. У випадку IOS використовується програмне забезпечення з назвою TestFlight.

TestFlight - офіційний продукт компанії Apple для відкритого бета-тестування, призначений для iOS-пристроїв, щоб полегшити процес збору кодів тестових пристроїв. Даний сервіс дозволяє легко запросити користувачів протестувати додатки для iOS, watchOS і tvOS, перш ніж вони будуть випущені в App Store.

Через TestFlight поширюються додатки, які можна тестувати тільки для версій iOS від 8 та вище, так як на даний момент iOS-пристроїв з більш старими версіями майже не використовується, оскільки політика Apple обмежує використання пристроїв на застарілих версіях. Для завнтаження сервіс безкоштовний, але щоб використовувати його повноцінно треба мати Apple Developer аккаунт, реєстрація в програмі коштує 99 доларів США.

Основні функції TestFlight:

- відкриття доступу до додатка на 1000 бета-тестувальників, використовуючи лише адреси їх електронних пошт;
- можливість використовувати до 5 пристроїв на один аккаунт тестувальника;
- можливість завантажувати, встановлювати та оновлювати тестові версії додатків розробниками;
- проста і швидка передача продукту на перевірку до піблікації на маркетплейс додатків після того, як бета-тестування завершено;

- можливість завантажувати офіційний додаток з App Store без етапів пошуку і відправки розробнику UDID-коду свого девайса для прийняття участі в тестуванні;

- отримання повідомлення про появу нового білду тестованого продукту відразу на пристрій тестувальника використовуючи push-повідомлення;

- можливість для бета-тестувальників залишати свої відгуки через додаток TestFlight, і замовник або розробник отримають їх на електронну пошту, або переглянути через додаток розробника на сайті;

- відображення кількості раз запуску і його термінового закриття по причині критичної помилки. Додатково можна отримувати деяку інформацію про налагодження.

3.2.Реєстрація користувача в кабінеті покупця

Після завантаження додатку та його запуску, програма відкриє сторінку авторизації (рис. 3.1)

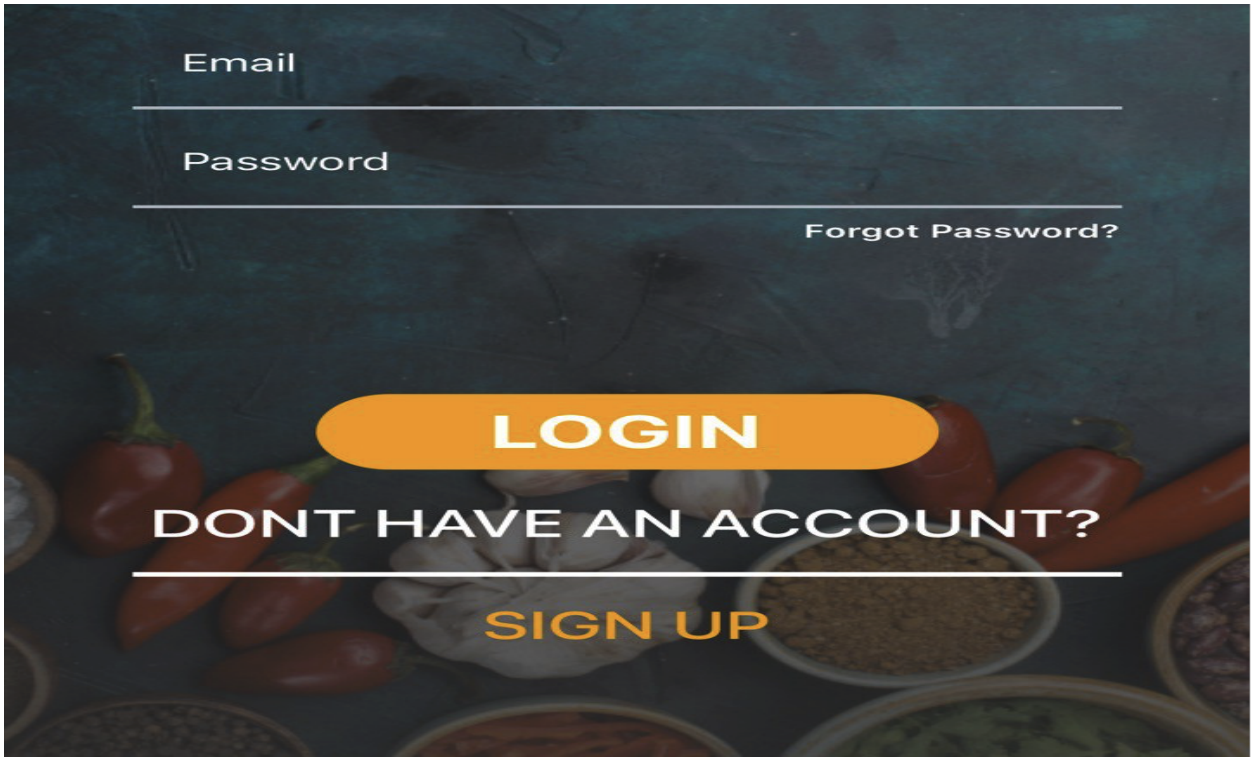

Рис. 3.1. Сторінка авторизації в системі замовлення продуктів харчування

Для переходу на екран реєстрації потрібно натиснути кнопку SIGN UP та перейти на перший крок реєстрації де потрібно ввести ім’я та адресу електронної пошти користувача що зображено на рис. 3.2.

