Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Інтеграція та управління бізнес акаунтом Facebook та Instagram в мобільному додатку "Easy Accounting for Social Biz"

Виконав: студент VI курсу, групи СНм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Крушельницький В. О.
(підпис) (прізвище та ініціали)

Керівник _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Дуда О. М.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Загородна Н. В.
(підпис) (прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__25__» грудня 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Крушельницькому Владиславу Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інтеграція та управління бізнес акаунтом Facebook та Instagram в мобільному додатку "Easy Accounting for Social Biz"

Керівник роботи Боднарчук Ігор Орестович, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 24 » листопада 2023 року № 4/7-1099

2. Термін подання студентом завершеної роботи 22 грудня 2023р.

3. Вихідні дані до роботи Офіційна документація Facebook

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Загальний Розділ.
2 Розробка технічного та робочого проєкту.
3 Спеціальний розділ.
4 Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Перелік посилань.
Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 Титульна сторінка. 2 Актуальність дослідження. 3 Технічне завдання.
4 Дані, які потрибно отримати з Facebook API.
5 Методи, реалізовані в клієнтській частині.
6 Методи, реалізовані в серверній частині.
7 Схема передачі повідомлень. 8 Висновки. 9 Дякую за увагу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 24 листопада 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	25.11.2023	*Виконано*
2.	Підбір наукових джерел про інтеграцію Facebook API	26.11.2023-28.11.2023	*Виконано*
	в мобільний додаток		
3.	Опрацювання наукових публікацій та збір даних по	29.11.2023-1.12.2023	*Виконано*
	темі роботи		
4.	Виконання дослідження згідно мети кваліфікаційної	2.12.2023-4.12.2023	*Виконано*
	роботи		
5.	Оформлення розділу «Загальний розділ»	5.12.2023-7.12.2023	*Виконано*
6.	Оформлення розділу «Розробка технічного та	8.12.2023-10.12.2023	*Виконано*
	робочого проєкту»		
7.	Оформлення розділу «Економічний розділ»	11.12.2023-13.12.2023	*Виконано*
8.	Виконання завдання до підрозділу «Охорона праці»	14.12.2023-15.12.2023	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	16.12.2023-17.12.2023	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	18.12.2023-19.12.2023	*Виконано*
11.	Нормоконтроль	19.12.2023-20.12.2023	*Виконано*
12.	Перевірка на плагіат	22.12.2023	*Виконано*
13.	Попередній захист кваліфікаційної роботи	22.12.2023	*Виконано*
14.	Захист кваліфікаційної роботи	26.12.2023	

Студент _____ Крушельницький В. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчук І. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Інтеграція та управління бізнес акаунтом Facebook та Instagram в мобільному додатку “Easy Accounting for Social Biz”// Кваліфікаційна робота освітнього рівня «Магістр» // Крушельницький Владислав Олегович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНм-61 // Тернопіль, 2023 // С. 83, рис. – 21, табл. – 2, кресл. – 0, додат. – 6, бібліогр. – 31.

Ключові слова: інтеграція, сервер, клієнт, розробка, повідомлення, facebook, instagram.

Кваліфікаційна робота присв’ячена розробці інтегріції бізнес аккаунту Facebook та Instagram в мобільний додаток для подальшого управління ним, що б спрощувало процес ведення бізнесу у соціальних мережах.

В першому розділі кваліфікаційної роботи описуються аналітичний огляд існуючих рішень в конкурентних програмах та аналіз технічного завдання проекту. Також перший розділ містить інформацію про область застосування, технічні показники та стадії етапи розробки.

У другому розділі представлено процес взаємодії з facebook API, тестування та налагодження програми.

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині записки.

Основні питання охорони праці та техніки безпеки розглянуто в четвертому розділі.

ANNOTATION

Integration and management of Facebbok and Instagram business accounts in the mobile application “Easy Accounting For Social Biz” // The educational level "Master" qualification work // Krushelnytkyi Vladyslav Olegovich // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNnm-61 group // Ternopil, 2023 // P. 83, fig. - 1, tables - 2, posters - 0, annexes - 6, ref. - 31.

Key words: integration, server, client, development, message, facebook, instagram.

The qualification work is dedicated to the development of the integration of Facebook and Instagram business accounts into the mobile application for further management of it, which would simplify the process of conducting business in social networks.

In the first section of the qualification work, an analytical review of existing solutions in competitive programs and an analysis of the technical terms of the project are described. Also, the first section contains information about the scope of application, technical indicators and stages of development.

The second section presents the process of interacting with the facebook API, testing and debugging the program.

The calculation of the cost of development and economic efficiency is given in the economic part of the memo.

The main issues of occupational health and safety are discussed in the fourth chapter.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

НДР (Науково-дослідна робота) – робота пошукового, теоритичного та експериментального характеру.

API (англ. Application programing interface) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

HTTP (англ. HyperText Transfer Protocol) – мережевий протокол прикладного рівня.

JSON (англ. JavaScript Object Notation) – це текстовий формат обміну даними між комп’ютерами.

SDK (англ. software Development Kit) – набір із засобів розробки, утиліт і документації.

SQL (англ. Structured query language) – декларативна мова програмування для взаємодії з базами даних.

UML (англ. Unified Modeling Language) – це мова графічного опису для об’єктного моделювання.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі використання мобільних пристроїв стає не тільки популярним, але й невід'ємною складовою бізнесу. Мобільні додатки стають важливим інструментом для зручної та ефективної роботи, навіть в дорозі. Сьогодні однією з найбільш потрібних інструментів для підприємців є облікова система, яка дозволяє ефективно відстежувати фінансову діяльність свого бізнесу.

У цифровій епосі бізнеси також відчувають необхідність в активному використанні соціальних мереж для залучення клієнтів, підвищення уваги до свого бренду та підтримання взаємодії з аудиторією. Facebook та Instagram, належать до найпопулярніших платформ у цьому відношенні, і для ефективного управління бізнесом на цих платформах важливо мати бізнес-акаунти.

Мета і задачі дослідження. Дана робота присвячена опису інтеграції Facebook API в мобільний застосунок "Easy Accounting for Social Biz", який допоможе підприємцям здійснювати облік фінансових операцій свого бізнесу, використовуючи мобільний пристрій. В процесі розробки додатку будуть розглянуті такі завдання, як інтеграція Facebook API та Instagram у мобільний додаток, можливість отримання товарів з бізнес-сторінок у додаток, а також підключення месенджера для відправлення повідомлень від імені бізнес-сторінки, безпосередньо з мобільного додатку.

Об’єктом дослідження даної роботи виступає Facebook API, його доступ і можливості, які підходять для нашого додатку.

Предметом дослідження є класи та методи, які реалізовані як в серверній так і в клієнтській частині додатку, для взаємодії з Facebook API.

Наукова новизна даного проекту полягає в тому, що аналогів із таким самим функціоналом на ринку немає.

Практичне значення одержаних результатів. Розроблений додаток, який повноцінно взаємодіє із бізнес сторінками Facebook та Instagram.

Апробація результатів магістерської роботи. Основні результати проведених досліджень обговорювались на XI міжнародної студентської

науково-технічної конференції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя (м. Тернопіль, 2023 р.).

Публікації. Основні результати кваліфікаційної роботи опубліковано у двох працях конференції (Див. додатки А - Г).

Структура й обсяг кваліфікаційної роботи. Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку літератури з 31 найменувань та 6 додатків. Загальний обсяг кваліфікаційної роботи складає 83 сторінки, з них 59 сторінки основного тексту, який містить 21 рисунок та 2 таблиць.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Огляд програми “Easy Accounting for Social Biz”

“Easy Accounting for Social Biz" - це мобільний додаток, розроблений для спрощення ведення бізнесу у соціальній мережі Instagram. Додаток надає ряд корисних функцій, які допомагають власникам бізнесу ефективно керувати замовленнями, продуктами, клієнтами та іншими аспектами їхнього бізнесу.

Деякі з ключових функцій включають:

- Керування замовленнями;
- управління продуктами;
- керування клієнтами;
- імпорт продуктів та клієнтів ззовні;
- створення календарних нагадувань;
- створення електронного чеку;
- переписка з клієнтом через чат;
- зміна власних користувацьких налаштувань.

Додаток дозволяє створювати, редагувати та видаляти замовлення. Можна легко оновлювати статус замовлення, відстежувати виконання та повідомляти клієнтам про стан їхніх замовлень. Керування замовленнями також включає перегляд вже завершених замовлень, тобто так звана “історія” замовлень.

"Easy Accounting for Social Biz" дозволяє створювати та керувати списком продуктів, які продаються. Є можливість додавати нові продукти до замовлення, оновлювати інформацію про наявність, ціни та описи продуктів. Управління продуктів також включає можливість імпортувати інформацію про продукти зі сторінки у соціальній мережі.

Додаток дозволяє створювати та зберігати профілі клієнтів. Можна додавати нових клієнтів, зберігати контактну інформацію таку як номер телефону, адресу, ПІБ, а також відстежувати історію їхніх замовлень.

Додаток забезпечує можливість імпортування клієнтів та продуктів із зовнішній джерел, таких як соціальні мережі. Необхідно реалізувати

автозоповнення необхідних даних про клієнта та наявні продукти, аби не довелося виконувати заповнення вручну.

"Easy Accounting for Social Biz" надає змогу створити нагадування, яке буде збережено у локальний календар на мобільному пристрої. Це допоможе краще організовувати робочий графік, не пропускаючи важливі дати, зустрічі та інші події.

Додаток забезпечує формування електронного чеку замовлення, аби можна було надіслати його у месенджері своїм клієнтам.

Додатку надає можливість користувачеві змінювати персональні налаштування, що допоможе налаштувати додаток під власні вподобання. Особливо важливими налаштуваннями є колірна схема та мова.

Побудуємо UML-діаграму варіантів використання мобільного додатку "Easy Accounting for Social Biz", що зображена на рисунку 1.1.

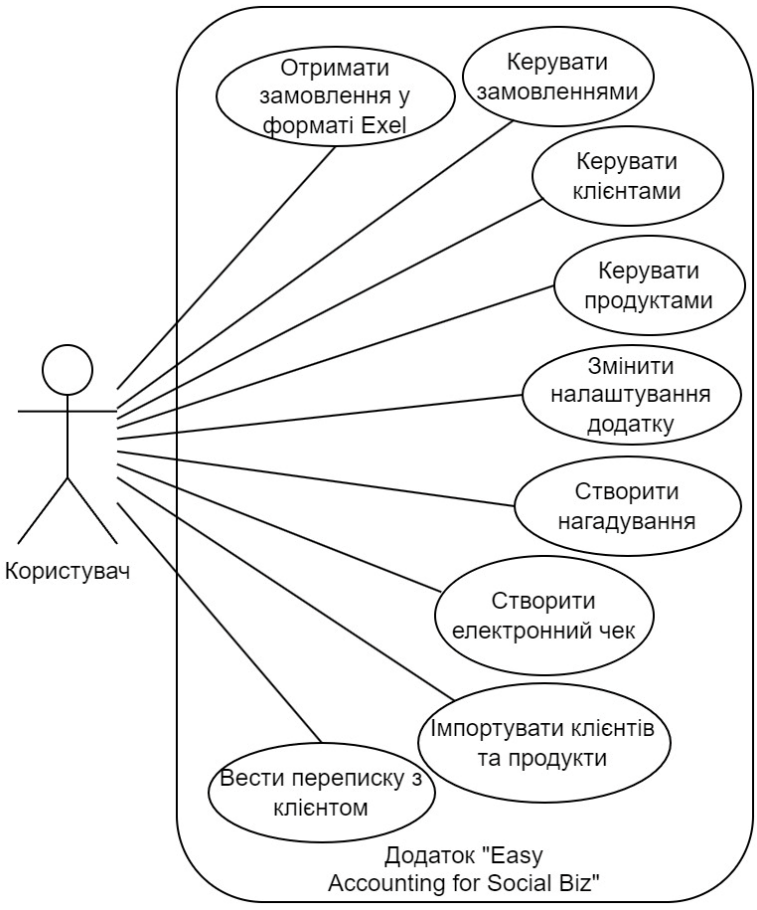

Рисунок 1.1 – UML-діаграма варіантів використання додатку “ Easy Accounting for Social Biz ”

1.2 Огляд існуючих рішень

Перед тим як втілювати будь-яку бізнес ідею, потрібно вивчити ринок, а також можливих конкурентів. Головною задачею мобільного застосунку “Easy Accounting for Social Biz” – є введення бізнесу з телефону з можливістю підключення біснес акаунтів Facebook та Instagram і управління ними з одного джерела. Для вивчення ринку конкурентноспроможних програм, було вибрано крамницю застосунків від Google, а саме – PlayMarket. Це сама популярна крамниця для скачування як безплатних, так і навпаки мобільних застосунків для телефонів з операційною системою Android.

На просторах крамниці ми можемо знайти лише 2 мобільні додатки, які за своїм функціоналом подібні до “Easy Accounting for Social Biz”. Це: “Meta Business Suite” і “WhatsApp Business”.

1.2.1 Огляд програми Meta Business Suite

Meta Business Suite - це інструмент, розроблений компанією Meta Platforms, Inc. (раніше відомою як Facebook), який спрямований на полегшення керування бізнес-акаунтами на платформах Facebook і Instagram. Цей інструмент важливий для підприємств, які прагнуть підвищити ефективність своєї соціальної медіа стратегії та спростити процес взаємодії з аудиторією в цих соцмережах.

Переваги:

1. Об'єднання Facebook і Instagram: Однією з основних переваг Meta Business Suite є можливість керувати як бізнес-акаунтом на Facebook, так і на Instagram, одночасно і з одного інтерфейсу. Це значно спрощує процес розміщення контенту та взаємодії з аудиторією на обох платформах.

2. Планування та автоматизація: Додаток дозволяє планувати публікації, що допомагає бізнесам підтримувати активну та регулярну присутність в соцмережах. Ви можете створювати контент заздалегідь та

налаштовувати час публікації, що робить процес управління соцмережами більш ефективним.

3. Аналітика та відстеження: Важливим аспектом додатку є можливість аналізувати статистику та взаємодію аудиторії. Meta Business Suite надає користувачам доступ до розширеної аналітики, що допомагає відстежувати ефективність рекламних кампаній та контенту. Це дозволяє бізнесам адаптувати свою стратегію для досягнення кращих результатів.

4. Спрощення рекламних кампаній: Meta Business Suite дозволяє створювати та керувати рекламними кампаніями на Facebook і Instagram. Користувачі можуть визначити цільову аудиторію, встановити бюджет, а також відстежувати результати реклами безпосередньо в додатку.

5. Зручність та доступність: Додаток доступний як для мобільних пристроїв, так і для веб-версії, що робить його доступним для користувачів у різних ситуаціях. Ця гнучкість використання дозволяє бізнесам ефективно керувати своєю діяльністю, навіть якщо вони перебувають в дорозі. Частина інтерфейсу додатку показана на рисунку 1.2[16].

Функціонал цього мобільного застосунку, дозволяє нам отримати доступ до бізнес акаунтів Facebook та Instagram, відправляти повідомлення з додатку на різні платформи від імені бізнес акаунту, створювати та редагувати пости, дивитись статистику, запускати рекламу тощо. Проте, це трішки відрізняється від основної ідеї “Easy Accounting for Social Biz”. Тому що Meta Business Suite – це більше як програма- менежер, основна ідея якої – надати простий інструмент для керуванням вже існуючого функціоналу, в той час як “Easy Accounting for Social Biz” – пропонує зручний функціонал саме для введення бізнесу, через такі інструменти як створення додаткових товарів, введення обліку товарів, створення замовлень тощо.

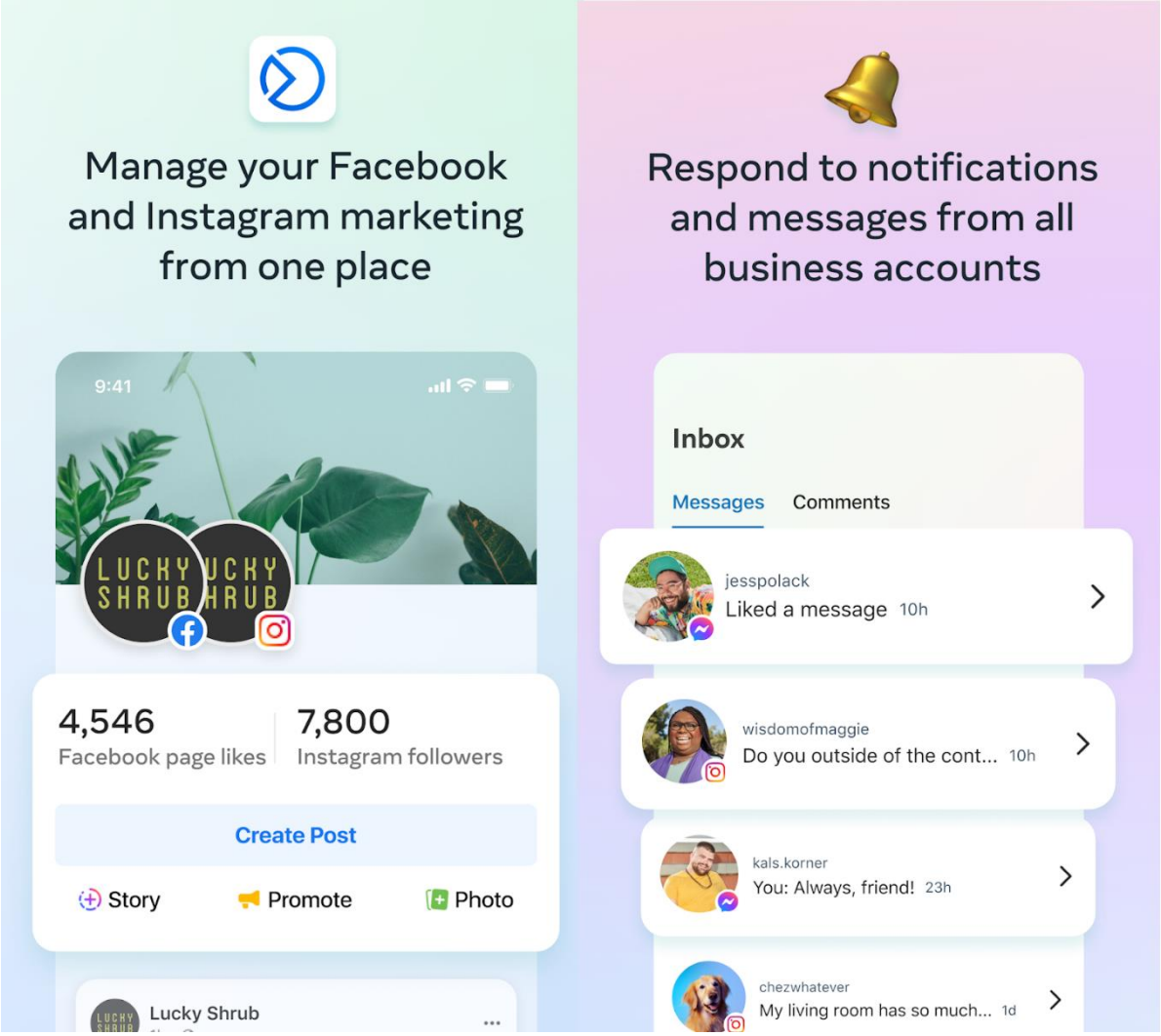

Рисунок 1.2 - Інтерфейс мобільного застосунку “Meta Business Suite”

1.2.2 Огляд програми WhatsApp Business

WhatsApp Business - це окремий додаток, створений спеціально для користування бізнесами з метою поліпшення комунікації з клієнтами та партнерами через популярну месенджер-платформу WhatsApp. Цей додаток надає широкий спектр можливостей, які допомагають бізнесам залучати клієнтів, збільшувати продажі та покращувати обслуговування клієнтів. Ось детальний огляд можливостей WhatsApp Business:

1. Професійний бізнес-профіль: WhatsApp Business дозволяє бізнесам створювати спеціальні профілі, де можна внести інформацію про компанію, таку як назва бізнесу, фотографія логотипу, адреса, години роботи, контактна

інформація та посилання на веб-сайт. Частина інтерфейсу додатку показана на рисунку 1.3[17].