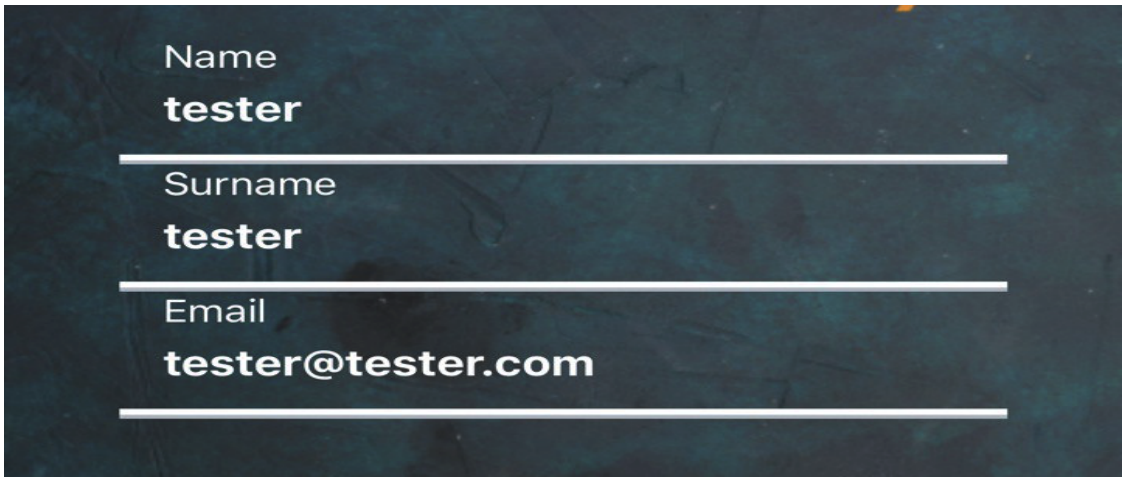

Рис. 3.2. Етап вводу даних до системи для реєстрації користувача

Після заповнення даних користувач перенаправляється на сторінку заповнення номеру телефону (рис. 3.4), користувач повинен вибрати код країни де працює оператор його стільникового телефону (рис 3.3) та ввести номер телефону. На телефон прийде СМС повідомлення в якому буде міститись код підтвердження (рис 3.5).

Рис. 3.3. Список кодів стільникових опреаторів різних країн

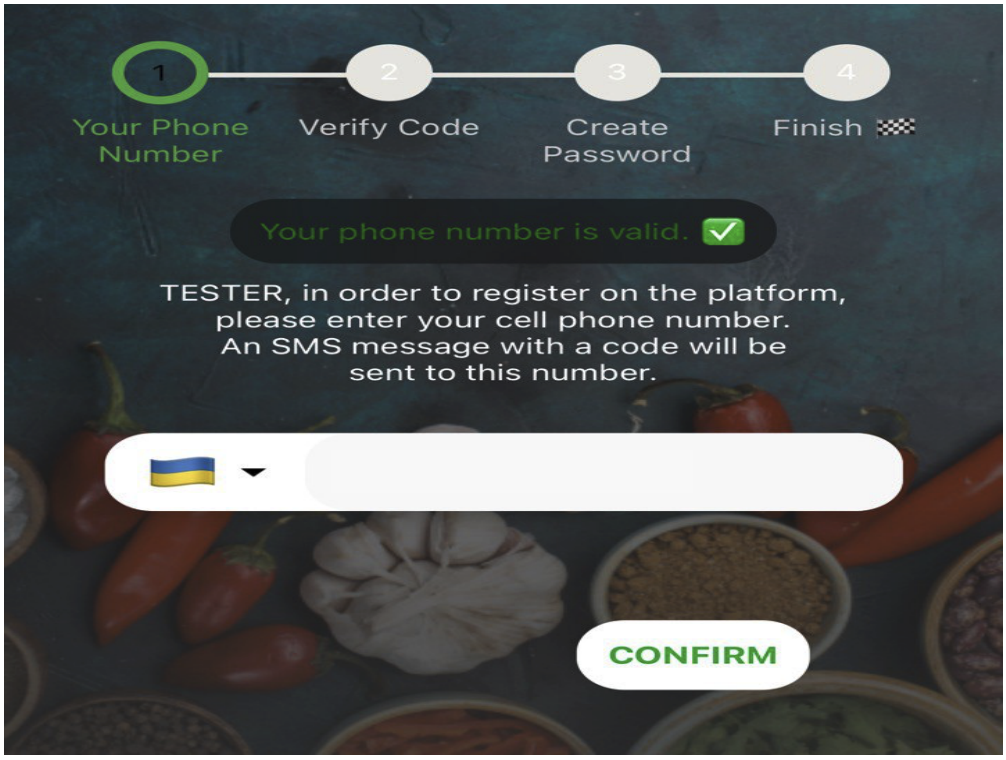

Рис. 3.4. Сторінка вводу номера телефону

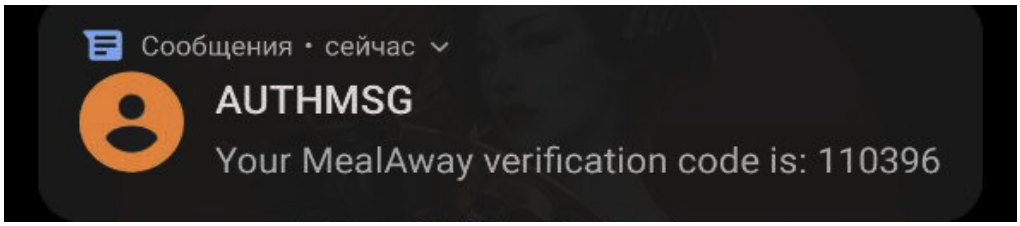

Рис. 3.5. Повідомлення, яке надсилається користувачам для авторизації

Після цього користувач повинен ввести код з повідомлення до наступної сторінки та ввести свій новий пароль. По завершенню всіх дій процес реєстрації завершиться та користувач буде перенаправлений на сторінку авторизації. На цій сторінці він повинен буде авторизуватися за допомогою даних що були використані при реєстрації.

3.3. Замовлення страви

Після авторизації в додатку користувач перенаправляється на головну

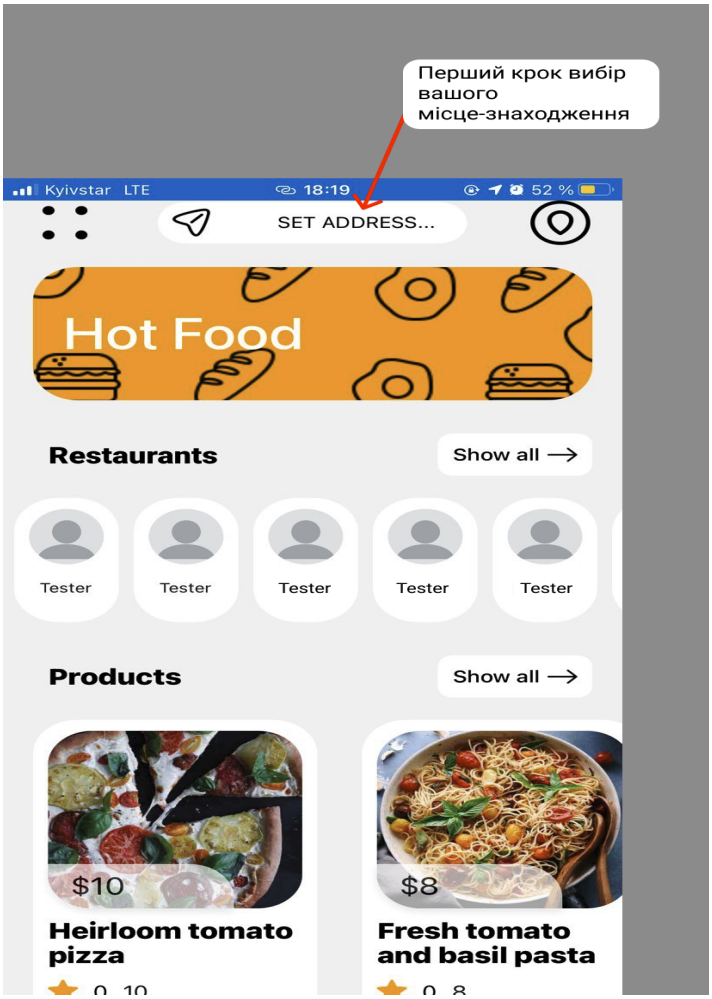

Рис. 3.6. Інтерфейс головної сторінки замовлення продуктів харування

сторінку де йому будуть відображені страви що можна замовити. Також програма попросить користувача ввести свою адресу для відображення товарів поблизу (рис. 3.6).

Щоб користувачу додати адресу, він повинен зайти в меню адреси натиснути кнопку “Add new address” наступним вікном він побачить карту, на якій можна обрати місце, де він зараз знаходиться, цей процес відображений на рис. 3.7. Це можливо зробити двома способами:

- Перетягуванням карти і встановленням маркеру

- Введенням точної адреси в поле зверху дісплея.

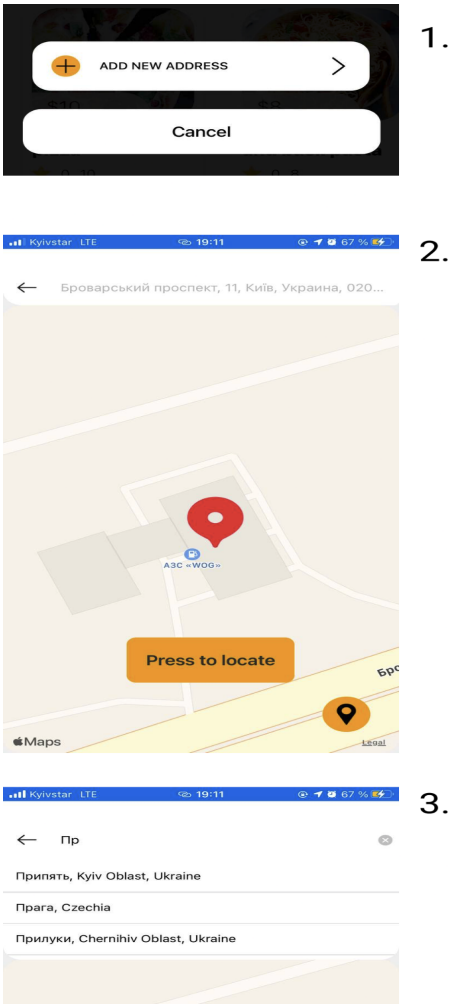

Рис. 3.7. Процес введення даних про місцезнаходження та адресу в ситему

Після введення початкової адреси, можна перейти до процесу замовлення. Після обрання товару, вибору його кількості програма перенаправляє покупця на сторінку замовлення. На цій сторінці користувач повинен обрати такі параметри:

- місце, куди буде доставлятися продукт;
- тип оплати (Банківською картою, готівкою);
- тип доставки продукту (самовивіз, доставка кур’єром);
- час доставки продукту (у випадку коли обраний тип доставкою кур’єром);
- вибір банківської карти з тих що прив’язані до аккаунту (рис 3.9).

Сторінка вибору конфігурації замовлення відображена на рис. 3.8.