2. Автоматизовані вітання: Бізнеси можуть налаштувати автоматичні вітання для відвідувачів, що надсилають повідомлення вперше або в позаурочний час. Це допомагає покращити перший контакт з клієнтами.

3. Авто-відповіді: За допомогою цієї функції, бізнеси можуть створити автоматизовані відповіді на питання, які часто задаються. Це значно зекономлює час та допоможе клієнтам отримати швидку відповідь.

4. Мітки для повідомлень: Для легкості організації комунікації з клієнтами, можна додавати мітки до повідомлень, щоб позначити їх за темами або статусами (наприклад, "запит на інформацію", "замовлення в обробці").

5. Аналітика та статистика: WhatsApp Business надає статистичні дані, які дозволяють бізнесам відстежувати, скільки повідомлень доставлено, прочитано та відповідей. Це допомагає оцінювати ефективність комунікації.

6. Повідомлення для клієнтів: Бізнеси можуть надсилати інформаційні повідомлення клієнтам, такі як оголошення, новини про продукти або спеціальні пропозиції.

7. Групові чати: Додаток дозволяє створювати групові чати для обговорення проектів або подій зі співробітниками та клієнтами.

8. Безпека та шифрування: WhatsApp Business використовує криптографічне шифрування для захисту конфіденційності даних та повідомлень.

9. Багатомовність: Додаток підтримує багатомовну комунікацію, що корисно для бізнесів з міжнародними клієнтами.

10. Безкоштовність: WhatsApp Business є безкоштовним для завантаження та використання, що робить його доступним для різних типів бізнесів.

Функціонал цієї програми більше нагадує бізнес сторінку Facebook. З її основних недоліків можна відмітити: це обмеження в платформах, оскільки WhatsApp Business надає послуги для керування бізнею сторінкою виключно в месенджері WhatsApp, а також ряд недоліків присутніх і у Meta Business Suite.

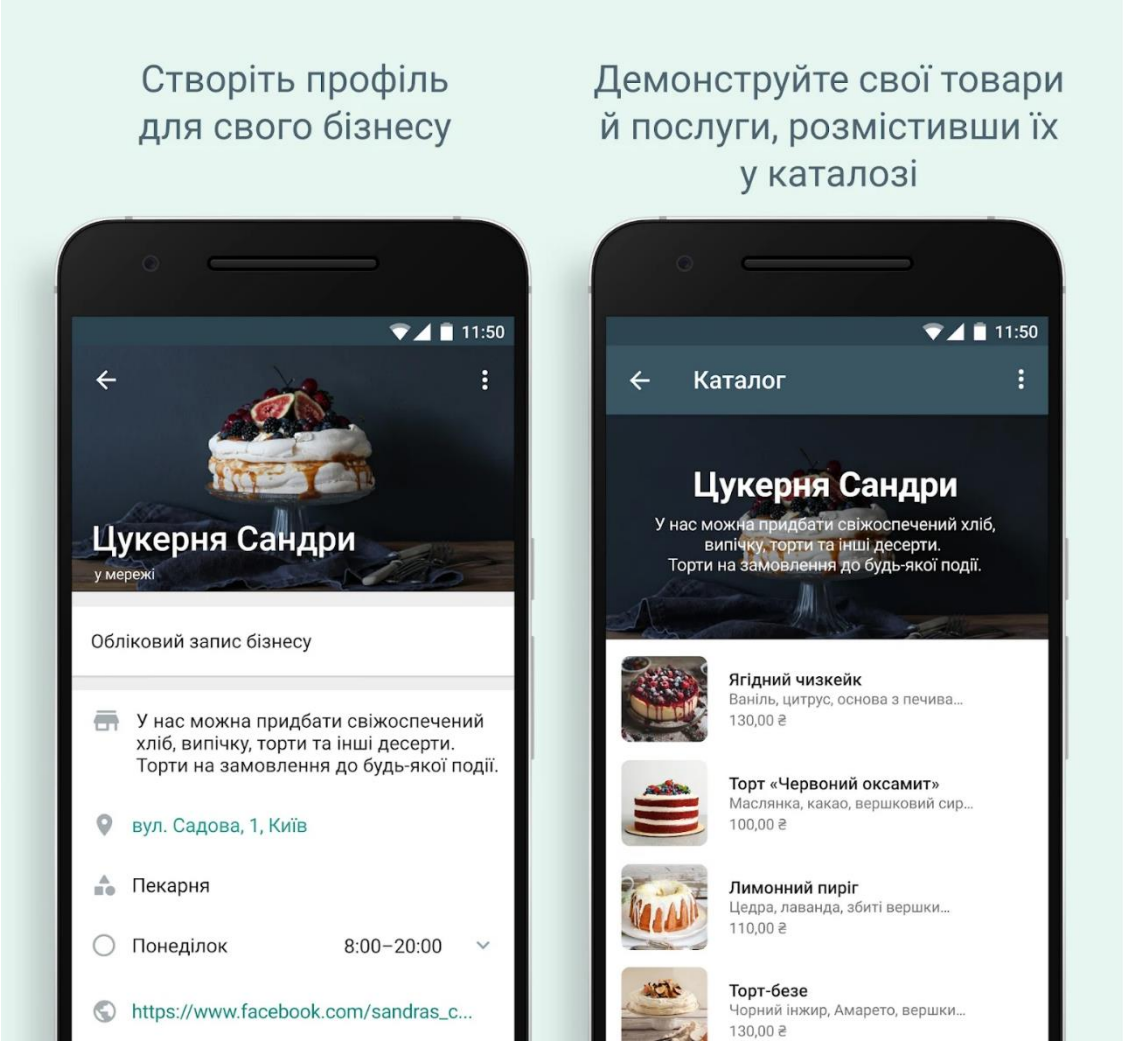

Рисунок 1.3 - Інтерфейс мобільного застосунку “WhatsApp Business”

Звідси можна зробити висновок, що “Easy Accounting for Social Biz” – є унікальною в своєму роді програмою, яка має необхідний і оригінальний функціонал.

1.3 Технічне завдання

1.3.1 Найменування та область застосування

Повне найменування проекту – інтеграція та управління бізнес акаунтом Facebook та Instagram в мобільному додатку "Easy Accounting for Social Biz".

Стисла характеристика області застосування програмного виробу та об’єкта, в якому передбачається його використання: інтеграція бізнес акаунту

Facebook та Instagram в мобільному додатку "Eazy Accounting for Social Biz" для власників бізнесу з метою полегшення менеджменту у соціальних мережах.

Інтеграція бізнес акаунтів в "Easy Accounting for Social Biz" дозволить власникам бізнесу:

1. Створювати та редагувати продукти;
2. створювати та редагувати клієнтів;
3. прив’язати продукт до посту в “Instagram”;
4. прив’язати продукт до сторінки в “Instagram”;
5. налагодити передачу повідомлень з клієнтом.

1.3.2 Призначення розробки

Експлуатаційне призначення інтеграції біснес акаунтів в мобільний додаток “Eazy Accounting for Social Biz” полягає в забезпеченні доступу до біснес акаунтів Facebook та Instagram користувача, їх постів, а також взаємодія із месенджером бізнес-сторінки безпосередньо з мобільного застосунку.

Серверна частина має бути лаконічно організована для ефективного “спілкування” з клієнтською частиною. Наявність зручного та зрозумілого API є однією з найважливіших складових якісного беку, тому серверна частина мобільного додатку “Eazy Accounting for Social Biz” повинна бути інтуїтивно зрозумілою для використання та експлуатації.

Для того щоб додаток зміг повноцінно співпрацювати з Facebook API функціонал взаємодії потрібно буде реалізовувати як на серверній, так і на клієнтській частині додатку.

Функціональне призначення: для досягнення вищеописаного функціоналу будуть застосовані наступні програмні рішення:

- Hasura GraphQL Engine;
- PostgreSQL;
- NodeJS;
- Flutter;
- FacebookSDK;

- webhook.

Hasura GraphQL Engine буде відповідальний за операції отримання всіх повідомлень з переписок із замовниками.

Вищеописані повідомлення зберігатимуться у реляційній базі даних PostgreSQL.

NodeJS сервер та Webhook допоможуть реалізувати механізм передачі повідомлень у реальному часі з мінімальними затримками.

Flutter – фреймворк на якому написаний мобільний додаток і у якому буде реалізована решта логіки по взаємодії з Facebook API і бізнес-акаунтами.

FacebookSDK – сервіс наданий Facebook, який полегшить взаємодію з Facebook API, надавши готові макети запитів.

1.3.3 Вимоги до програмного забезпечення

Вхідні дані.

Серверна частина мобільного додатку “Eazy Accounting for Social Biz” прийматиме вхідні дані у вигляді запитів. Оскільки для реалізації усього функціоналу використовуватиметься Hasura GraphQL Engine, то запити та вхідні дані будуть відрізнятися.

Запити до Hasura виглядають як звичайний HTTP-запит, але з тією відмінністю, що тіло запиту не містить громісткий JSON-об’єкт з усіма параметрами та аргументами. До сервера надсилається лише стрічка з запитом на мові запитів GraphQL. Періодичність надходження залежить від інтенсивності використання функціоналу та операцій, що здійснює користувач, або ж клієнт бізнес-акаунту.

Клієнтська частина мобільного додатку “Eazy Accounting for Social Biz” прийматиме вхідні дані від Facebook API у відповідь на запити до цього сервісу. Для ефективнішої взаємодії із Facebook API буде використовуватись інструмент для роботи з Facebook для Flutter, а саме FacebookSDK. Вихідна інформація від Facebook API може відрізнятись, в залежності від запиту. Тому всю необхідну інформацію ми будемо зберігати в локальній базі данних – Hive. Її функціонал

являє собою зберігання данних за системою: назва боксу > ключ боксу. Таким чином ми зможемо розділити необхідні нам данні і оптимізувати доступ до них. Як і у Hasura періодичність надходження залежить від кількості операцій пов’язаних з обліковим записом Facebook з нашого мобільного додатку.

Вихідна інформація.

Відповідь серверної частини мобільного додатку "Eazy Accounting for Social Biz" яка буду містити інформацію про переписку із клієнтом буде у об’єкті формату JSON, що перетворюватиметься у відповідну програмну модель.

Вихідна інформація, пов’язана із данними з Facebook API, з сторони клієнтської частини буде у вигляді модальних вікон та сповіщень з інформацією для користувача. Після додавання, оновлення або видалення також буде відображатись віконечко з простими та зрозумілими повідомленням про успішність або помилку виконуваної операції.

Опис функцій та обмежень.

Для початку розглянемо процес роботи серверної частини, що обробляється Hasura GraphQL Engine. Спочатку потрібно визначити схему GraphQL, яка описує доступні типи даних та запити. Схема визначає поля, які можна запитувати, а також способи отримання даних. Після визначення схеми можна створити запит, який відповідає необхідним потребам. Запит до сервера вказується мовою запитів GraphQL, що нагадує скрипт до нереляційних баз даних на кшталт MongoDB. Декларативний підхід GraphQL сприяє зручному опису, які дані потрібно отримати, а не як саме їх отримати.

Після створення запиту потрібно надіслати його на сервер GraphQL Engine. Зазвичай використовуються HTTP-запити, такі як POST або GET. Сервер GraphQL Engine отримує запит та обробляє його згідно з визначеною схемою GraphQL. Він перетворює стрічку зі скриптом на мові GraphQL у звичайний SQL запит та звертається до бази даних або іншого джерела даних, отримує результат та формує відповідь.

Після обробки сервер GraphQL Engine повертає відповідь на клієнтську частину у вигляді об’єкту json. Відповідь може містити дані, які запитувались, або повідомлення про помилку, якщо запит був недійсним або виникла проблема

під час обробки. Після отримання відповіді можна обробити дані згідно з потребами. Відповідно є можливість використовувати отримані дані для відображення на користувацькому інтерфейсі або подальшої обробки.

Використовувати Hasura GraphQL Engine ми будемо тільки для того, щоб зберігати данні про переписку користувача із клієнтами. Вхідні дані будуть однакові із вихідними(id клієнта, id користувача, дата повідомлення, текст повідомлення), оскільки немає причин зберігати лишню інформацію.

Функції клієнтської частини можна розділити на дві категорії: функції до сервісу Hasura GraphQL Engine, які були вище описані і функції безпосередньої взаємодії із Facebook API. Функції другої категорії будуть здійснені за допомогою сервісу FacebookSDK, який значно спростить задачу, надавши готовий макет запиту. До другої категорії можна віднести такі функції:

- Керування продуктами. Вхідні дані: деталі продукту (назва, опис, ціна, кількість). Вихідні дані: створений, оновлений або видалений продукт. Обмеження: користувач повинен мати права на додавання, редагування та видалення продуктів. Точність обчислень не застосовується до цієї функції.
- Керування клієнтами. Вхідні дані: контакта інформаціія клієнта (ім'я, прізвище, адреса, номер телефону, назва облікового запису в Instagram). Вихідні дані: створений, оновлений або видалений клієнт. Обмеження: користувач повинен мати права на додавання, редагування та видалення клієнтів. Точність обчислень не застосовується до цієї функції. Реалізація бізнес-логіки відбувається аналогічним чином до продуктів.

Вимоги до надійності.

Серверна частина має бути однакова надійна як і решта всієї backend логіки. А це:

- Висока доступність: Серверна частина повинна бути доступною в будь-який час, забезпечуючи безперебійну роботу додатку. Для цього можуть використовуватися механізми автоматичного масштабування та резервного копіювання.

- Захист даних: Забезпечення безпеки та конфіденційності даних є ключовим аспектом надійності. Серверна частина повинна використовувати надійні механізми шифрування та захисту даних.
- Обробка помилок: Серверна частина повинна бути стійкою до помилок та вміти відновлюватися після непередбачених ситуацій. Обробка помилок повинна бути належно налаштована, а несправності повинні відображатися в зрозумілому для розробників та адміністраторів форматі. відповідних заходів для вирішення несправностей.
- Захист від атак: Серверна частина повинна бути захищена від різних видів атак, таких як SQL ін’єкції, перетин сайтів, переповнення буферу тощо. Застосування належних заходів безпеки, таких як валідація введення, параметризовані запити до бази даних та захист від вразливостей, є важливими для забезпечення надійності сервера.

Клієнстька частина має бути написана у стилі, який не відрізняється від стилю написання всього іншого коду у додатку. Ці правила можна описати так:

- Код повинен мати чітку ієрархічну структуру, що дозволяє керувати роботою системи на різних рівнях. Це допомагає виявляти та усувати збійні ситуації, забезпечує логічну організацію функцій та полегшує розширення та підтримку додатку.
- Код повинен мати блоки, які контролюють та усувають збійні ситуації в роботі системи. Наприклад, можуть бути блоки, які перевіряють наявність та цілісність даних.
- Код повинен запезпечувати механізми контролю введених даних, щоб забезпечити стійкість програми до помилок користувача. Наприклад, можуть бути встановлені правила контролю по діапазону значень даних, перевірка формату введеної інформації або валідація коректності введених даних.
- Код повиненнадавати можливості збереження та використання даних під час відновлення роботи після аварійного переривання. Це може включати автоматичне збереження проміжних результатів, стану програми або резервне копіювання даних.

- Код повинен вміти відповідати на виняткові ситуації, наприклад, якщо запис не знайдено або виникла помилка під час обробки даних. Він повинен вміти коректно повідомляти про такі ситуації та реагувати належним чином.

Умови експлуатації.

Умови експлуатації серверної частини будуть відповідати за наступні параметри:

Доступ до сервера: інструкції для адміністраторів щодо доступу до серверної частини, включаючи адресу сервера, порти, ідентифікаційні дані та інші необхідні дані для забезпечення з'єднання.

Мережеві вимоги: вказівки щодо налаштування мережевого середовища, включаючи протоколи комунікації, безпеку мережі (наприклад, використання SSL / TLS), фаєрволи та налаштування маршрутизаторів.

Конфігурація сервера: інструкції щодо конфігурації сервера, включаючи налаштування веб-сервера, бази даних та інших необхідних компонентів.

Захист даних: політика безпеки та конфіденційності, яка описує заходи, прийняті для захисту даних, включаючи шифрування, контроль доступу та механізми резервного копіювання.