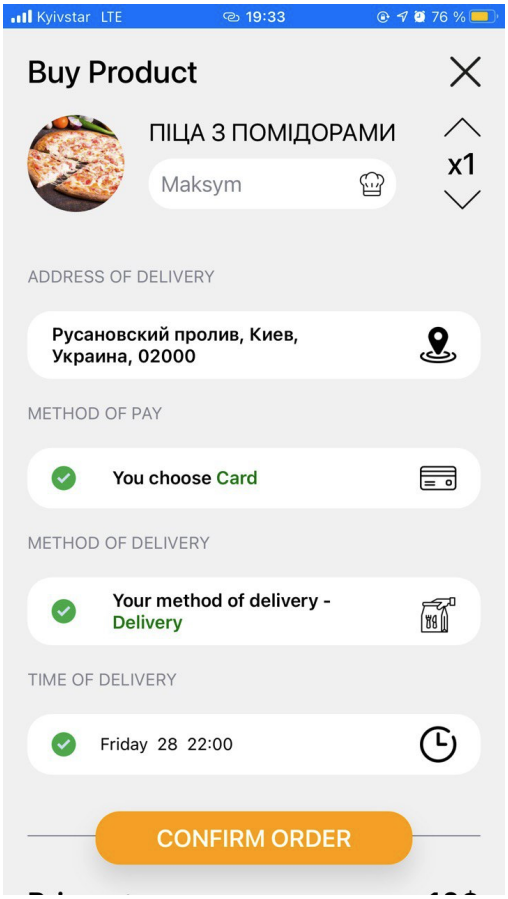

Рис 3.8. Сторінка вибору конфігурації замовлення

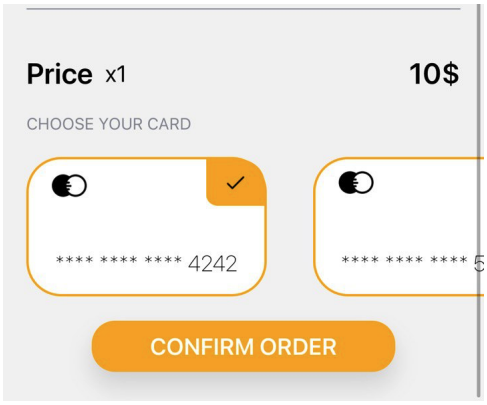

Рис. 3.9. Компонент для обрання банківської карти

Після заповнення юзер може замовити та оплатити замовлення (у випадку банківської карти).

Коли сервер відправить на клієнтський додаток позитивний статус виконання платежу програма перенаправить користувача до списку замовлень (рис 3.10).

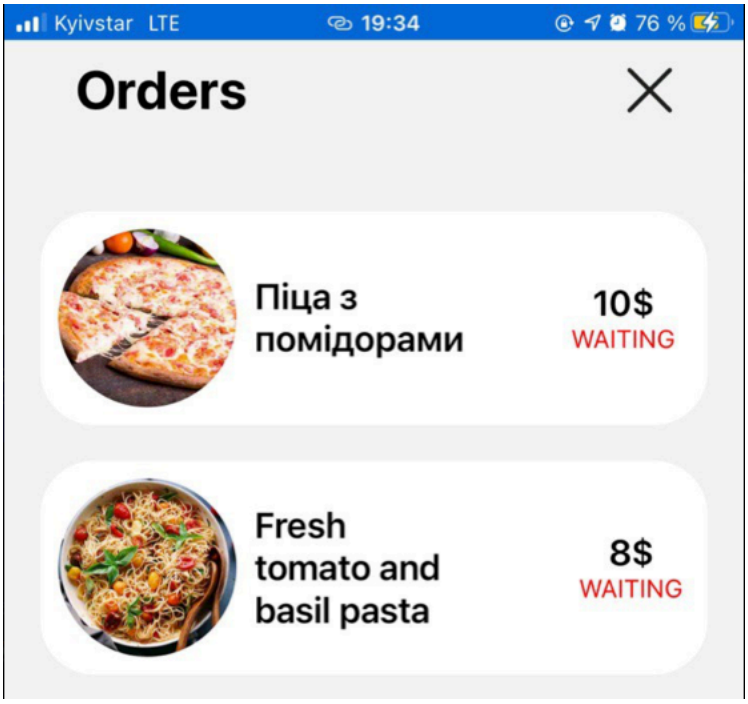

Рис 3.10. Інтерфейс сторінки замовлень, які знаходяться в процесі

Там можливо буде відслідкувати товар на карті, перевірити його статус, зв'язатися з рестораном або кур’єром (у випадку, коли товар доставляється таким).

Далі користувач може натиснути на пункт замовлення та отримати інформацію про комплект замовлення та етапи його виконання.

Всього в додатку відображаються 5 етапів. Протягом кожного з них користувач може взаємодіяти з різними функціями, які притаманні цьому етапу. Waiting, In progress, Delivery, OnTheSpot, Rate&Support ці етапи будуть відображати користувачу статуси виконання продукту що він замовив.

1. Waiting - остаточне підтвердження готовності продукту та готовності його до замовлення. Підтвердженням цього пункту ресторан дає згоду на обслуговування клієнта (рис. 3.11).

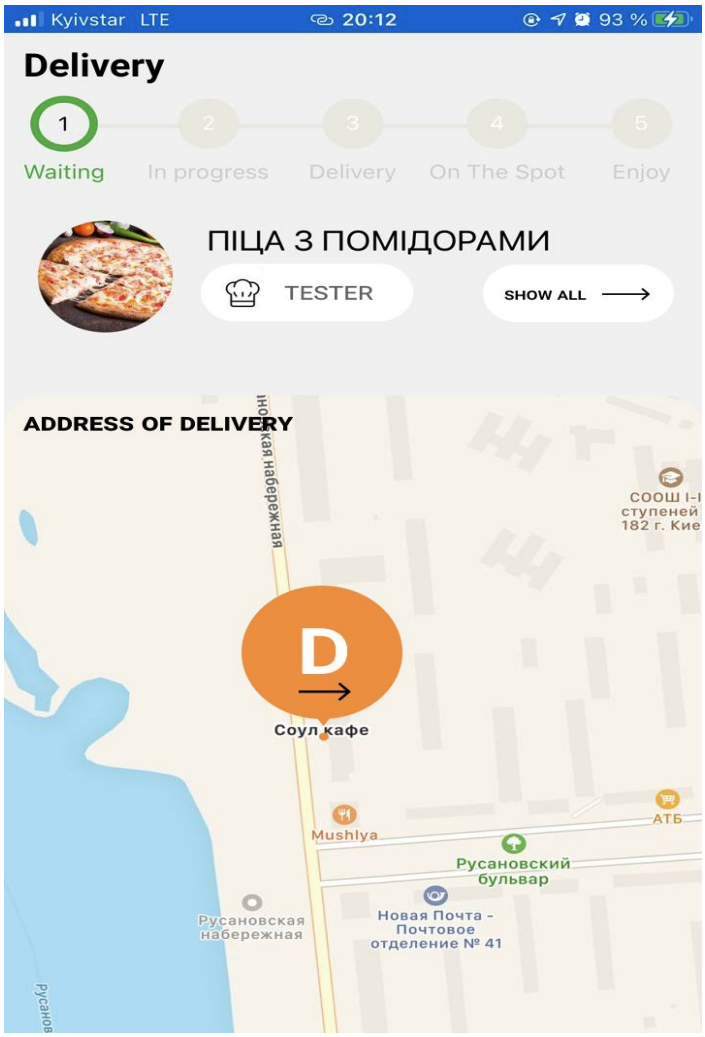

Рис. 3.11. Інтерфейс сторінки етапу підтверження замовлення

2. In progress - етап протягом якого товар готується та упаковується для доставки замовнику. Протягом цього етапу у користувача є можливість зв'язатися з продавцем або рестораном для надання йому уточнюючої інформації (рис. 3.12).

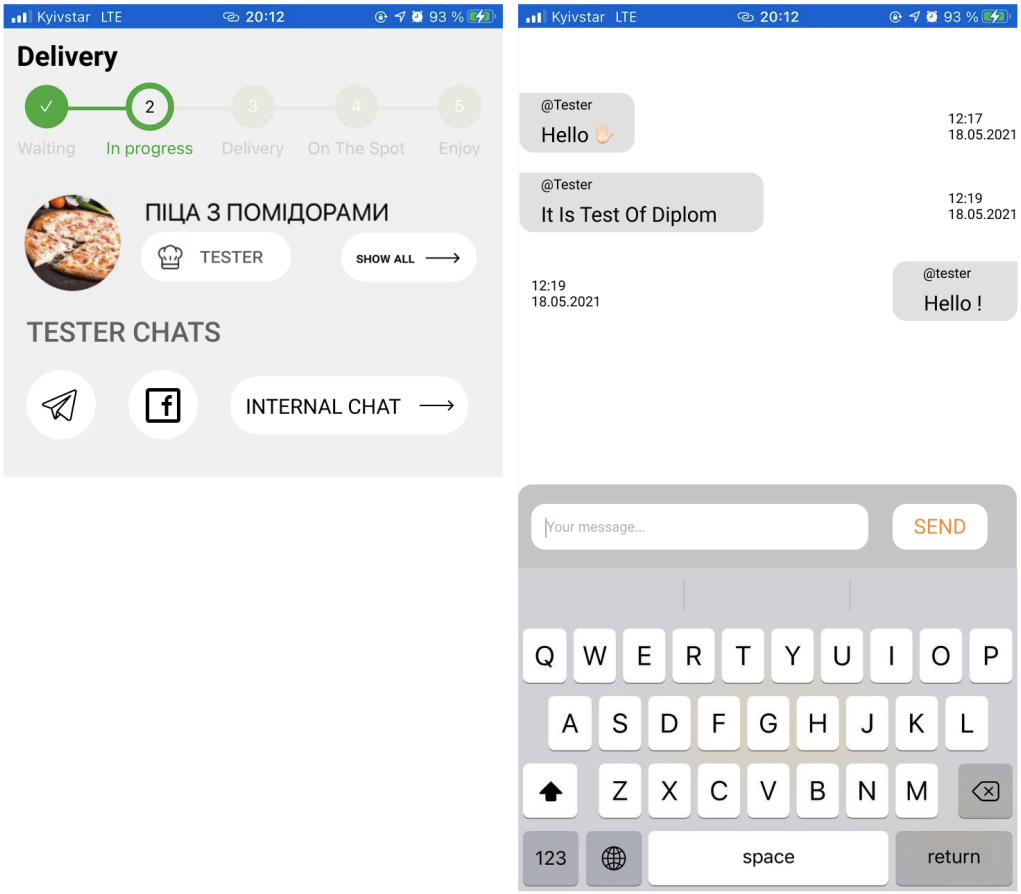

Рис. 3.12. Інтерфейс сторінки зв’язку з рестораном

3. Delivery - протягом цього етапу на екрані додатку відображається інформація про час доставки, місце, та інформація про товар. Також користувач має можливість відслідкувати кур'єра через інтерактивну карту, таким чином можна взнати наскільки близько він знаходиться та у випадку якихось проблем повідомити про це кур'єра який знаходиться на маршруті доставки.

4. On the spot - цей етап потрібен для реєстрування передачі замовлення замовнику та в той же час авторизації користувача. Такий метод перевірки введено для того, щоб кур'єр випадково не передав замовлення не тій особі. Для цього на сервері генерується чотиризначний код який відправляється в додаток замовника. Замовник повинен продиктувати цей пароль кур'єру, а він в свою чергу веде цей пароль в свій додаток. Після цього етапу процес доставки закінчується (рис. 3.13).

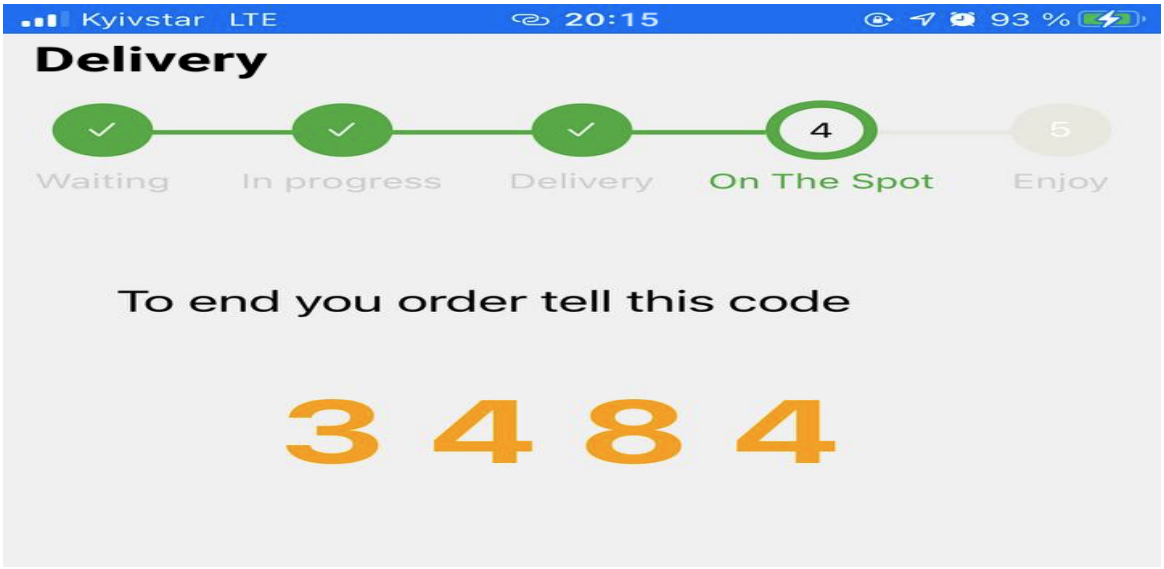

Рис. 3.13. Приклад роботи етапу підтверження передачі замовлення

5. Rate&Support - після виконання доставки, ініціюється процес оцінки та підтримки. Користувач (Замовник) може надати відгук про якість надання послуг та смакові якості замовлення яке було йому доставлено. Також у випадку проблем, користувач має можливість зв'язатися зі службою підтримки, яка зможе оперативно вирішити питання по даному замовленню.