Резервне копіювання: вказівки щодо регулярного резервного копіювання серверної частини та бази даних, а також процедури відновлення даних в разі аварійного випадку або втрати даних.

Моніторинг та логування: опис процесу моніторингу серверної частини, включаючи важливі метрики працездатності, спостереження за помилками та інші сигнали здоров'я системи.

Клієнтська частина відповідатиме наступним умовам експлуатації:

- Режим взаємодії користувача з програмним комплексом.

Забезпечує користувачам зручний та інтуїтивно зрозумілий інтерфейс, який не відхиляється від основної тематики стилів додатку. Взаємодія з додатком відбувається через сенсорний екран пристрою, за допомогою тапів, жестів та інших елементів керування.

- Режими роботи програми.

Додаток працює в режимі реального часу, надаючи користувачеві можливість миттєво взаємодіяти зі своїми бізнес-акаунтами в Instagram та сервером з усіма даними.

- Видача вихідної інформації.

Додаток надає можливість видачі інформації на екран користувача, а також на друкуючий пристрій для отримання фізичних копій даних.

Часові характеристики.

Часові характеристики серверної частини інтеграції Facebook API залежать від різних факторів, таких як обсяг трафіку, кількість одночасних запитів, складність запитів до бази даних та загальна продуктивність серверного обладнання. Однак, нижче наведені загальні рекомендації, дотримання яких допоможуть підтримувати швидкість та ефективність роботи серверу.

Серверна частина мобільного додатку повинна бути здатна надавати відповідь на запити у межах 100-200 мс. Це забезпечить швидку відгуковість додатку та зручну взаємодію з користувачем. Якщо додаток потребує отримання значної кількості даних з серверної частини, то час завантаження повинен бути прийнятним для користувача. Оптимальний час завантаження залежить від обсягу даних та якості мережевого з'єднання, проте зазвичай рекомендується максимум 2-3 секунди. Якщо виконується складний запит до бази даних, такий як злиття, фільтрація або сортування великих обсягів даних, важливо оптимізувати їх час виконання. Важливо забезпечити, щоб складні запити виконувалися протягом кількох секунд або менше.

Серверна частина повинна масштабуватися для обробки зростаючого обсягу трафіку та кількості користувачів. Використання масштабованих інфраструктур, таких як хмарні технології, може допомогти забезпечити високу доступність та швидкість роботи серверу.

Клієнтська частина має реагувати на дії користувача без помітної затримки, щоб забезпечити плавну і безперебійну роботу інтерфейсу. Затримка між дією користувача і реакцією додатку повинна бути максимально незначною, зазвичай менше ніж 0,2 секунди.

При завантаженні даних з сервера до мобільного додатку, час завантаження має бути оптимізований, щоб забезпечити швидке і безперебійне відображення інформації. Це може включати кешування даних, оптимізацію запитів до сервера та ефективне використання мережевого з'єднання. Час завантаження може залежати від розміру даних і швидкості інтернет-з'єднання, але зазвичай має бути менше ніж 2 секунди.

При обробці даних, таких як створення, редагування або видалення продуктів та клієнтів, додаток повинен працювати швидко та ефективно. Час обробки залежить від обсягу даних та складності поточної операцій. Стандартні операції обробки даних повинні виконуватися менше ніж за 1 секунду.

Якщо додаток отримує оновлення даних з сервера, то час оновлення має бути оптимізований, щоб мінімізувати затримки для користувача. Додаток може використовувати механізми підписки на оновлення або сповіщення, щоб отримувати нову інформацію в режимі реального часу.

1.3.4 Вимоги до програмної документації

При розробці програмної документації важливо враховувати потреби та очікування розробників, які будуть працювати з продуктом. Нижче розглянемо деякі загальні вимоги для програмної документації:

- Опис функціональності програми, включаючи основні можливості, режими роботи та передбачувані результати виконання;
- представлення кроків для встановлення, налаштування та підготовки середовища розробки;
- детальний опис API та інтерфейсів, включаючи типи даних, параметри, повернені значення та приклади використання;
- додавання прикладів коду для основних операцій та складних випадків;
- опис Ієрархічної структури проєкту, модулі, компоненти та залежності;
- правила та процедури для внесення змін до кодової бази.

1.3.5 Стадії та етапи розробки

Розробка клієнтської частини мобільного додатку “Easy Accounting for Social Biz” буде проходити у наступні етапи:

– Аналіз та планування.

Визначення вимог як до клієнтської, так і до серверної частини додатку.

– Проєктування.

Розробка структури. Визначення потрібних компонентів, модулів та функціональності. Розробка прототипів та їх тестування.

– Реалізація.

Написання програмного коду. Розробка основних модулів та компонентів, їх взаємодії та інтеграція зі зовнішніми сервісами.

– Тестування.

Виконання модульних тестів окремих компонентів. Виконання інтеграційних тестів для перевірки взаємодії між різними компонентами.

1.3.6 Порядок контролю та прийому

Прийом розробленого програмного продукту повинен відбуватися в терміни, які зазначені в індивідуальному завданні кваліфікаційної роботи. Для прийому роботи Виконавець повинен надати та представити:

Діюча програма.

Виконавець повинен представити діючу програму, яка повністю відповідає технічному завданню. Це означає, що програма повинна бути повністю розроблена та готова до використання.

Вихідний код.

Виконавець також повинен надати вихідний програмний код, який був використаний для створення програми. Цей код повинен бути записаний на оптичний носій інформації.

Прийом програмного забезпечення повинен відбуватися перед комісією з чотирьох чоловік у такій послідовності:

Доповідь Виконавця.

Виконавець представляє доповідь про виконану роботу. У цій доповіді він розповідає про основні аспекти проєкту, включаючи його цілі, особливості та результати.

Демонстрація програми.

Виконавець демонструє роботу програми, показуючи її функціональність та можливості. Це може включати показ різних функцій, взаємодію з інтерфейсом та демонстрацію роботи на реальних вхідних даних.

Контрольні випробовування.

Комісія проводить контрольні випробовування роботи програми, перевіряючи її функціональність, відповідність технічному завданню та коректність роботи. Це може включати введення тестових даних, перевірку результатів та інші перевірки.

Відповіді на запитання та зауваження.

Комісія має можливість задавати Виконавцю запитання щодо проєкту, а також робити зауваження та коментарі. Виконавець повинен відповісти на запитання та розглянути зауваження, пояснюючи свої рішення та виправляючи помилки, якщо це необхідно.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

2.1 Постановка задачі на розробку програмного забезпечення

У даному підрозділі пояснювальної записки детально розглядаються особливості реалізації інтеграції Facebook API в мобільному додатку “Easy Accounting for Social Biz”. Для того, аби розглянути особливості реалізації, дамо загальний опис, поставленої у кваліфікаційній роботі, задачі.

Задача полягає у розробці як серверної так і клієнтської частини для взаємодії з Facebook API. Необхідно пам’ятати, що додаток вже має певний реалізований функціонал і код, тобто, це означає, що стиль написання нашого коду не має відрізнятись від вже написаного, як на сервері так і на клієнті. Інформацію, яку ми маємо отримати від Facebook api це: бізнесс-акаунт Facebook, прив’язаний до нього бізнес акаунт Instagram, список клієнтів, список товарів, переписку із клієнтами як із Facebook, так і з Instagram в реальному часі.

Розробка нашої логіки буде швидше нагадувати доповнення. Тому, ми повинні адаптуватись до вже існуючих правил і стилів і розробити свою логіку на основі вже створених систем. Існуюча клієнт-серверна модель в додатку зображена на рисунку 2.1. Згідно цієї моделі, вимагається наявність головного сервера, що буде приймати та опрацьовувати клієнтські запити. Клієнти в свою чергу надсилають запити паралельно, тобто вони є незалежними один від одного [4].

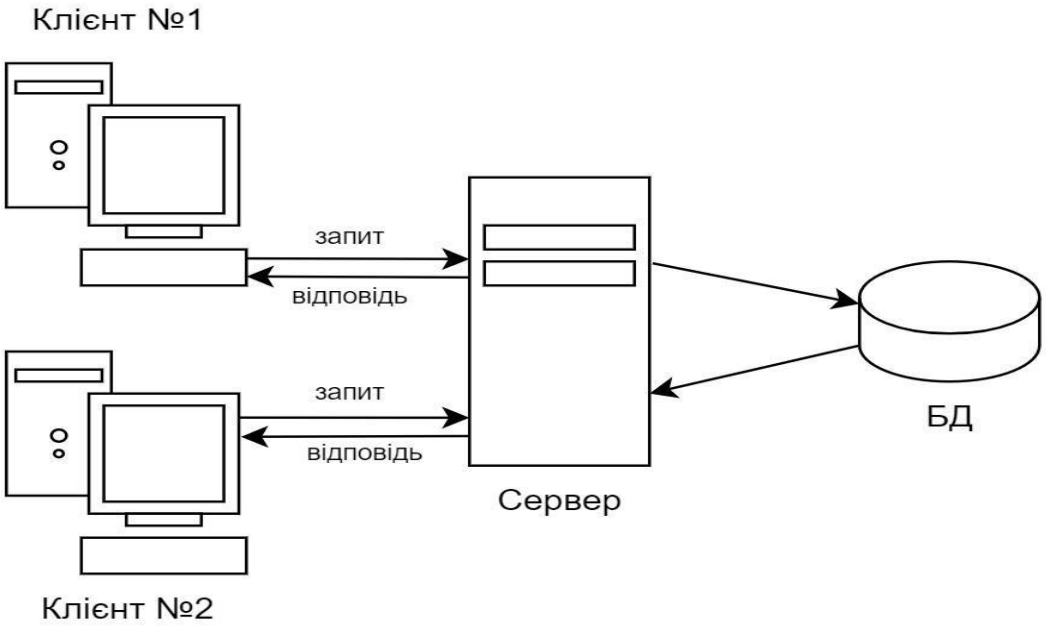

Рисунок 2.1 – Клієнт-серверна модель взаємодії

Типові операції для взаємодії з даними реалізуються за допомогою Hasura. Тобто, це автоматичне створення GraphQL API на основі бази даних. Hasura надає можливість підключити реляційну базу даних, таку як PostgreSQL, та автоматично генерує схему GraphQL, яка відповідає структурі бази даних. Це робить розробку серверної частини додатку швидкою та ефективною, оскільки не потрібно вручну писати та підтримувати велику кількість CRUD операцій.

Для реалізації передачі повідомлень в реальному часі між клієнтом та продавцем використано механізм підписок (subscription) у GraphQL та механізм WebHooks. Підписки дозволять клієнтській частині підписатися на певні події або оновлення даних та отримувати повідомлення в реальному часі при виникненні цих подій. WebHooks, у свою чергу, дозволяють серверу надсилати дані до клієнтської частини, щоб повідомляти про події у режимі реального часу [5].

Принцип роботи WebHooks зображений на рисунку 2.2.

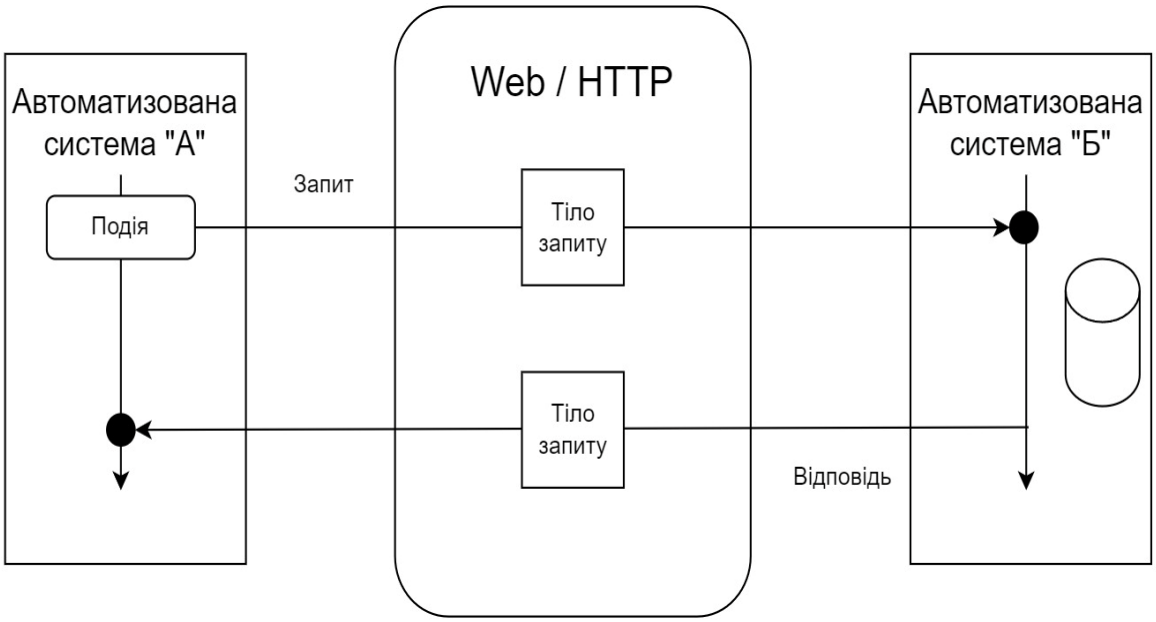

Рисунок 2.2 – Принцип роботи WebHooks

Для реалізації клієнтської частини ми будемо використовувати багато класів задач, кожен з яких потребує детального опису специфічних особливостей реалізації. Це вимагає великої уваги до деталей і глибокого розуміння різних аспектів розробки.

Кожен клас задач має свої особливості реалізації. Опис реалізації включає огляд вимог, аналіз архітектури, вибір потрібних інструментів та технологій, розробку логіки та інтерфейсу користувача. Таким чином, реалізація кожної конкретної задачі для розробки клієнтської частини мобільного додатку потребуватиме детального опису особливостей реалізації. Отже, дамо конкретний опис задачам, а також опишемо особливості їх вирішення.

Перша задача, яка потребує детального опису специфічних особливостей розробки – це управління запитами до Facebook API. У додатку реалізований архітектурний паттерн «BLoC».

У паттерні BloC бізнес-логіка додатку відокремлюється від користувацького інтерфейсу та зберігається в окремих компонентах, які називаються "блоками" (блоками бізнес-логіки). Блоки відповідають за обробку подій та станів додатку, а також за взаємодію з джерелами даних [8]. Тобто, ми будемо розробляти нашу інтеграцію в 3 шарі цього паттерна, а саме в частині доступу до данних.

Частина звязку із джерелами данних, розроблена в окремих класах «репозиторіях». Саме в цих класах ми і будемо писати наші інтеграційні запити, а також будемо звязуватись із локальною базою данних. Все що буде необхідно для бізнес логіки – це отримати доступ до класу- репозиторію. Цей клас буде розроблено за допомогою паттерну «Singleton»

Шаблон singleton — це шаблон, який використовується в об'єктно-орієнтованому програмуванні, який гарантує, що клас має лише один екземпляр, а також забезпечує глобальну точку доступу до нього. Іноді для класу важливо мати лише один екземпляр, інакше ми можемо примусово перевести свою програму в дивний стан. Є кілька прикладів, коли повинен існувати лише один екземпляр класу, і обмеження має бути виконано. Наприклад, нам потрібен лише один екземпляр класу, який представлятиме наше локальне сховище, або ми можете отримати два різних джерела даних, які не синхронізовані. З тієї ж причини операційна система повинна мати рівно одну файлову систему. Ідея полягає в тому, що будь-де у нашому коді, при викликані конструктора класу, він повертатиме той самий екземпляр цього класу з тим самим станом тощо. Формально шаблон Singleton визначається як забезпечення існування лише одного екземпляра класу та існування глобальної точки доступу до нього.

Наступна задача, яка повинна піддатися опису особливостей реалізації з використанням ЕОМ – це забезпечення системи безпеки. Засобами FacebookSDK клієнтська частина повинна звертатись із запитом до сервера автентифікації, для того аби отримати так званий ключ доступу. Після отримання відповідного ключа доступу, що являє собою JWT-токен, клієнтська частина повинна зберігати його локально. При звертанні до Facebook API, клієнт повинен додавати маркер доступу до кожного запиту. Якщо ж строк дії ключа доступу закінчився, або виникла непередбачувана помилка, то клієнт повинен належним чином поновити його, не зупиняючи роботу додатку.

Отже, кроки, які необхідно здійснити для реалізації забезпечення безпеки включають в себе:

- Звертання до серверу автентифікації та отримання ключа доступу;
- Зберегання ключа доступу для подальших операцій;

- Додавання ключа доступу до кожного запиту;
- Поновлення ключа при необхідності.

Послідовність кроків, що необхідно реалізувати у клієнтській частині мобільного додатку, ілюструє UML-діаграма послідовності, що зображена на рисунку 2.3.

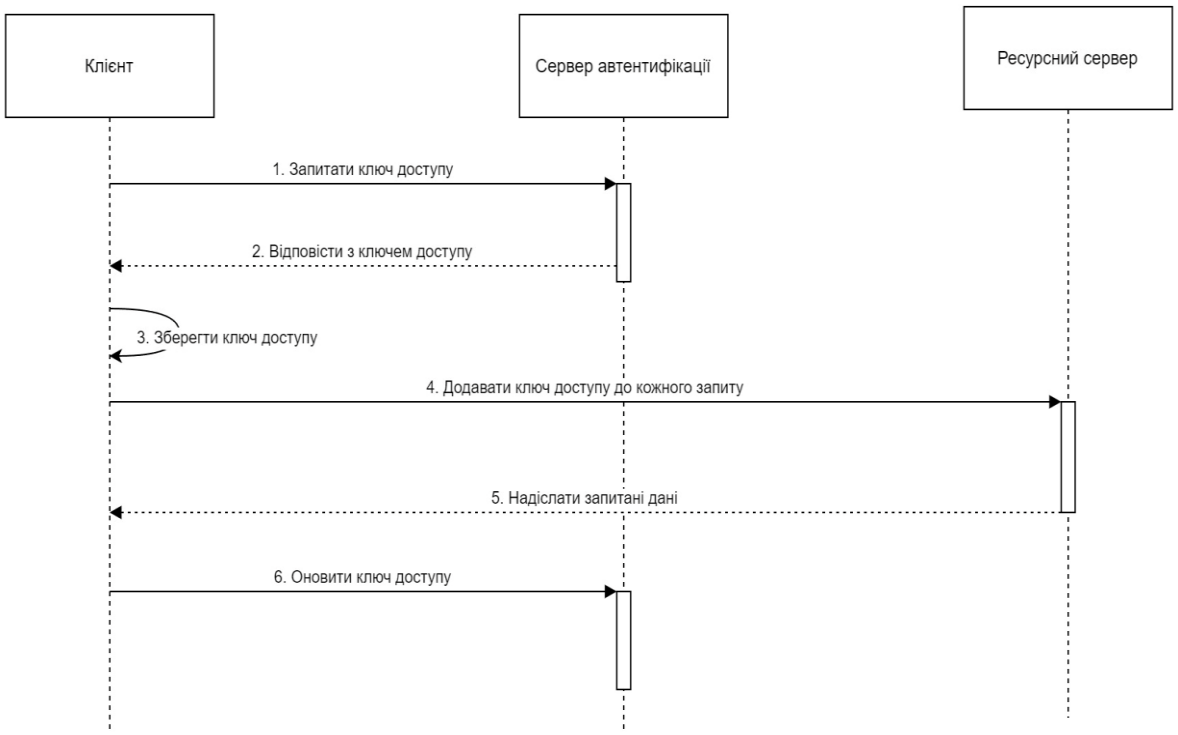

Рисунок 2.3 – UML-діаграма послідовності управління access токеном

2.2 Розробка системи класів

Для зв’язку із сервером, або сторонніми сервісами використовуються класи-репозиторії та класи-сервіси відповідно. Вони відповідають за передачу запитів та даних. Для того, аби вдало спроєктувати систему класів-репозиторіїв та класів-сервісів, необхідно кожному класу задати область даних з якою він буде працювати.

Розподілимо набір даних та призначимо кожному класу свою область. Набір основних даних включає:

- Продукти;
- Замовлення;

- Клієнти;
- Дані Instagram;
- Дані автентифікації.

Таким чином, кожен клас-репозиторій бере на себе обов’язки по обробці певних даних. Отже, будуть створені такі класи як ProductRepository, OrderRepository, ClientRepository, InstagramRepository, AuthenticationRepository.

Далі необхідно спроєктувати систему класів для представлення моделів даних. Моделі даних використовуються для представлення та обробки тієї інформації, яка надходить зовні, з сервера або стороннього сервісу. Ці дані можуть бути у різних форматах і їх слід перетворити на програмні сутності, які легше обробляти в додатку. Моделі даних у мобільному додатку будуть представлені у вигляді класів, які відображають сутності або об'єкти, що містять дані. Ці класи будуть мати властивості, методи та залежності, що визначають поведінку об'єктів.

До основних моделів мобільного додатку “Easy Accounting for Social Biz” увійдуть вже раніше описані продукти, замовлення, клієнти. Крім перерахованих моделів, необхідно створити моделі для відображення даних із соціальної мережі Instagram, таких як сторінка користувача, повідомлення, медіа повідомлення, тощо. Проєктування цих моделів буде базуватися на відповідних моделях Instagram API.

До системи класів також увійдуть другорядні моделі даних, які виокремлюють певну частину даних з основних моделей. Ці допоміжні моделі беруть на себе управління та обробку конкретних аспектів і таким чином спрощують роботу з основними моделями та забезпечують більшу модульність коду. Можуть бути створені такі допоміжні моделі як OrderInfo, OrderProduct та OrderProductInfo.

OrderInfo буде містити другорядну інформацію про замовлення, таку як ідентифікатор, дата створення, тощо. Клас виконує функцію об’єднання загальної інформації про замовлення.

OrderProduct буде представляти окремий товар у замовленні. Він буде містити властивості, такі як ідентифікатор товару, кількість, ціна, тощо. Цей клас

дозволить зберігати та керувати даними про кожен окремий товар, що міститься у замовленні.

OrderProductInfo буде містити додаткову інформацію про товар у замовленні, яка потрібна для відображення та обробки.

2.3 Розробка методів

Розглянемо процес розробки методів NodeJs серверу. Для реалізації функціоналу чату використовуватиметься метод webhook (зворотні виклики). Для початку необхідно визначити усі пакети, залежності та змінні, які будуть використовуватись у роботі сервера. Встановимо залежність dotenv, яка домопоже отримувати дані з .env файлів. Підключимо до сервера body-parser для зручнішої маніпуляції json об’єктами. Додамо підтримку GET методів до webhook [7].

Створимо метод обробки ендпоінту для webhook. Цей метод є обробником POST-запиту на шлях "/webhook" у сервері. Він виконує деякі дії, коли отримує вхідні дані від Instagram webhook. Основний алгоритм методу включає наступні кроки. Метод отримує об'єкт req як параметр, який містить інформацію про вхідний запит. Дані у req.body містить тіло запиту, яке включає інформацію, надіслану веб-хуком від Instagram. За допомогою console.log виводяться вхідні дані для перевірки і налагодження. У цьому випадку виведення виконується у форматі об'єкту з console.dir. Метод перевіряє, чи отримана подія відповідає підписці на сторінку Instagram. У випадку body.object є рівним "instagram", це означає, що це подія від Instagram, і виконується подальший код. В іншому випадку, якщо body.object рівне "page", виводиться повідомлення про невідповідність і повертається статус 404. Коли тип події відповідає Instagram, то метод ітерується через кожну подію у body.entry. Для кожної події перевіряється наявність поля "messaging". У противному випадку, виводиться попередження і переходиться до наступної події. Якщо поле "messaging" присутнє, метод ітерується через кожну подію в entry.messaging. Для кожного повідомлення перевіряється, чи це не "echo" повідомлення (запити, які

повертаються назад до додатка). Якщо повідомлення є "echo", відбувається повернення. В іншому випадку, виконується деяка обробка повідомлення. Виконується операція Hasura, яка передає отримані дані до сервера Hasura для обробки. У цьому випадку викликається функція execute з параметрами, що містять дані повідомлення. Якщо виникли помилки при виконанні операції Hasura, виводиться попередження про помилку. Помилка повертається, і виконання методу для даного повідомлення припиняється. У випадку успішної обробки повідомлення метод повертає статус 200 з текстом "EVENT_RECEIVED". Це підтверджує Instagram, що запит було успішно отримано. Якщо тип події не визнаний, метод повертає статус 404.

Розглянемо метод підписки на повідомлення Instagram. Цей метод є обробником POST-запиту на шлях "/subscribe_to_insta_page". Він відповідає за підписку на повідомлення від Instagram. Основний алгоритм методу включає наступні кроки. Метод отримує об'єкт req як параметр, який містить інформацію про вхідний запит. Параметр req.body.input містить дані, які зберігають page_id (ідентифікатор сторінки) та page_access_token (токен доступу до сторінки). За допомогою бібліотеки request відправляється HTTP-запит до Messenger Platform API [9]. Запит відправляється до офіційного API, що створений компанією Facebook за адресою https://graph.facebook.com/${page_id}/subscribed_apps з параметрами subscribed_fields (підписані поля, у даному випадку - "messages") та access_token (токен доступу до сторінки). Після виконання запиту, метод отримує відповідь від Messenger Platform API. Якщо відповідь не містить помилок (err дорівнює null), то повертається статус 200 з повідомленням "Subscribed to messages!". Якщо відповідь містить помилку, повертається статус 400 з повідомленням про нездатність підписатися на повідомлення та текстом помилки.

Перейдемо до розробки методів для класів, що комунікують з сервером та сервісами Facebook API. Розробимо методи у контексті сервісу авторизації та класу “AuthenticationRepository”. Основні завдання, які повинен виконувати клас – це авторизація, за допомогою різних засобів, проте, нас цікавить саме

авторизація за допомогою сторінки Facebook. Отже, завдання по авторизації ми винесемо у метод signInViaFacebook().

Для індикації успіху авторизації –метод буде мати повертаюче значення типу bool. Авторизація передбачає звертання до зовнішнього сервісу, тому метод повинен працювати за асинхронною моделлю, а отже повертати значення в обгортці Future.

2.4 Опис файлової структури

Правильна файлова структура є важливою при розробці серверної частини з кількох причин.

Правильна файлова структура допомагає організувати проект і забезпечує логічний розподіл компонентів та модулів. Це полегшує співпрацю в команді розробників, полегшує зрозуміння проекту та його архітектури.

Правильна структура дозволяє легко знаходити необхідні файли і компоненти, що сприяє ефективному управлінню проектом. Розміщення файлів за логічними категоріями та шаровою структурою спрощує роботу над проектом.

Правильна структура файлів дозволяє забезпечити масштабованість та розширюваність програми. Легко додавати нові функціональності, модулі і компоненти без необхідності розбиратися в хаосі непорядної структури.

Правильна файлова структура допомагає зберігати стандартизований підхід до розробки проекту. Вона дозволяє всім розробникам працювати з однаковою структурою та дотримуватися загальних правил і стандартів організації коду.

Правильна структура файлів сприяє легкому тестуванню і підтримці проекту. Зручний доступ до необхідних файлів та компонентів допомагає здійснювати тестування, розробку виправлень та змін без зайвих зусиль. В цілому, правильна файлова структура є основою для успішної розробки серверної частини.

Розглянемо файлову структуру серверної частини мобільного додатку “Easy Accounting for Social Biz”. Детальний аналіз файлової структури сприяє

полегшенню розробки та підтримки додатку в майбутньому. Далі розглянемо кожен каталог та його призначення, а також визначимо ключові файли, які відіграють важливу роль у функціонуванні додатку.

В кореневій папці проекту окрім стандартних README.md та package.json знаходяться файли конфігурації VCS системи git. Такі файли як .gitattributes та .gitignore забезпечують правильну роботу системи контролю версій під час роботи з версіями програми.

У папці functions та nodejs-express містяться файли для NodeJs серверу, що забезпечує роботу чату в реальному часі. Папка functions призначеня для зберігання файлу index.js, у якому прописані загальні налаштування для взаємодії з ресурсним сервером та сервером автентифікації. Також у ньому відбувається імпорт усіх пакетів та компонентів, що необхідні для реалізації функціоналу.

Наступна папка у файловій структурі серверної частини відповідає за організацію версій бази даних на різних етапах розробки проекту. Каталог rm_migrations зберігає ряд інших папок та файлів. Файли з розширенням env містять змінні HASURA_GRAPHQL_ENDPOINT та HASURA_GRAPHQL_ADMIN_SECRET, вони використовуються для доступа NodeJs серверу до Hasura. Каталог metadata, що міститься вседерині rm_migrations, зберігає файли з розширеннями yaml та graphql. У файлі actions.graphql прописані запити на мові GraphQL, які описуються підписку на таблицю з даними про чат продавця з клієнтами.

Файли з розширенням yaml, що знаходяться у папці rm_migrations/databases призначенні для опису структури таблиці бази даних, їх зв’язків між собою, дозволи на взаємодію з даними таблиці та фільтри, що забезпечать відображення лише тих даних, які належать користувачу. Приклад такого файлу можна побачити у додатку Х.

Клієнтська частина містить декілька основних каталогів, з якими ми будемо працювати. Головною ознакою, за якою приймається рішення про групування тих чи інших файлів у певні каталоги (пакети) – є функціональне призначення. Таким чином, якщо нагромаджується певна група файлів, що

мають схоже призначення, буде прийняте рішення щодо розміщення їх в одному пакеті.

Каталог "lib" є одним із найважливіших каталогів, оскільки він містить вихідний код (логіку) мобільного додатку. Каталог "lib" є центральним місцем для розробки функціоналу додатку та організації його структури. У каталозі "lib" створюються підкаталоги та розміщуються файли з розширенням ".dart". Ці файли містять код на мові програмування Dart, який описує поведінку та функціональність мобільного застосунку.

Каталог "graphql" призначений для реалізації GraphQL-запитів на стороні клієнта. Він містить файли з розширенням ".graphql", де описані запити, що використовуються при звертанні до серверної частини.

Опишемо детальніше внутрішню структуру та вміст папки “lib”, адже вона містить ключові підкаталоги, в яких і визначається основна логіка мобільного застосунку.

Каталог "api" містить файли, пов'язані із взаємодією з API, такі як клієнти для роботи з мережевими запитами, моделі даних, сервіси та утиліти, пов'язані з роботою з API.

Каталог "di" використовується для реалізації залежностей (Dependency Injection). Він містить файли, пов'язані з налаштуванням контейнера залежностей, провайдери сервісів та фабрики об'єктів.

2.5 Визначення інформаційних зв’язків

Інформаційні зв’язки програмних компонентів відносяться до процесу обміну даними, повідомленнями або викликами функцій між різними програмними частинами або модулями, що працюють разом у складі більшої системи. Це означає, що компоненти взаємодіють один з одним, щоб виконувати певні функції, обмінюватися даними або спільно працювати над досягненням загальної мети.

Взаємодія програмних компонентів може мати різні форми, залежно від архітектури системи та способу організації комунікації.

Деякі з найпоширеніших способів взаємодії включають:

– Виклик функцій.

Компоненти можуть взаємодіяти шляхом виклику функцій одного компонента з іншого. Це може включати передачу параметрів та отримання результату виконання функції.

– Події.

Компоненти можуть публікувати або підписуватися на події, які стосуються певних подій або станів системи. Це дозволяє сповіщати інші компоненти про зміни та реагувати на них.

– Передача повідомлень.

Компоненти можуть обмінюватися повідомленнями або пакетами даних через певний канал зв'язку, такий як мережа або спільна пам'ять. Це дозволяє передавати та отримувати дані між компонентами.

Способи взаємодії дозволяють компонентам співпрацювати та обмінюватися необхідною інформацією, щоб виконувати свої функції та спільно працювати у складі системи.

Отже, для визначення інформаційних зв’язків серверної частини мобільного додатку “Eazy Accounting for Social Biz”, проаналізуємо кожен програмний компонент, після чого визначимо всі залежності та відносини між ними.

Для того, аби визначити та упорядкувати програмні зв’язки, наведемо список програмних компонентів, які беруть участь у роботі серверної частини.

Програмні компоненти, що задіяні у роботі серверної частини включають в себе:

– Flutter-клієнт;

– зовнішній сервіс Instagram;

– WebHook-сервер Node.js;

– ресурсний сервер Hasura;

– база даних серверної частини.

Після того, як були визначені програмні компоненти, опишемо їх інформаційні зв’язки.

Flutter-клієнт має інформаційні зв'язки з ресурсним сервером Hasura. Коли Flutter-клієнт надсилає запит до сервера, він встановлює зв'язок з серверною частиною, що реалізовано з використанням GraphQL API. Запити надсилаються через мережу до сервера Hasura, де вони оброблюються відповідним чином. Після обробки запиту серверна частина може надіслати відповідь до Flutter-клієнта, який отримує і обробляє цю відповідь. Таким чином, інформаційні зв’язок полягає у тому, що Flutter-клієнт та Hasura взаємодіють між собою за допомогою мережевих запитів і відповідей.

Визначимо інформаційні зв’язки між такою групою програмних компонентів: зовнішнім сервісом Instagram та сервером Node.js. Взаємодія між Instagram-сервісом та сервером Node.js відбувається за допомогою Webhooks. Webhooks є механізмом, який дозволяє одній системі (в даному випадку Instagram) автоматично надсилати повідомлення до іншої системи (серверу Node.js) про події, що відбуваються в першій системі. У даному випадку, Instagram-сервіс надсилає повідомлення до Webhook-сервера Node.js про певні події, наприклад, коли клієнт надіслав нове приватне повідомлення. Ці повідомлення надсилаються на певну URL-адресу, що вказується під час налаштування Webhook-сервера. Webhook-сервер Node.js слухає вхідні HTTP-запити на зазначеній URL-адресі і обробляє отримані повідомлення від Instagram-сервісу.

Отже, інформаційний зв’язок між Instagram-сервісом та Webhook-сервером Node.js проявляється у передачі повідомлень в режимі реального часу за допомогою Node.js серверу, дозволяючи йому отримувати та обробляти інформацію про події, що відбуваються в сторонньому сервісі Instagram.

Визначимо взаємозв’язок між ресурсним сервером Hasura та базою даних, у нашому випадку це PostgreSQL. Hasura аналізує структуру та поля запиту GraphQL, який надходить до нього. Він розбирає запит та визначає, які таблиці бази даних відповідають різним частинам запиту. Після аналізу запиту GraphQL Hasura генерує відповідний SQL-запит для виконання на базі даних PostgreSQL. Цей SQL-запит містить всю необхідну інформацію для вибірки, фільтрації, сортування та об'єднання даних відповідно до специфікацій запиту GraphQL.

Hasura також проводить оптимізацію SQL-запиту, щоб забезпечити кращу продуктивність та швидкодію виконання запиту до бази даних. Він може застосовувати різні оптимізаційні стратегії, такі як кешування, використання індексів тощо, для поліпшення продуктивності запиту. Після трансформації та оптимізації Hasura виконує згенерований SQL-запит на базі даних PostgreSQL. Запит виконується у базі даних, і результати відправляються назад до Hasura. Hasura отримує результати SQL-запиту з бази даних і перетворює їх у відповідь GraphQL, яку він надсилає клієнту. Відповідь містить дані, які були запитані в запиті GraphQL, у відповідному форматі.

2.6 Тестування та налагодження програми

Для забезпечення якості та надійності серверної частини програмного продукту використовується процес тестування, який спрямований на перевірку функціональності, продуктивності та безпеки серверного додатку.

Існує багато наборів автоматизованих тестів для різних видів бекенду. Але для GraphQL API найкраще рішення – це Karate [10]. Karate – це набір автоматизованих тестів з відкритим кодом від Intuit для тестування API Hasura GraphQL. Karate має масу функцій, від автоматизації браузера для тестування інтерфейсу користувача до паралельного тестування та оптичного розпізнавання символів.

Щоб розпочати тестування необхідно локально розгорнути копію серверної частини мобільного додатку. Для цього використовується інструментарій для керування ізольованими контейнерами. Усі необхідні сервіси потрібно помістити в один контейнер. Це можна зробити за допомогою команди docker-compose up [11]. Контейнер будуватиметься на основі спеціального конфігураційного файлу з розширенням “yaml”, у якому міститься вся інформація про розгортання різних сервісів та взаємодію між ними.