3.4.Опис роботи програми кур’єра

Після встановлення та авторизації в системі кур’єра, система входить в режим очікування замовлення. Після реєстрації замовлення система перевіряє вільних кур’єрів та відправляє завдання до додатку. Виконувач доставки побачить модульне вікно, на якому буде відображатися початкова та кінцева адреса, а також дві кнопки: прийняти замовлення та відхилити. Даний процес відображений на рис. 3.14.

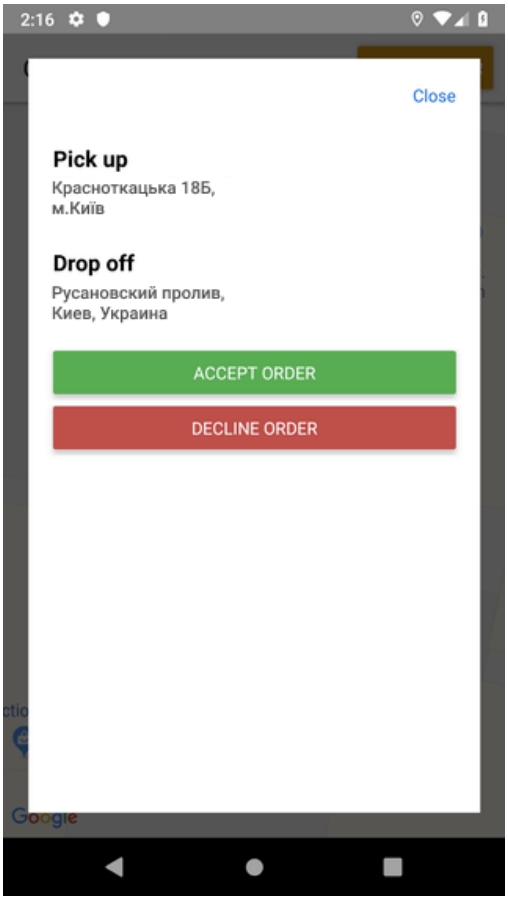

Рис. 3.14. Інтерфейс додатку кур’єра, на якому відображена адреса доставки

Як тільки водій погодиться, місце розташування ресторану та клієнта наноситься на карту разом із маршрутом між ними. Карта оновлюється в режимі реального часу, коли водій їде до місця призначення (рис. 3.15). Після того, як водій вибрав замовлення, нова кнопка буде накладена зверху на карту (рис. 3.16). Це дозволить водієві зв’язатися із замовником. Це буде корисно у випадках, коли адреса некоректна або у кур’єра виникли проблеми з доставкою. Далі у користувача буде можливість увійти в чат з замовником. На рис знаходиться приклад інтерфейсу чату на рис. 3.17. На етапі передачі страви, в момент коли замовник та кур’єр зустрілись, програма підійме модальне вікно, в якому знаходиться поле для введення коду верифікації. Код генерується на сервері та відправляється до замовника, той в свою чергу повинен продиктувати цей код водію.

Після вводу операція доставки закінчується і аккаунт кур’єра переходить в режим очікування (рис. 3.18).