Сервіси, у свою чергу, описують відокремлені програмні засоби, що потрібні для роботи проекту. У випадку цієї кваліфікаційної роботи – це контейнер бази даних, в основі якого лежить image PostgreSQL, сервер на NodeJS

для реалізації інтегрованого чату та сервер бізнес-логіки Hasura GraphQL Engine. Після збірки усіх сервісів серверна частина працюватиме на зазначених портах, як це показано на рисунку 2.4.

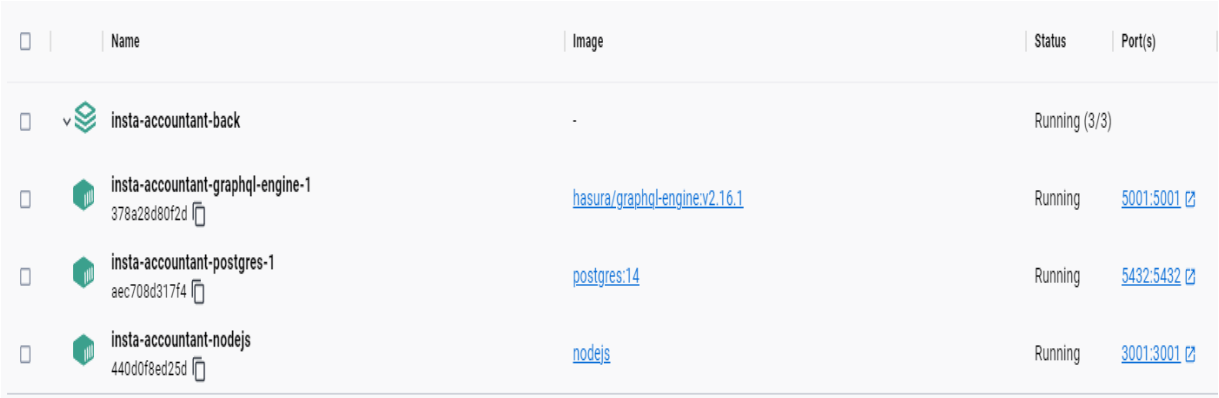

Рисунок 2.4 – Локально розгорнутий Docker контейнер

Другим кроком є підключення Karate до працюючого серверу. До вищеописаного контейнеру Docker потрібно добавити контейнер Karate, що буде сформований на основі DockerFile. Сервіс Karate виглядатиме наступним чином див. рисунок 2.5.

```
karate:
  build: ./karate
  container_name: karate
  ports:
    - "5900:5900"
  depends_on:
    - "insta-accountant-graphql-engine"
```

Рисунок 2.5 – Сервіс набору автоматизованих тестів Karate

Після запуску сервісу Karate можна почати писати тести для серверної частини. Усі тести прописуються у файлі з розширенням feature, також у ньому вказуються реквізити для звертання до сервера. В файлах feature існує поняття Scenario, що містить сценарій тестування якогось функціоналу [12]. Розглянемо

процес тестування запитів на створення та отримання замовлення, клієнта та продукта.

Замовлення потребує двох речей: клієнт та продукт. Протестуємо запит на створення продукту. Першим кроком буде створення файлу з розширенням gql, що міститиме запит на мові GraphQL до сервера. Щоб Karate мав змогу отримати код з файлу необхідно його створити на одному рівні з файлом feature. Після цього можна приступати до написання Scenario створення продукту див. рисунок 2.6.

```
Scenario: Create product
  Given def query = read('insertProduct.gql')
  And def variables = { count: 1, description: 'test', price: 50, title: 'product'}
  And request { query: '#(query)', variables: '#(variables)' }
  And header x-hasura-admin-secret = 'admin-secret'
  And header x-hasura-role = 'user'
  And header x-hasura-user-id = 'jDU8p2tDz0W1BomwkowoxXhAHph1'
  And header Authorization = 'Bearer token'
  When method post
  * status 200
  * match response == "#object"
  * match response.errors == '#notpresent'
  * def row_obj = response.data.insert_products
  * match row_obj contains { affected_rows : 1 }
```

Рисунок 2.6 – Scenario створення продукту

Перша змінна у test case'і зчитує GraphQL запит з файлу, який разом з json об'єктом variables формують запит до сервера Hasura. Хедери x-hasura-admin-secret, x-hasura-role, x-hasura-user-id та Authorization відповідають за захист серверу від несанкціонованого доступу. Admin secret потрібен для доступу до ендпоїнтів сервера.

Роль визначає функціонал, який доступний користувачу, що відсилає запит. Hasura user id та токен відповідає за ідентифікацію користувача на сервері, за допомогою цих хедерів Hasura налаштує зв'язок між рядком користувача та продукту у базі даних.

Рядки коду після “When method post” визначають очікувані результати обробки запиту. Karate перевірятиме чи сервер повернув об’єкт та чи немає у json об’єкті інформації про помилки.

Після цього оператор contains перевіряє наявність поля affected_rows у відповіді, який показує інформацію по кількість змінених рядків у базі даних. Запускаємо Scenario за допомогою плагіну у Visual Studio Code, або командою docker-compose up karate див. рисунок 2.7.

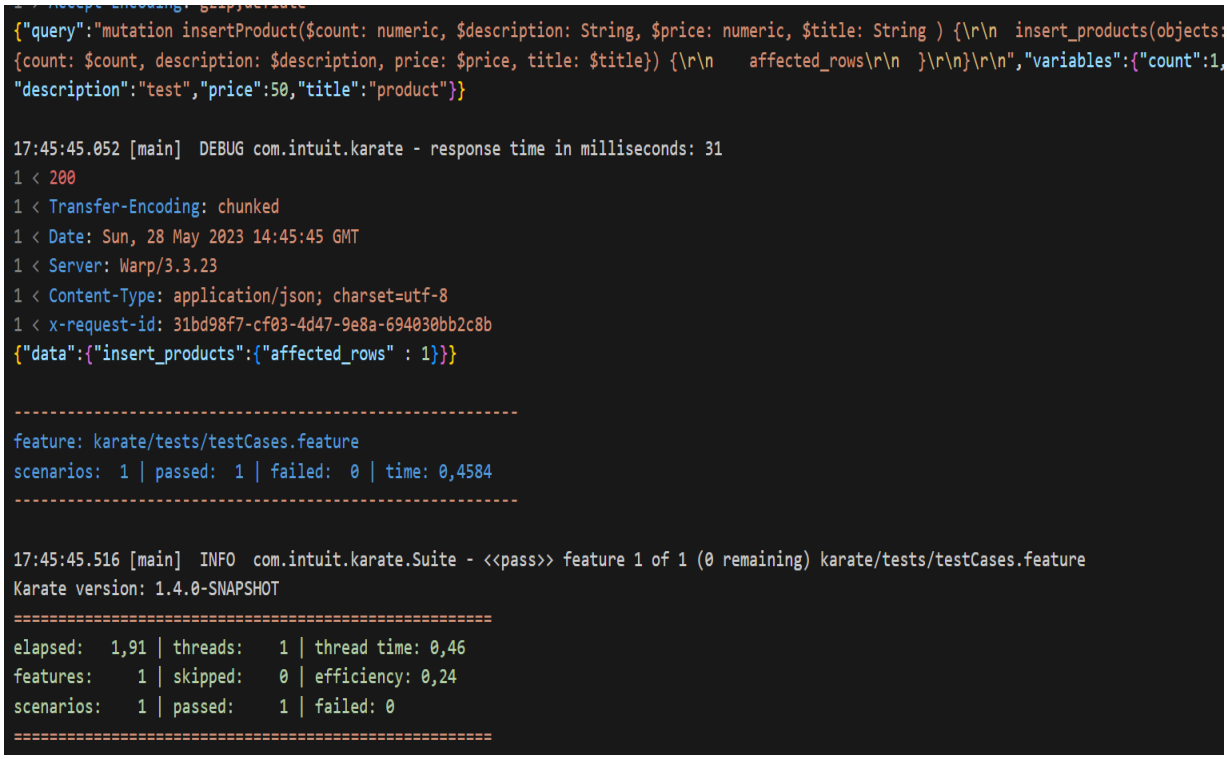

Рисунок 2.7 – Результат тестування Scenario на створення продукту

За допомогою відповідного сценарію, виконаємо тестування запиту до ресурсного серверу Hasura, який дозволить отримати детальну інформацію про певний продукт. Використовуючи сценарій, створюється реалістична ситуація, яка демонструє, як клієнт взаємодіє з сервером Hasura і у якому вигляді отримує необхідні дані, див. рисунок 2.8.

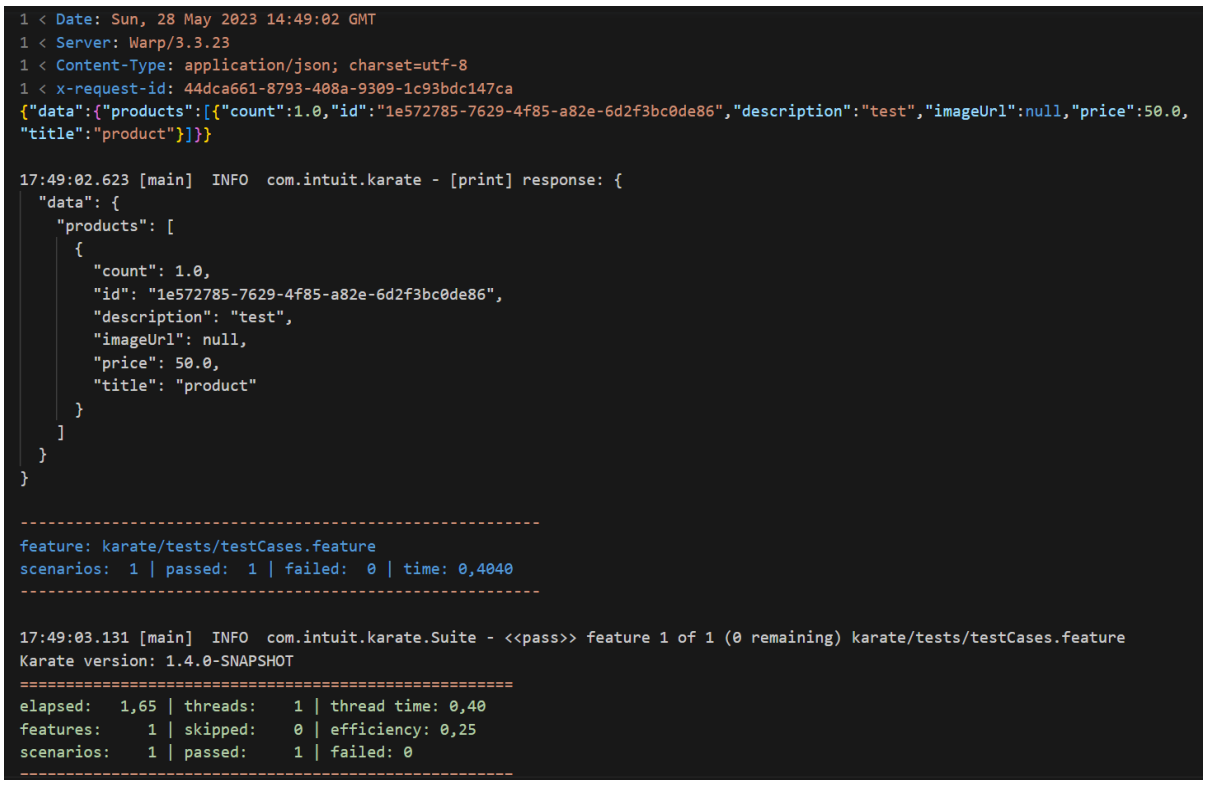

Рисунок 2.8 – Результат тестування Scenario на отримання продукту

Проведемо аналогічне тестування щодо клієнта див. рисунки 2.9 й 2.10.

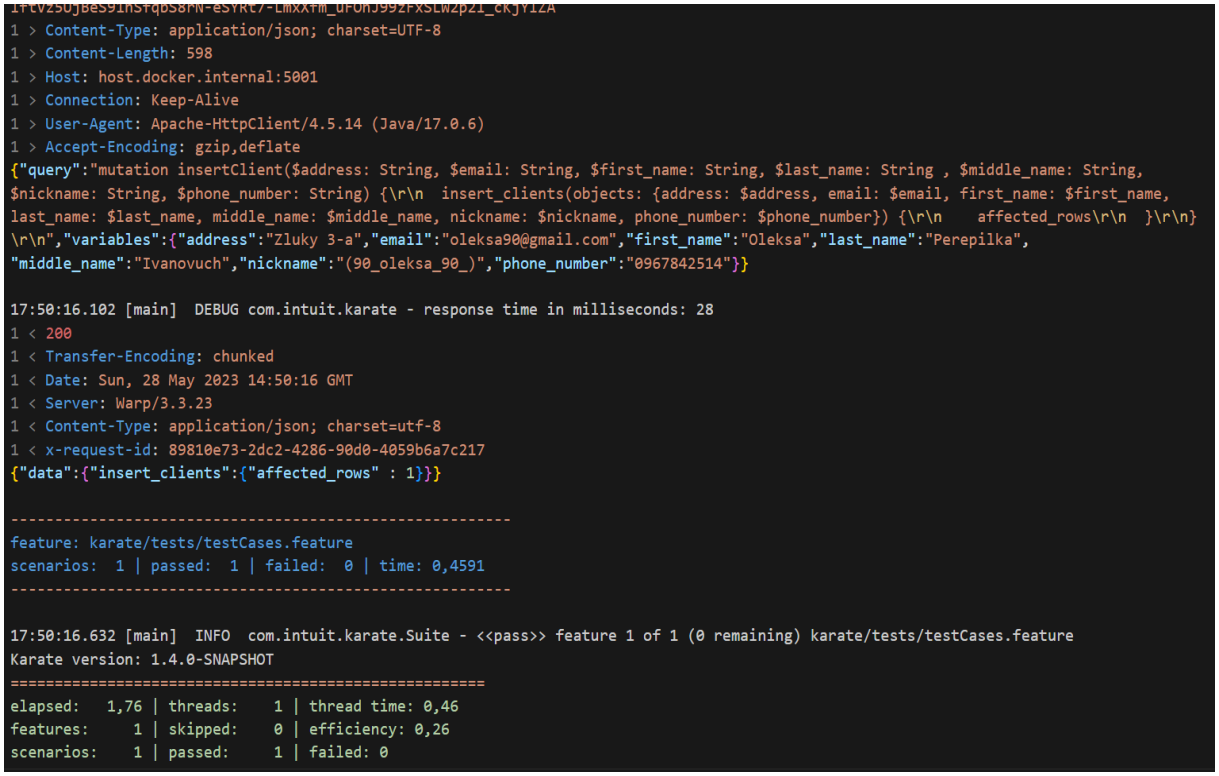

Рисунок 2.9 - Результат тестування Scenario на створення клієнта

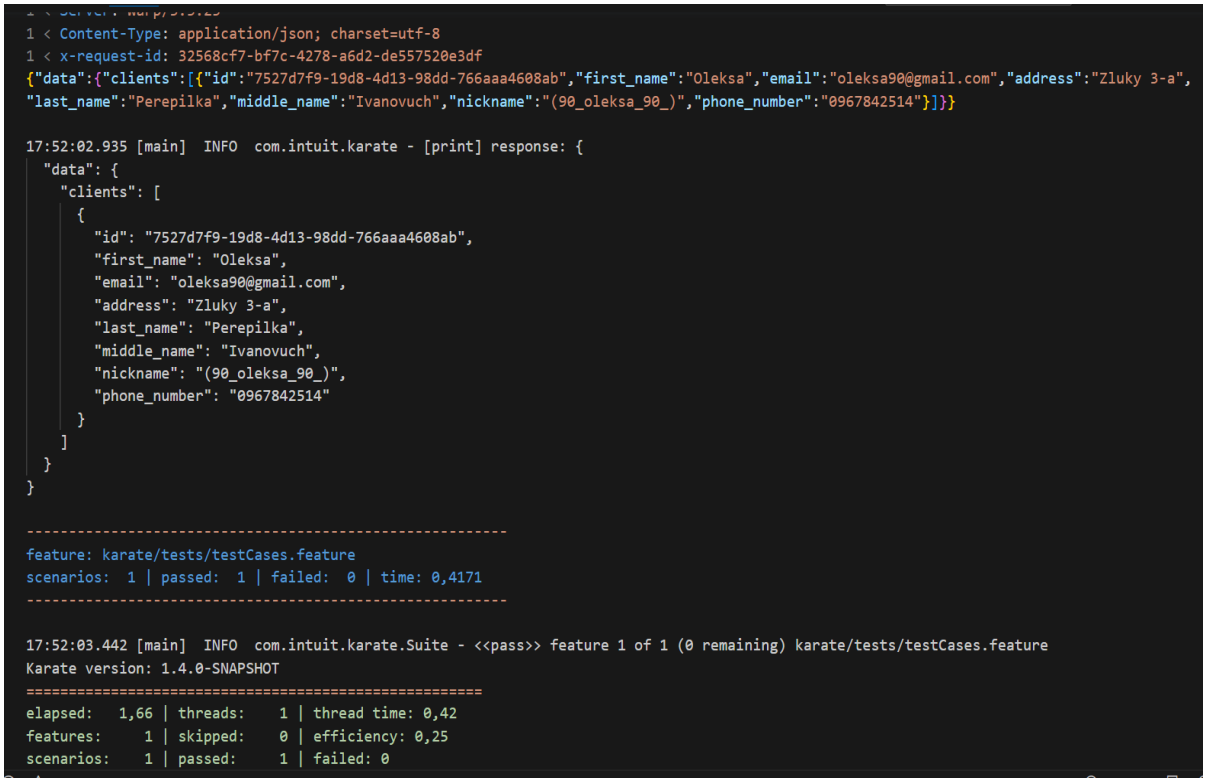

Рисунок 2.10 - Результат тестування Scenario на отримання клієнта

Усі тести пройшли успішно і в базу даних додалися нові рядки. Переглянути актуальні дані можна на ресурсному сервері, який локально піднятий у докері див. рисунок 2.4.

Переходимо у браузер за адресою: “http://localhost:5001/console/api/api-explorer”. За допомогою Hasura GraphQL Engine та його зручного інтерфейсу можна ефективно контролювати стан даних та тестувати запити в інтегрованій консолі.

Після додавання клієнта та продукту можна створити замовлення. У таблиці замовлень існують посилання на зовнішні ключі клієнта та продукта, тому для створення нового order необхідно у GraphQL mutation запиті передати відповідні id кожного елементу.

Ці дані можна дізнатися у консолі Hasura. Для цього переходимо у вкладку DATA і вибираємо потрібну таблицю, у нашому випадку - це clients, як це показано на рисунку 2.11.

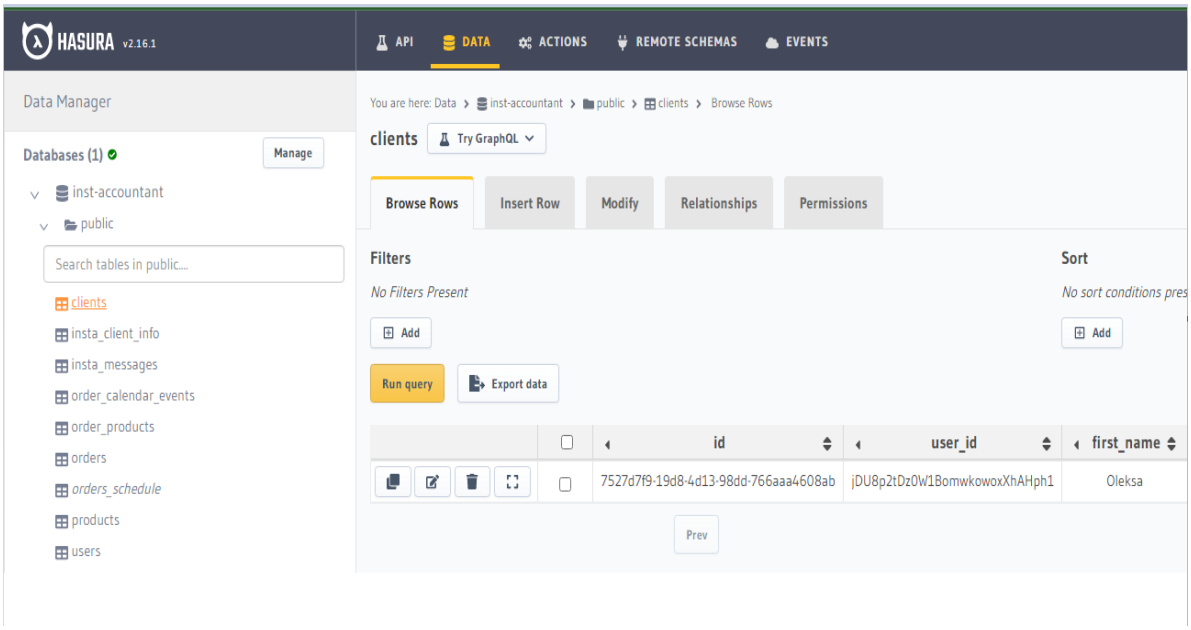

Рисунок 2.11 – Інтерфейс серверу Hasura

У таблиці бачимо рядок з раніше створеним клієнтом, копіюємо його id та вставляємо у json об’єкт variables. У Scenario прописуємо усі потрібні хедери, реквізити і звичайно файл з запитом на мові GraphQL. Окрім вже описаних перевірок потрібно впевнитися у тому, що замовлення належить клієнту з іменем ‘Oleksa’, який був створений при тестуванні запиту на додавання клієнта. Щоб це реалізувати потрібно, щоб GraphQL запит повертав client_id після створення замовлення. Розглянемо такий запит на рисунку 2.12.

```
mutation insertOrder($client_id: uuid = "", $comment: String = "", $event_id: String = "",
$paid: numeric = "", $perform_to: timestamptz = "", $status: String = "") {
  insert_orders(objects: {client_id: $client_id, comment: $comment, event_id: $event_id,
  paid: $paid, perform_to: $perform_to, status: $status}) {
    affected_rows
    returning {
      client_id
    }
  }
}
```

Рисунок 2.12 – Запит GraphQL на створення замовлення

Як бачимо запит окрім вже описаного affected_rows повертає id клієнта, якому належить це замовлення. Тепер є можливість витягнути client_id,

помістити його у зміну та порівняти його з наявним у базі даних див. рисунок 2.13.

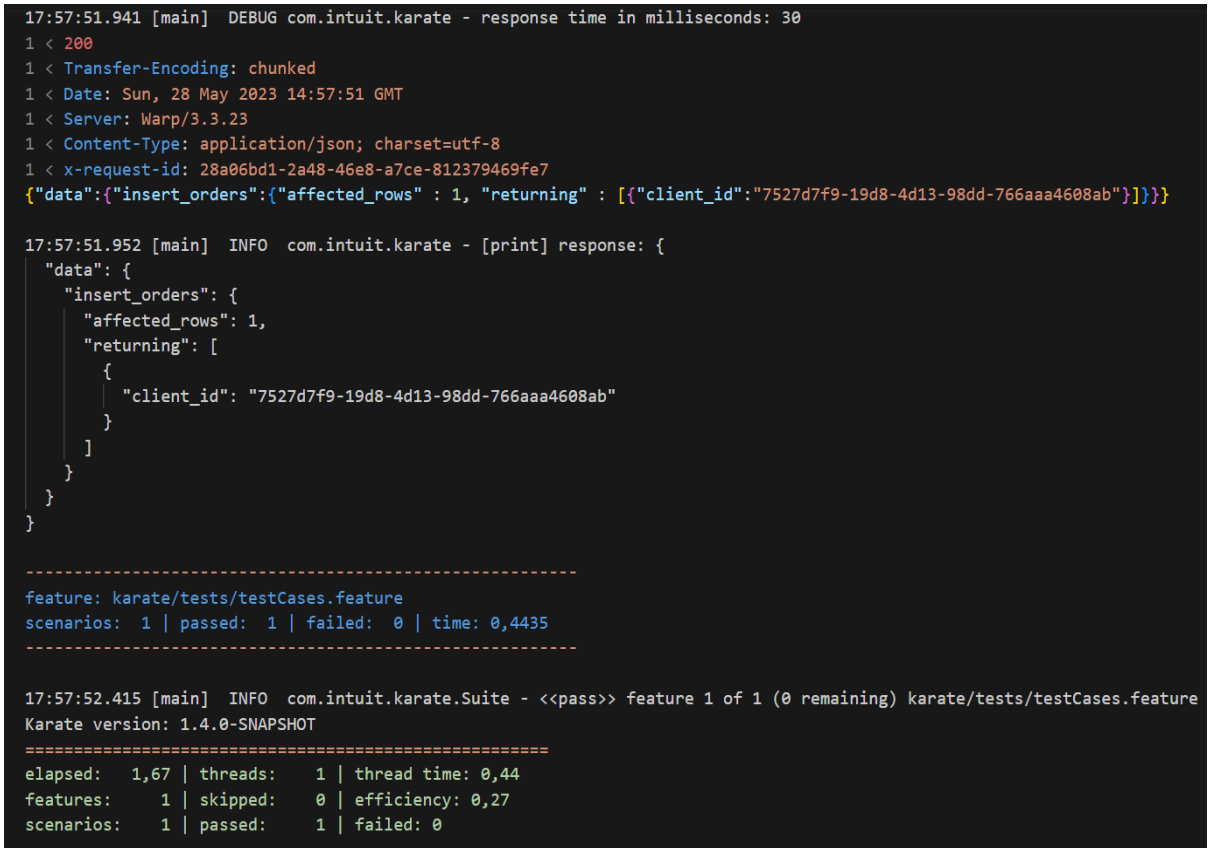

Рисунок 2.13 – Результат тестування Scenario на створення замовлення

Замовлення успішно створено і відповідно результат тестування не видав помилок. Можна скористатися графічним інтерфейсом консолі Hasura та оглянути новостворений рядок у таблиці orders, як це показано з клієнтами на риснку 2.11.

Тепер протестуємо видалення замовлення. Щоб уникнути можливих помилок із зовнішніми ключами та додати можливість адміністраторам переглядати вже видаленні замовлення, рядок у базі даних лише буде позначатися як видалений. Для цього у кожній таблиці існує поле deleted_at з типом даних timestamp. У це поле записується дата видалення відповідного рядка у базі даних. Таким чином можна буде легко знайти потрібний запис, знаючи приблизну дату видалення.

Створимо відповідний запит та Scenario для перевірки видалення замовлення див. рисунок 2.14.

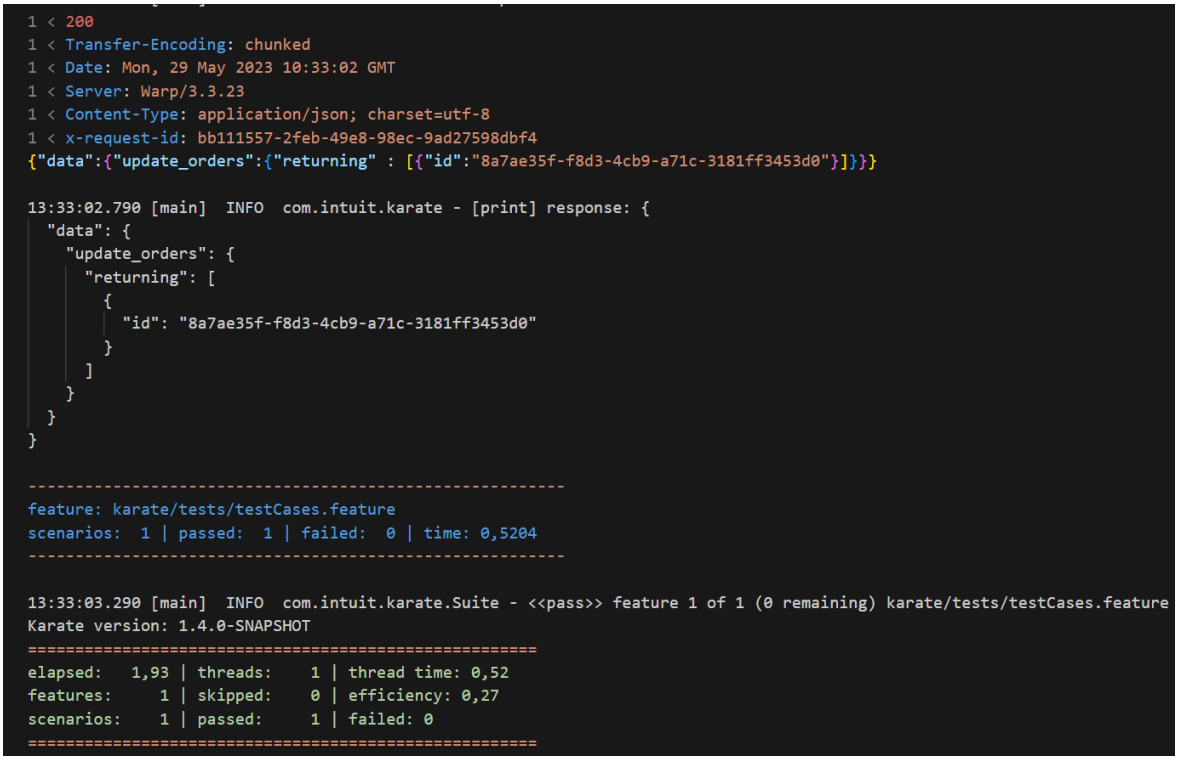

Рисунок 2.14 – Результат тестування Scenario на видалення замовлення

Для перевірки працездатності клієнтської частини мобільного додатку "Easy Accounting for Social Biz" використовувалося декілька видів тестування, деякі з них включає:

– Інтеграційне тестування;

– тестування продуктивності.

Були розроблені автоматизовані тести, написані мовою програмування Dart. Ці тести дозволяють автоматично перевіряти функціональність та правильність роботи додатку шляхом запуску набору тестових сценаріїв.

Автоматизовані тести включають в себе різні рівні тестування, такі як модульне тестування та інтеграційне тестування. Модульні тести перевіряють правильність роботи окремих функцій та модулів додатку. Інтеграційні тести перевіряють взаємодію клієнтської частини з іншими системами або сервісами.

Отже, опишемо процес тестування клієнтської частини мобільного додатку "Easy Accounting for Social Biz", виконаємо декілька тестових сценаріїв для кожного виду тестування, аби впевнитись у працездатності застосунку.

Наступним видом тестування, що необхідно провести є інтеграційне, тобто перевірка взаємодії клієнтської частини із зовнішніми системами та сервісами. Виконаємо тестовий сценарій – текстова переписка із клієнтом. Тут задіяні декілька різних систем, включаючи серверну частину та сторонній сервіс “Instagram API”.

Від лиця клієнта, надсилаємо текстове повідомлення через соціальну мережу Instagram. Знаходимо у списку замовлення, яке прив’язане до тестового клієнта, відкриваємо сторінку із чатом. На сторінці з’являється повідомлення надіслане від клієнта. Результат тестування зображений на рисунку 2.15.

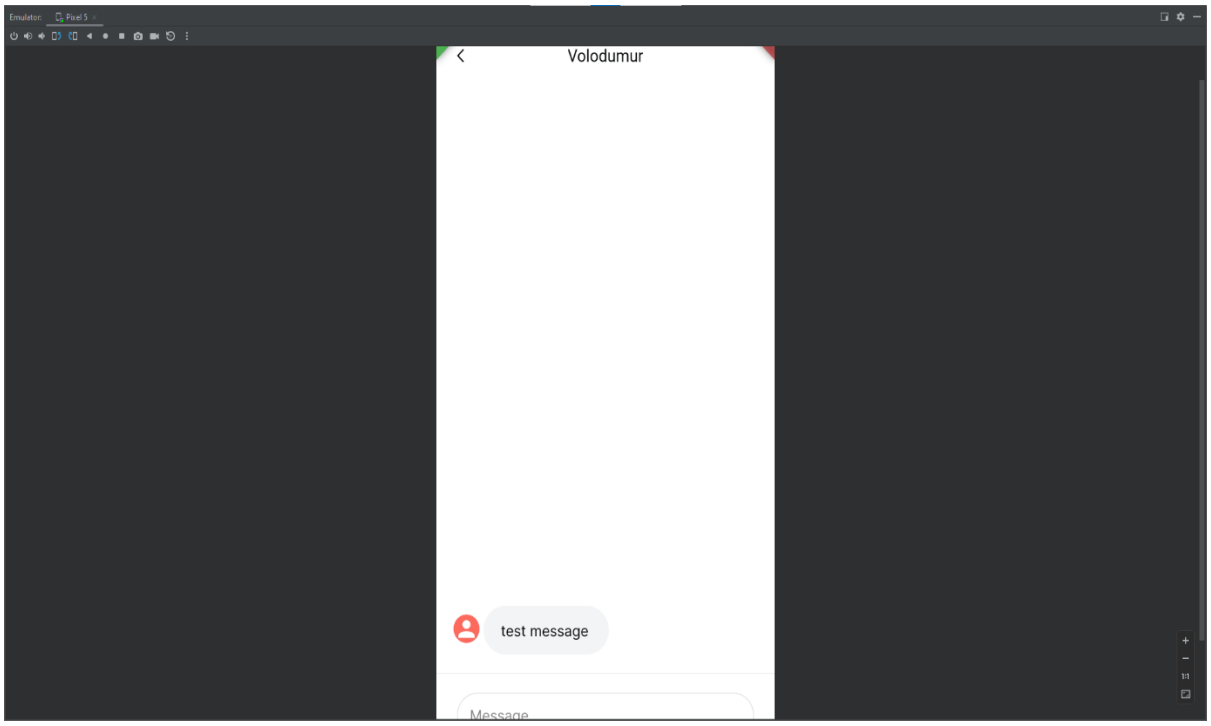

Рисунок 2.15 – Результат тестування отриманих повідомлень

Спробуємо надіслати зустрічне повідомлення клієнту, від лиця власника соціального бізнесу, використовуючи мобільний додаток. Після введення та відправки повідомлення: – “I got your message!”, воно з’явилося на екрані. Після перевірки всередині мобільного додатку, переходимо у приватні повідомлення тестового клієнта у соціальній мережі “Instagram”, та бачимо, що повідомлення успішно надіслане. Результат тестування зображено на рисунку 2.16.

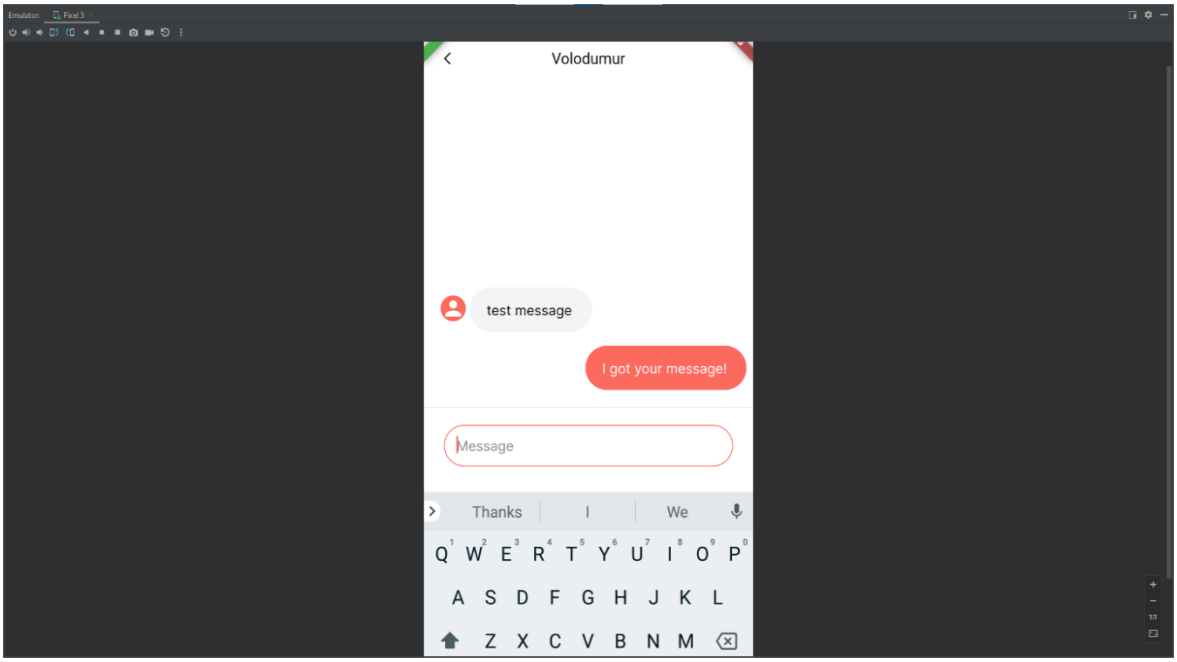

Рисунок 2.16 – Результат тестування надісланих повідомлень

Наступним важливим етапом є тестування продуктивності. По мірі використання додатку, дані користувача лише зростають в об’ємі, тому необхідно подбати про перевірку та оптимізацію запитів на завантаження цих даних.