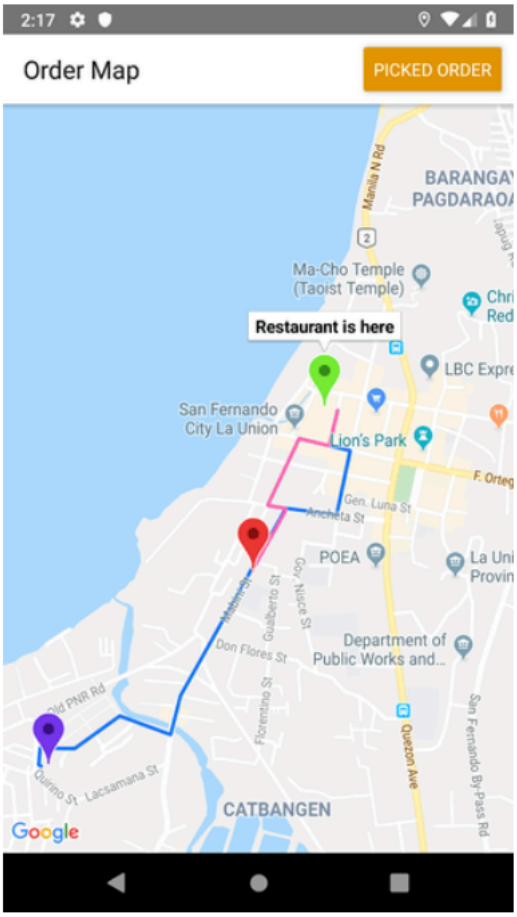

Рис. 3.15. Робота інтерактивної карти, побудова маршруту для кур’єра

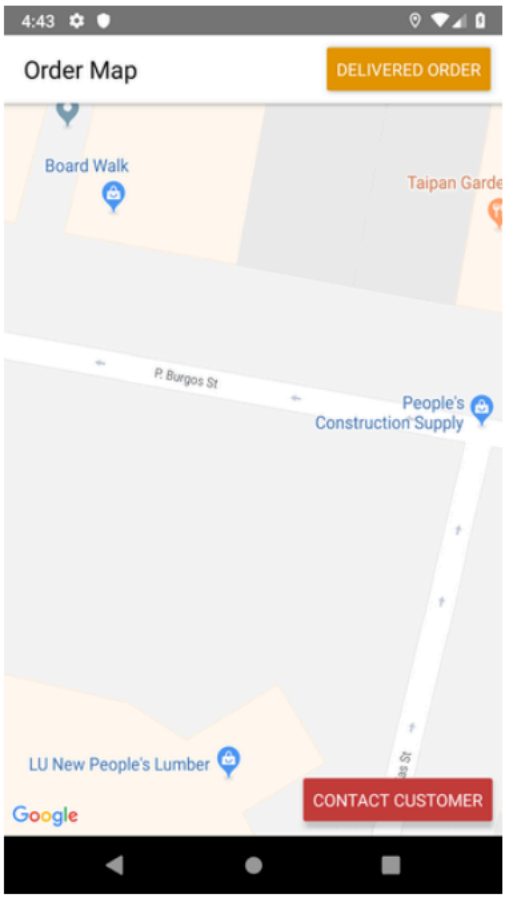

Рис. 3.16. Елемети інтефейсу додатку кур’єра

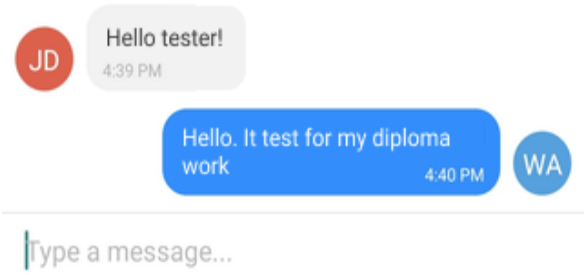

Рис. 3.17. Тестування роботи чату між замовником та кур’єром

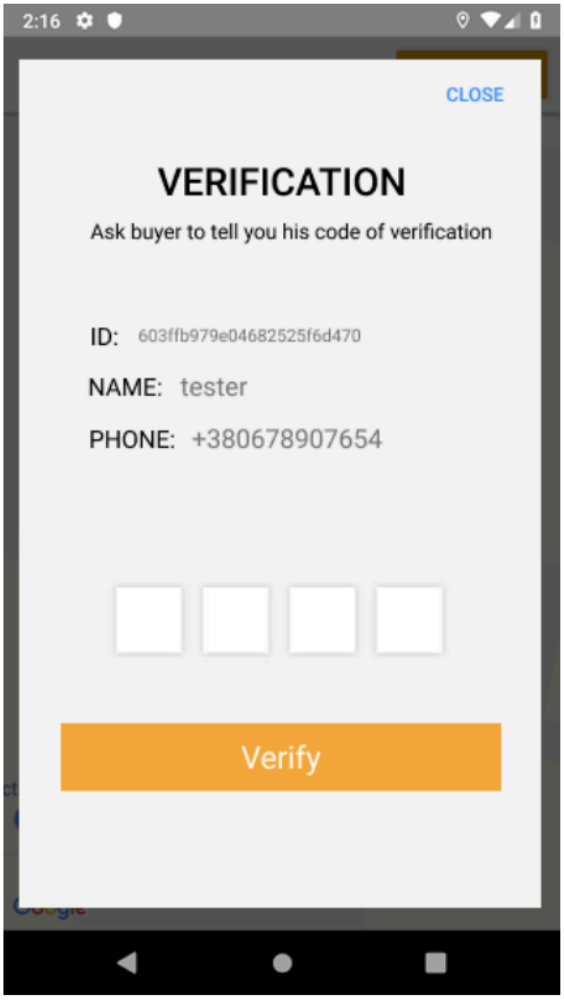

Рисунок 3.18. Інтерфейс сторінки підтверження передачі продукту харчування

ВИСНОВОК

Під час виконання роботи було розглянуто існуючі підходи та алгоритми реалізації сервісів логістичної доставки продуктів харчування покупцям. За час виконання було створено алгоритм автоматизованої доставки продукті з участю кур’єрів. В результаті на базі алгоритма та створеної за час написання архітектури було реалізовано три програмні комплекси.

Був створений

- додаток замовника;
- додаток кур’єра;
- серверний додаток для обробки API запитів від додатків.

Результатом кваліфікаційної роботи став проект, який може працювати з реальною аудиторією та з деякими вдосконаленнями приности реальний дохід.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adriana Nunez eMarketer, December 2020- On-demand delivery services power up across the globe –[Електронний ресурс] – Ресурс доступу: https://www.emarketer.com/content/on-demand-delivery-services-power-up-across-globe

2. Ashish Kumar - Ashish Kumar , A Study On The Impact Of Covid-19 On Home Delivery Of Food Items Through Food Delivery Platforms 17(12). ISSN 1567-214x – [Електронний ресурс] – Ресурс доступу: https://archives.palarch.nl/index.php/jae/article/download/7225/6889/14170

3. Що таке Node.js - Node.js – [Електронний ресурс]. – Режим доступу: https://nodejs.org/en/about/

4. Кількість використання ОС – [Електронний ресурс]. – Режим доступу: https://gs.statcounter.com/

5. MongoDB в действии, Бэнкер Кайл – [Електронний ресурс]. – Режим доступу:https://www.ozon.ru/product/mongodb-v-deystvii-8688130/

6. Вирооп Нарла, Майбутнє світового ринку послуг з доставки продуктів харчування онлайн– [Електронний ресурс]. – Режим доступу:https://store.frost.com/future-of-global-online-food-delivery-services-market-forecast-to-2025.html

7. Розуміння рідної архітектури React – [Електронний ресурс]. – Режим доступу: https://dev.to/goodpic/understanding-react-native-architecture-22hh

8. Різниця між мобільним сайтом та мобільним додатком – [Електронний ресурс]. – Режим доступу:https://webexpert.com.ua/ua/mobilnij-dodatok-i-mobilnij-sajt

9. Причини обрати React Native для розробки – [Електронний ресурс]. – Режим доступу:https://freexbcodes.com/rozvitok/7-prichin-vibrati-react-native-dlja-rozrobki/

10. Методи розробки – [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/36348/1/Schesterikov_bakalavr.pdf