Виконаємо тестовий сценарій – перевірка оптимізації завантаження продуктів. Завантажуємо великий обсяг тестових даних про продукти, після чого переходимо на сторінку із продуктами. Намагаємось завантажити тестові продукти, заміряємо час, який знадобився на відображення їх на екрані, після чого повторюємо процес 2-3 рази. Щоразу, продукти успішно завантажились, час на відображення кожного продукту не перевищив значення у 2 секунди. Виконуємо перевірку запиту до серверу, намагаємось удосконалити конструкцію запиту, аби зменшити загальний час завантаження продуктів. Результат тестування завантаження та відображення продуктів зображений
на рисунку 2.17.

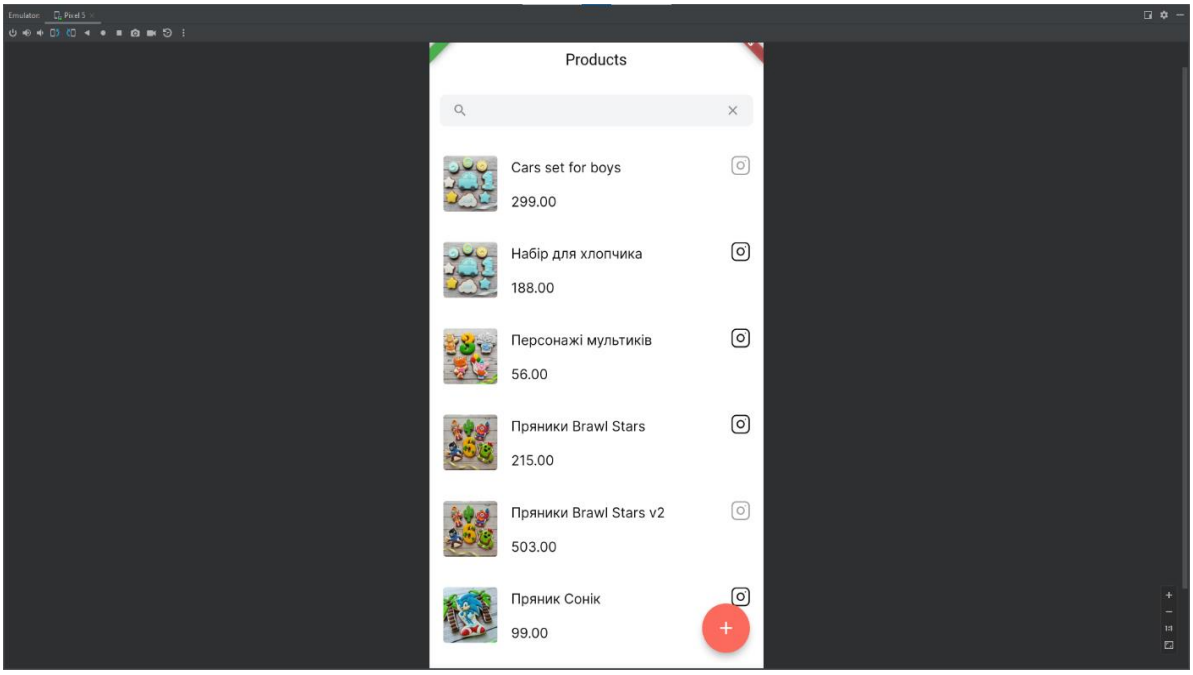

Рисунок 2.17 – Результат тестування продуктивності завантаження продуктів

Отже, під час перевірки працездатності клієнтської частини мобільного додатку "Easy Accounting for Social Biz" були успішно пройдені основні види тестування, що включали інтеграційне тестування та тестування продуктивності. Ці тести допомогли виявити та усунути помилки різного характеру та складності, що посприяло підвищенню якості розробленого програмного продукту.

Після проходження тестування мобільний додаток "Easy Accounting for Social Biz" відповідає вимогам технічного завдання і може забезпечувати коректну роботу та задоволення потреб користувачів. Завдяки вдалому тестуванню та виправленню помилок, розроблений програмний продукт здатний забезпечити надійну та ефективну роботу. Гарантується, що додаток буде працювати стабільно, відповідати функціональним вимогам та надавати користувачам очікувані можливості.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Інструкція з інсталяції програмного забезпечення

Оскільки ми розробляєм інтеграцію Facebook API в мобільний додаток, то, відповідно, щоб користуватись нашим функціоналом, потрібно інсталювати сам додаток. Для інсталяції та використання мобільного додатку "Easy Accounting for Social Biz", необхідна наявність певних технічних пристроїв та засобів. Визначення мінімальної конфігурації комплексу технічних засобів гарантує, що додаток буде правильно інсталюватися та задовольняти вимоги функціональності.

Для інсталяції програмного забезпечення, потрібно мати мобільний пристрій, який відповідає певним технічним характеристикам. У таблиці 3.1 наведено список мінімальних технічних характеристик, які повинен задовільняти мобільний пристрій для успішної інсталяції розробленого програмного продукту.

Таблиця 3.1 – Мінімальні характеристики для мобільного пристрою

Назва вимоги	Мінімальне значення	
	Android	IOS
Версія ОС	Android 9.0 (Pie)	IOS 11.0
Процесор	1 ГГц	
Оперативна пам’ять	2 ГБ	
Дисплей	5,5 дюйма	
Роздільна здатність	720 x 1280 пікселів	
Швидкість з’єднання	3 мбіт/с	
Вбудована пам’ять	2 ГБ	

Мобільний додаток був розроблений для операційних систем Android та iOS і може стабільно працювати на них. Для інсталяції програмного забезпечення на пристрої з ОС Android необхідно мати мінімальну версію

Android 9.0 (Pie) або вище. Для пристроїв з ОС IOS, мінімальна версія операційної системи повинна бути IOS 11.0 або вище.

Визначені мінімальні вимоги до операційних систем гарантують сумісність мобільного додатку зі смартфонами та планшетами, які працюють на цих платформах. Рекомендується користуватись оновленнями операційних систем для забезпечення оптимальної роботи додатку та доступу до всіх його функцій.

Встановлення мобільного додатку на смартфон відбувається за допомогою інсталяційних файлів. Інсталяційні файли будуються на основі файлів налаштування, де прописані відповідні дерективи для визначення необхідних параметрів збірки додатку.

Мобільний додаток "Easy Accounting for Social Biz" для Android, встановлюється за допомогою файлу інсталяції з розширенням ".apk". Файл .apk (Android Application Package) є архівом, який містить всі необхідні файли та ресурси додатку для його встановлення та запуску на пристроях з операційною системою Android.

Для встановлення мобільного додатку на смартфон з операційною системою IOS необхідний інсталяційний файл відповідного розширення – .ipa. Файл з розширенням .ipa є пакетом iPhone Application, який використовується для інсталяції та розповсюдження додатків на платформі IOS.

Обидва файли .apk та .ipa належать до завантажувального типу представлення програм.

Для встановлення застосунку на Android необхідно скомпілювати.apk файл, після чого завантажити його на мобільний пристрій, після чого запустити на виконання та надати необхідні дозволи програмі.

Для інсталяції застосунку на операційну систему IOS необхідно скомпілювати готовий .ipa файл з доданим до ним цифровим підписом, після чого запустити на виконання файл інсталяції. Також потрібно надати дозволи, які запитує програма.

3.2 Інструкція з використання тестових наборів

Даний підрозділ містить інформацію та керівництво щодо використання тестових наборів для перевірки якості, функціональності та стабільності засобів інтеграції Facebook API. Цей підрозділ включає розгляд різних типів тестів, таких як аварійні, вироджені, стикові, структурне тестування та функціональне тестування.

Інструкція надає опис кроків, які необхідно виконати перед тестуванням, процес виконання тестових наборів, аналіз результатів. Наголошується та акцентується увага на важливості використання тестових наборів як частини процесу розробки для забезпечення якості та надійності. Отже, опишемо процес запуску набору тестів та передбачимо використання тестів різних типів.

Перед тестуванням необхідно переглянути опис тестового набору для розуміння його складу та цілей, слід переконатися, що середовище розробки та тестування налаштоване і готове до виконання тестів. Також потрібно перевірити, чи наявні всі необхідні залежності та ресурси для виконання тестів. Після дотримання даних умов, можна запускати виконання тестових наборів згідно з вимогами проєкту та вибраними типами тестів.

Під час виконання тестових наборів, необхідно слідкувати за процесом проходження та реєструвати результати виконання. У випадку появи помилок або несподіваних результатів необхідно скласти відповідне повідомлення про проблему.

Після проходження тестування, потрібно провести аналіз результатів тестування та зробити висновки про якість додатку. Тестові набори необхідно використовувати постійно як частину процесу розробки та випуску нових версій додатку. Після внесення знім, необхідно порівняти результати з попередніми тестовими запусками для оцінки вдосконалення додатку з часом.

Після складання інструкції проведення та опису процесу запуску тестових наборів, перейдемо до визначення та опису аварійних ситуацій, що увійдуть до кожного типу тестування.

При проєктуванні тестових наборів для мобільного додатку слід враховувати необхідність використання набору аварійних тестів. Аварійні тести спрямовані на перевірку реакції додатку на несподівані ситуації, помилки та винятки. Вони створюють сценарії, що відтворюють аварійні ситуації, і перевіряють, чи здатний додаток відновитися або обробити помилку без збоїв. Наведемо та опишемо приклади аварійних ситуацій.

Недоступність сервера – перевірка, як додаток реагує на недоступність сервера або проблеми з мережевим з'єднанням.

Помилкові дані введення – перевірка, як додаток обробляє некоректні дані, такі як неправильний формат електронної пошти, слабкий пароль, або введення спеціальних символів. Некоректна обробка помилок – перевірка, як додаток реагує на помилки та винятки. Незавершені операції – перевірка, як додаток поводиться під час незавершених або неочікуваних операцій, таких як відключення пристрою або переривання процесу завантаження.

Вироджені тести створюють сценарії, що відтворюють некоректні, непередбачувані або неправильні умови використання додатку. Їх мета полягає у перевірці того, як додаток реагує на такі умови та чи може він відновити нормальне функціонування. Опишемо, які випадки може включати вироджене тестування.

Після опису набору вироджених тестів, необхідно передбачити та описати набір стикових (edge) тестів. Стикові тести спрямовані на перевірку роботи додатку у межах його функціональних та системних меж.

Останній набір тестів – функціональний. При проєктуванні набору тестів для функціонального тестування мобільного додатку, слід враховувати потребу в ретельному перевірянні функціональних можливостей додатку згідно з його специфікацією та вимогами.

Виконання кожного набору тестів передбачає дотримання визначених правил тестування програмного забезпечення.

Щоб провести тестування на стороні сервера потрібно використати фреймворк Karate. Запустити тести можна виконавши наступні вказівки.

Для запуску Karate необхідно мати встановлений JDK (Java Development Kit). Щоб встановити JDK потрібно відкрити веб-браузер та перейти на офіційний веб-сайт Oracle для завантаження JDK. Прийняти ліцензійну угоду, вибрати версію JDK, яку треба встановити, та вибрати правильну платформу для операційної системи (Windows, macOS або Linux).

Після завантаження JDK виконати встановлюваний файл, щоб запустити програму інсталяції. Під час інсталяції вибрати потрібні параметри. За замовчуванням ми можемо залишити параметри без змін. Далі обрати місце для встановлення JDK. За замовчуванням рекомендується залишити шлях без змін. Натиснути "Next" або "Install" та дочекатися завершення процесу встановлення.

Щоб перевірити встановлений JDK потрібно відкрити командний рядок або термінал. Ввести команду java -version та натиснути "Enter". Якщо команда спрацьовує та видає інформацію про встановлене програмне забезпечення, то JDK було успішно встановлено.

Слід пам’ятати, що після встановлення JDK також потрібно налаштувати змінні середовища для коректної роботи з Java.

Щоб запустити тести з системою збірки Gradle використовується команда “gradle test”, для Maven: “mvn test”.

3.3 Інструкція з експлуатації програмного комплексу

У цьому підрозділі будуть наведені докладні кроки щодо роботи з додатком "Easy Accounting for Social Biz" для взаємодії з Facebook API. Буде розглянуто спосіб звертання до комплексу та надані рекомендації щодо шляхів розміщення командних файлів. Цей підрозділ допоможе отримати повну інформацію щодо коректного використання програмного комплексу.

Рекомендується розміщувати командні файли комплексу та інформаційні файли системи в певних визначених шляхах, щоб забезпечити належну організацію та доступ до цих файлів. Для командних файлів комплексу, які включають в себе, скрипти для збірки, тестування, чи розгортання додатку,

рекомендується розміщувати їх у спеціально створеному каталозі з назвою "scripts" або "bin" в кореневій директорії проєкту.

Щодо інформаційних файлів системи, таких як конфігураційні файли, локалізаційні файли або файли зі статичними ресурсами, рекомендується розміщувати їх у відповідних підкаталогах каталогу "lib". Наприклад, конфігураційні файли можна розмістити у каталозі "config", локалізаційні файли - у каталозі "locale", а файли зі статичними ресурсами (зображення, шрифти, іконки, тощо) – у каталозі "assets".

Після успішного входу в додаток з’являється нижня панель, яка слугує навігаційним меню додатку. Для її використання, достатньо вибрати відповідну іконку та натиснути на неї. Панель містить 5 різних вкладок, які надають доступ до основних функцій додатку – замовлення, календар, продукти, клієнти та налаштування.

На вкладках "Клієнти" та "Продукти" в додатку "Easy Accounting for Social Biz" користувач може додавати нових клієнтів та продукти. На сторінці присутні активні кнопки, що надають можливість імпортувати клієнтів або продукти з облікового запису соціальної мережі. Після натискання на відповідну кнопку, додаток буде отримувати необхідну інформацію про клієнтів або продукти з облікового запису Instagram та автоматично заповнювати відповідні поля в додатку. На рисунку 3.1 зображена UML-діаграма послідовності дій для додавання клієнта.

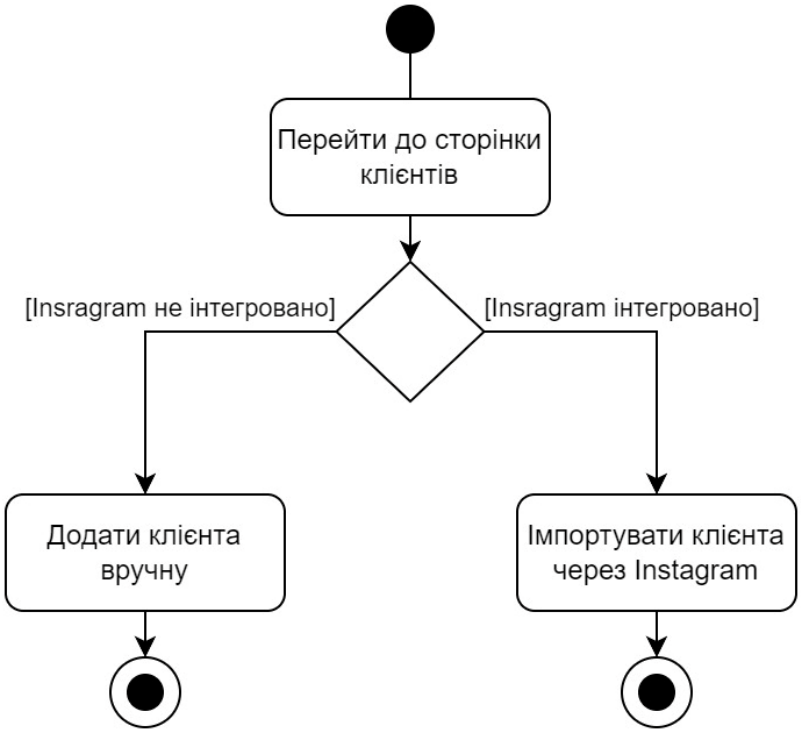

Рисунок 3.1 – UML-діаграма послідовності дій додавання клієнта

На вкладці "Замовлення" користувач може створювати нові замовлення для свого бізнесу. При створенні замовлення важливо прив'язати його до конкретного клієнта, оскільки це дозволяє зберігати зв'язок між замовленням та клієнтом для подальшого аналізу та обліку. При створенні нового замовлення користувач повинен вибрати наявного клієнта зі списку клієнтів, які були додані раніше. Якщо клієнт відсутній у списку, користувач має можливість створити нового клієнта прямо зі сторінки створення замовлення, натиснувши кнопку з іконкою “плюс”.

Для повноцінної роботи з вкладками "Замовлення", "Клієнти" та "Продукти" необхідна інтеграція з соціальною мережею Instagram. Інтеграція дозволяє забезпечити зручний доступ до функцій, пов'язаних з управлінням замовленнями, клієнтами та продуктами прямо з Instagram-акаунту. Відповідний пункт меню для інтеграції знаходиться на вкладці "Налаштування". Щоб підключити соціальну мережу Instagram до додатку, необхідно пройти процес автентифікації через обліковий запис Instagram. Після успішної автентифікації, додаток отримає доступ до необхідних даних з Instagram-акаунту.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Освітлення виробничих приміщень

Залежно від джерела світла освітлення буває природне, штучне та суміщене. Природне створюється прямими сонячними променями та розсіяним світлом небосхилу, штучне –- електричними джерелами світла, суміщеним є освітлення, коли недостатнє природне освітлення доповнюється штучним[19].