Додаток А - приклад методів роботи додававння адреси користувача на сервері Node.js

```
router.put('/putAddress',verify, async(req, res) => {
   Buyer.findByIdAndUpdate({_id: req.user._id}, {
        $addToSet: {addressInfo: {
            id: uuidv1(),
            address: req.body.address,
            loc: req.body.loc
          }}

      },
      { new: true, useFindAndModify: false },
      async (err, userInfo) => {
        if(err) return res.status(400).send(err);
        res.send(userInfo.addressInfo)
      })

})
router.post('/deleteAddress' , verify, async (req, res) => {
   Buyer.findByIdAndUpdate({_id: req.user._id}, {
        $pull: {addressInfo: {
            id: req.body.id
          }}

      },
      { new: true, useFindAndModify: false },
      async (err, user) => {
        if(err) return res.status(400).send(err);
        res.send(user.addressInfo)
      })
})
module.exports = router;
```

Додаток Б - додавання банківської карти в кабінеті замовника

```
router.post('/addNewCardStrip', verify, async (req, res) => {
    const { email, product , authToken } = req.body;
    const { token } = authToken;
    const { card } = token;
    try {
        const customer = await stripe.customers.create({
            email: email,
            source: token.id
        })
        addCard(customer)
    }catch (err){
        console.log(err)
        res.send(err)
    }

    function addCard(customer){
        Buyer.findByIdAndUpdate(
            {_id: req.user._id},
            {$push: {payInfoStripe: {
                        customerInfoStripe: customer,
                        cardInfo: card,
                        moreInfoResponse: authToken}}},
            { new: true, useFindAndModify: false },
            async(errB, buyer) => {
                if (errB) return console.log(errB);
                res.send(buyer.cardInfo)
            })
    }

})
```

Додаток В - лістининг класа для обробки замовлень які знаходяться у статусі виконаних та у виконанні

```
class BuyerCompleteOrdersView extends Component {
    state={
        orders: [],
        setRefreshing: false,
        refreshing: false,
        lastRefresh: '',
        verifyCodes: ''
    }
    constructor(props) {
        super(props);
        this.state = {
            refreshing: false,
        };
        this.refreshScreen = this.refreshScreen.bind(this);
    }
    refreshScreen() {
        this.setState({lastRefresh: Date(Date.now()).toString()});
        this.setState({refreshing: true});
        this.fetchData().then(() => {
            setTimeout(() =>  this.setState({refreshing: false, lastRefresh: Date(Date.now()).toString()}),
1000)
        });
    }

    async componentDidMount() {
        this.refreshScreen()
        this.refreshing = React.useState(false);
        this.setRefreshing = React.useState(false);
    }
    componentWillMount() {
        this.refreshScreen()
    }
```

```
 getVerifiedCode = async(item) =>{
    const verifyCode = await getVerifiedCode(item._id)
    this.setState({
        verifyCodes : verifyCode
    })
}

async fetchData() {
    try{
        const orders = await getOrderedProduct()
        this.setState({
            orders: orders.data
        })
    } catch (err){
        console.log('err: ', err)
    }

}
render() {
    const {
        orders,
        refreshing,
        lastRefresh
    } = this.state
    if (typeof orders === 'undefined'){
        return null

} else if (orders.length !== 0){
        return(
            <SafeAreaView style={{flex: 1}}>

 <View style={{flexDirection: 'row', justifyContent: "space-between", alignItems: "center"}}>
                    <Text style={{marginHorizontal: 40, fontSize: 30, fontWeight: '700',
paddingVertical: 10}}>Orders</Text>

                    <TouchableOpacity style={{marginHorizontal: 40}}
                    onPress={() => this.props.navigation.navigate('BuyerMain')}>
```

```
                        <Image style={{width:25, height: 25}} source={letter_x}/>
                    </TouchableOpacity>
                </View>
                <ScrollView
                    showsVerticalScrollIndicator={false}
                    showsHorizontalScrollIndicator={false}
                    refreshControl={
                        <RefreshControl


                            refreshing={refreshing}
                            onRefresh={this.refreshScreen.bind(this)}>
                            {!!refreshing &&
                            <View style={{width: SIZES.width, flexDirection: "row", justifyContent:
"center", alignContent: "center"}}>
                                <Text style={{fontSize: 12, textTransform: "uppercase"}}>Last Refresh:
{moment(lastRefresh).startOf('minutes').fromNow()}</Text>
                            </View>
                            }


                        </RefreshControl>
                    }>


                    <FlatList data={orders}
                              renderItem={this.renderOrdersCard}
                              keyExtractor={(item) => `${item._id}`}
                              inverted={true}
                              contentContainerStyle={{
                                  paddingHorizontal: SIZES.padding * 2,
                                  paddingBottom: 30,
                                  paddingTop: 20,
                              }}/>
                    <View
                        style={{
                            color: COLORS.darkgray,
                            backgroundColor: COLORS.darkgray,
                            height: 1,
                            borderColor: COLORS.darkgray,
```

```
                                marginHorizontal: 50,
                                borderRadius: 10
                            }}
                        />
                    </ScrollView>
                </SafeAreaView>
            )
        }else {
            return(

<SafeAreaView style={{flex: 1, alignItems: "center", justifyContent: "center"}}>
                <View style={{flexDirection: "row" }}>
                    <Text style={{...FONTS.titleMainBuyerScreen}}>You don`t buy any products yet 🥺</Text>
                </View>

<View style={{flexDirection: "row", marginVertical: 50 }}>
                    <TouchableOpacity style={{backgroundColor: COLORS.white, borderRadius: 20}}
                    onPress={() => this.props.navigation.navigate('BuyerMain')}>
                        <Text style={{padding: 20, fontWeight: '600', fontSize: 16}}>To main page</Text>
                    </TouchableOpacity>
 </View>
            </SafeAreaView>
            )
        }


    }
    renderOrdersCard = ({item}) => {
        return <BuyerOrdersCard item={item} viewItem={this.viewItem} getVerifiedCode={this.getVerifiedCode}
codeVerify={this.state.verifyCodes}/>
    }
    viewItem = ({item, navigation}) => {
        this.props.navigation.navigate('BuyerAllInfoOrderedProductView', {
            item
        })
    }
}
export default BuyerCompleteOrdersView
```