4.1.1 Штучне освітлення

Штучне освітлення передбачається у всіх виробничих та побутових приміщеннях, де недостатньо природного світла, а також для освітлення у темний період доби приміщень, відкритих робочих ділянок, місць проходу людей та руху транспорту. Від його якості залежить продуктивність праці, здоров'я та безпека праці робітників.

Як джерела світла при штучному освітленні використовуються лампи розжарювання, газорозрядні лампи та світлодіоди. З точки зору психології бажано, щоб спектральний склад випромінювання джерел штучного освітлення максимально наближався до спектру природного, оскільки колір світла впливає на продуктивність праці. Так, якщо при білому світлі продуктивність праці людини за визначений термін часу прийняти за 100 %, то при жовтому світлі продуктивність вже складе 93 %, при зеленому – 92 %, при голубому – 78 %, а при червоному 76 %. Джерела світла повинні мати естетичний вигляд, зручно розміщуватися, світловий потік повинен мати таке направлення, щоб не засліплювати очей людини. Основними характеристиками джерел світла є номінальна напруга, споживана потужність, світловий потік, світлова віддача та строк служби.

Лампи розжарювання відносяться до теплових джерел світла. Вони характеризуються простотою конструкції та виготовлення, відносно низькою вартістю, зручністю експлуатації, широким діапазоном напруги та потужностей.

Разом з перевагами їм притаманні і суттєві недоліки: велика яскравість (засліплююча дія); низька світлова віддача (7 –- 20 лм/Вт); відносно малий термін експлуатації (до 2,5 тис. годин); переважання жовто-червоних променів у порівнянні з природним світлом; висока температура нагріву (до 140 °С і вище), що робить їх пожежонебезпечними.

Газорозрядні лампи. У цих лампах балон наповнюється парами ртуті та інертним газом, на внутрішню поверхню балона наносять люмінофор. Газорозрядні лампи бувають низького (люмінесцентні) та високого тиску. Основною перевагою газорозрядних ламп є їх економічність. Світлова віддача цих ламп становить 40 –- 130 лм/Вт, що в 3 –- 7 разів перевищує світлову віддачу ламп розжарювання.

Люмінесцентні лампи мають світлову віддачу 50 –- 80 лм/Вт, малу яскравість поверхні, що світиться, близький до денного спектральний склад світла. Термін експлуатації ламп досягає 10 тис. годин, а температура нагріву (люміне- 60 сцентні) – 30 – 60 °С. За спектральним складом видимого світла люмінесцентні лампи бувають: денного світла (ЛД), денного світла з покращеною передачею кольорів (ЛДЦ), холодного білого (ЛХБ), теплого білого (ЛТБ) та білого (ЛБ) кольорів.

Широко застосовуються також дугові ртутні лампи високого тиску (ДРЛ, ДРІ) та дугові трубчаті ксенонові та натрієві лампи (ДКсТ, ДНаТ), які мають значний строк служби та світловіддачу відповідно від 50 до 130 лм/Вт. Потужність цих ламп може досягати кількох кіловат і більше. Їх доцільно використовувати для освітлення промислових приміщень значної висоти, кар’єрів, території складів, зовнішнього освітлення тощо.

Основним недоліком газорозрядних ламп є пульсація світлового потоку, що погіршує умови зорової праці та може зумовити виникнення стробоскопічного ефекту, який полягає у спотворенні зорового сприйняття об'єктів, що рухаються. До недоліків цих ламп можна віднести також складність схеми включення, шум дроселів, значний час між включенням та запалюванням ламп, відносно високу вартість та наявність ртуті в колбі ламп, нестабільну роботу при низьких температурах та зниженій напрузі джерел живлення.

Негативну дію пульсуючого світлового потоку знижують шляхом вмикання сусідніх ламп у різні фази мережі живлення та підвищенням частоти струму живлення.

Останнім часом широкого розповсюдження набули енергозберігаючі компактні люмінесцентні лампи з електронною схемою керування, конструкція яких дозволяє замінювати ними лампи розжарювання. Крім того, ці лампи, на відміну від традиційних люмінесцентних ламп, не створюють пульсацій світлового потоку.

Джерело світла (лампа) разом з освітлювальною арматурою складає світильник. Освітлювальна арматура перерозподіляє світловий потік лампи в просторі або перетворює його властивості (змінює спектральний склад випромінювання), захищає очі працівника від засліплення. Окрім того, вона захищає джерело світла від впливу оточуючого пожежо- та вибухонебезпечного, хімічно-активного середовища, механічних ушкоджень, пилу, бруду, атмосферних опадів.

Світильники відрізняються цілою низкою світлотехнічних та конструктивних характеристик. Основними світлотехнічними характеристиками світильників є: світлорозподіл, крива сили світла, коефіцієнт корисної дії та захисний кут.

За світлорозподілом, що визначається відношенням потоку випромінюваного світильником в нижню півсферу до його повного світлового потоку, світильники поділяються на п’ять класів: прямого потоку світла (> 80%); переважно прямого світла (60 % < 80 %); розсіяного світла (40 % < 60 %); переважно відбитого світла (20 % < 40 %); відбитого світла (< 20 %).

Криві сили світла (КСС) світильників можуть мати різну форму в просторі навколо світлового приладу: концентровану (К), глибоку (Г), косинусну (Д), напівширoку (Л), широку (Ш), рівномірну (М), синусну (С). 61 Захисний кут світильника – це кут, утворений горизонталлю, що проходить через нитку розжарювання лампи (поверхню люмінесцентної лампи) та лінією, яка з'єднує нитку розжарювання (поверхню лампи) з протилежним краєм освітлювальної арматури.

Захисний кут визначає ступінь захисту очей від впливу яскравих частин джерела світла, тому його величину враховують з-поміж інших чинників при визначенні місця та висоти розташування освітлювальних приладів.

4.1.2 Нормування виробничого освітлення

В основу нормування виробничого освітлення покладено залежність необхідного рівня освітлення від зорової напруги (розряду та підрозряду зорової роботи). Розряд зорової роботи визначається розміром об'єкта розпізнавання, а підрозряд – контрастом між об'єктом і фоном та характеристикою фону. Всього встановлено вісім розрядів (залежно від розміру об'єкта розпізнавання), в свою чергу розряди (I – V) містять чотири підрозряди (а, б, в, г) — залежно від контрасту між об'єктом і фоном та характеристики фона (коефіцієнта відбиття). Нормування освітлення в громадських, допоміжних та житлових будівлях здійснюють залежно від призначення приміщення. Нормування природного освітлення здійснюється за коефіцієнтом природної освітленості. Нормовані значення КПО визначаються „Державними будівельними нормами України (Природне і штучне освітлення. ДБН В.2.5-28- 2006)”. При використанні системи бічного природного освітлення (крізь віконні прорізи у стінах) нормується мінімальне значення КПО (у точці робочої поверхні, розташованій на відстані 1 м від стіни, що найбільш віддалена від світлових прорізів). При використанні системи верхнього чи комбінованого природного освітлення нормується середній КПО, обчислений за результатами вимірювань у N точках (не менше 5) умовної робочої поверхні (або підлоги). Перша та остання точки приймаються на відстані 1 м від поверхні стін. Нормування штучного освітлення здійснюється за абсолютним значенням освітленості, яке залежить від характеристики зорової праці та системи освітлення (загальне, комбіноване). Найбільша нормована освітленість складає 5000 лк (розряд I а), а найменша – 30 лк (розряд VIII в). Витяг з ДБН В.2.5-28- 2006 нормативних значень освітлення для деяких розрядів зорової роботи наведений у табл. 4.1. Вимірювання освітленості проводять за допомогою люксметрів Ю 16, Ю 17, Ю 116, MS6610, Testo 540, ТКА-ЛЮКС, RS

180-7133 та їх модифікації. Принцип дії люксметрів оснований на явищі фотоелектричного ефекту. При освітленні поверхні фотоелемента, включеного в замкнутий електричний ланцюг, при цьому в ній виникає фотострум, який відхиляє рухому частину магнітоелектричного вимірника. Важливе значення для створення раціональних умов освітлення, забезпечення потрібних величин освітленості без додаткових витрат електроенергії має ретельний і регулярний догляд за устаткуванням природного та штучного освітлення. Забруднення скла світлових отворів, ламп та світильників може знизити освітленість приміщень у 1,5–2 рази. Тому вікна необхідно 62 мити не рідше двох разів на рік для приміщень з незначним виділенням пилу і не рідше чотирьох разів – при значному виділенні пилу. Періодичність очищення світильників складає 4–12 разів на рік (залежно від характеру запиленості виробничих приміщень).

Таблиця 4.1 – Норми штучного освітлення виробничих приміщень

Характери-стика здорової роботи	Найменший розмір об’єкта розпізнавання, мм	Розряд зорової роботи	Штучне освітле-ння		Сумісне освітлення	
			Освітленість, лк		КПО, %	
			Комбі-новане	Загаль-не	Верхнє або ком-біноване	Бокове
Високої точності	0.3-0.5	III	2000-400	500-200	3	1.2
Середньої точності	0.5-1.0	IV	750-300	300-150	2.4	0.9
Малої точності	1-5	V	300-200	200-100	1.8	0.6
Загальне спостереження за ходом виробничого процесу		VIII	-	75-30	0.7	0.2

4.2 Основні шкідливі фактори, що впливаютть на стан здоров’я людей, які працюють за комп’ютером

Можна виділити наступні основні фактори, що впливають на стан здоров'я людей, які працюють за комп'ютером:

- Сидяче положення на протязі тривалого періоду;
- вплив електромагнітного випромінювання монітора;
- втома очей, навантаження на зір;
- перевантаження суглобів кистей;
- стрес при втраті інформації.

У кожному з цих випадків ступінь ризику прямо пропорційний часу, що проводиться за комп’ютером і поблизу нього [18].

Сидяче положення па протязі тривалого періоду. Хоч за комп'ютером людина сидить в розслабленому положенні, проте для організму така поза є вимушеною і неприємною: напруження шиї, плечей, рук. Звідси надмірне навантаження на хребет, остеохондроз. У тих хто багато сидить між сидінням стільця і тілом створюється свого роду тепловий компрес, що призводить до застою крові в тазових органах і як наслідок - простатит та геморой. Крім того малорухливий спосіб життя часто приводить до ожиріння.

Проблеми, пов'язані з електромагнітнім випромінюванням. Кожен пристрій, який проводить або споживає електроенергію, створює електромагнітне випромінювання. Це випромінювання зосереджено навколо пристрою у вигляді електромагнітного поля. Електромагнітне випромінювання може викликати розлади нервової системи, зниження імунітету, розлади серцево-судинної системи аномалії в процесі вагітності.

По можливості необхідно мінімізувати згубний вплив електромагнітного випромінювання:

- Системний блок і монітор повинні знаходитись якнайдалі від людини;
- не залишати комп’ютер включеним на тривалий час якщо його не використовують;

- не забувати використовувати “сплячий режим” для монітора;
- у зв'язку з тим. що електромагнітне випромінювання від стінок монітора набагато більше, рекомендується поставити монітор в кут, так щоб випромінювання поглиналося стінками;
- по можливості скорочувати час роботи за комп'ютером і частіше перериватиробот.

Шкідливо знаходитись в полі позитивно заряджених іонів, а це зона перед екраном дисплея електронно-променевої трубки. Таке поле негативно впливає на слизисту оболонку ока, носоглотки та шкіру рук. „Сохне горло", „сохне шкіра" —типові скарги. Зняти це поле можна хорошим заземленням.

Проблеми зору. Це найпоширеніша проблема. М'язи, які управляють очима і фіксують їх ні певному предметі, просто втомлюються від надмірного навантаження. Потенційна втома очей існує при будь-якій роботі, в якій бере участь зір, але найбільша вона, коли потрібно розглядати об'єкти на близькій відстані. Проблема стає ще більшою, якщо така діяльність пов'язана з використанням пристроїв високої яскравості, наприклад, монітора комп'ютера.

Найочевидніше рішення для боротьби з погіршенням зору - це обмежити кількість часу, що проводиться за комп'ютером без перерви. Ідеальною „розрядкою" може бути фізична активність, що не вимагає напруги зору.

Важливо прийняти заходи по зменшенню віддзеркалень від монітора. Яскраве освітлення може викликати неприємні віддзеркалення на екрані. Можливі спроби вирішення цієї проблеми полягають у повороті монітора так, щоб ні прямо перед ним. ні ззаду не було яскравих джерел світла. Можна порадити спектральні окуляри.

У поле зору користувача не повинні потрапляти освітлювальні прилади, тому що це посилює навантаження на очі.

Не можна сидіти за комп’ютером у сутінках або темряві.

Якщо щось потрібно передрукувати з паперу, то аркуші слід розмістити якомога ближче до екрана, наприклад на кронштейні поруч з екраном, щоб зменшити „розсіювання" погляду.

Бажано періодично вставати з-за комп’ютера і дивитись у вікно, вдалечінь - на небо, траву, дерева. А ще кілька разів протягом робочого дня робити своєрідну виробничу гімнастику для очей. Наприклад, відвернутися від монітора, заплющити очі і швидко – 10 20 разів примружитися, не розтуляючи повік, потім інтенсивно покліпати, якомога швидше розплющуючи очі. Заплющити очі і „намалювати" ними вісімку, спочатку вертикальну, потім – горизонтальну. Промасажувати скроні біля кутиків очей круговими рухами кінчиків пальців – пальцями правої руки потрібно робити рухи за годинниковою стрілкою, а пальцями лівої – проти.

Бажано використовувати прийоми „далекозорості", які розслабляють м'язи кришталиків. Для цього треба зафіксувати погляд на найвіддаленішій точці в межах видимості, а потім плавно перевести погляд на кінчик власного носа. Або приклеїти до найближчої шибки паперову мітку, потім по черзі дивитися то на неї, то на дерево за вікном Повторювати вправу бажано 10 - 15 разів, щонайменше тричі на день.

Людям із напруженим режимом зорової роботи або ослабленим зором треба споживати продукти які зміцнюють судини сітки ока: чорницю, чорну смородину, шипшину, журавлину, моркву. У їхньому раціоні мають бути також печінка тріски та зелень – петрушка. салат, кріп, зелена цибуля. Можна приймати готові препарати які містять чорницю, каротиноїди, селен, цинк, мідь, аскорбінову кислоту, токофероли, а також полівітаміни.

Проблеми пов'язані з м'язами і суглобами. У людей, що заробляють на життя роботою на комп’ютерах, найбільше число скарг на здоров’я пов’язані із захворюванням м’язів і суглобів. Найчастіше це просто оніміння шиї, біль в плечах і попереку або колення в ногах. Але бувають, проте, і серйозні захворювання. Найбільш поширений кистьовий тунельний синдром, при якому нерви руки ушкоджуються унаслідок частої і тривалої роботи на комп’ютері.

Біль в руках, особливо в гроні правої руки, викликана довгою роботою за комп’ютером придбала назву тунельного синдрому або синдрому зап’ястного каналу, а так само придбала статус професійного захворювання комп’ютерників. Причиною виникнення болю є затискання нерва в зап’ястному каналі.

Затискання може бути викликане розпуханням сухожиль що проходять в безпосередній близькості до нерва, а так само, розпухання самого нерва. Причиною затискання нерва є постійне статичне навантаження на одні й ті ю м’язи, яке може бути викликане великою кількістю одноманітних рухів (головне, щоб мишка лягла в руку. А це залежить від розміру кисті, довжини пальців, положенні рук на столі) або незручними положеннями рук, під час роботи з клавіатурою, при якому зап’ястя знаходиться в постійній напрузі. Все це може привести до постійного відчуття болю або дискомфорту в руках, ослабленню і онімінню рук, особливо долонь.

На допомогу користувачам приходить ергономіка. Ця наука вивчає способи взаємодії людини з комп’ютером та пристроями, які до нього додаються, а також способи оптимальної організації робочого місця.

У документації до «мишок» і клавіатур від провідних виробників постійно наголошується на ризику виникнення професійних захворювань через неправильний вибір маніпулятора або неправильну роботу з ним.

Результати трирічного дослідження груп користувачів проведені американськими вченими, свідчать, що найголовніша умова для збереження опорно-рухового апарату здоровим – правильне сидіння на робочому місці.

Ось їхні основні рекомендації:

- Клавіатура неодмінно має міститися нижче ліктя;
- руки не повинні висіти – вони мають цілком спиратися на підлокітники крісла або робочий стіл, але не на край столу;
- голова має бути нахилена трохи вперед - Тривалість безперервної роботи з комп’ютером не повинна перевищувати двох годин з 10 – 15 хвилинними перервами;
- під час перерв слід виконувати рад вправ для рук.

Дуже важливо правильно вибирати собі маніпулятори, так би мовити по руці. Що стосується клавіатури, то фахівці заявляють, що її класичний „прямий" варіант - пережиток часів друкарських машинок, а людині, яка сидить за комп'ютером весь день треба користуватись сучасною ергономічною клавіатурою, тобто з розворотом двох блоків відносно один одного із „горбом".

Стрес при втраті інформації, депресія і інші нервові розлади, викликані впливом комп’ютера на психіку людини. Тривала робота за комп'ютером негативно позначається на здоров’ї, що позначається на психіці, крім того вона ще і зв'язана з постійним роздратуванням, джерелом якого можуть бути різні ситуації, наприклад зависання комп'ютера, втрата не збереженої інформації, проблеми з роботою програм і т.д.

За наслідками досліджень, стресові ситуації пов’язані з комп’ютером, а особливо з Internet приводять до збільшення споживання спиртних напоїв. Таким чином, ми отримуємо або психічну неврівноваженість або алкоголізм або все разом.

Японські і англійські лікарі б’ють тривогу. Серед їх пацієнтів збільшилася кількість скарг на порушення функцій пам’яті. Дослідження проведені ученими однієї з японських клінік серед пацієнтів у віці від 20 до 77 35 років, показали, що нинішнє покоління, виховане на всіляких пристроях «зовнішньої пам’яті», втрачає здатність запам’ятовувати нове, згадувати старе, а також виділяти з величезного об’єму інформації необхідні відомості.

Спілкування з комп’ютером, особливо з ігровими програмами, супроводжується сильною перевтомою, оскільки вимагає швидкої у відповідь реакції. Короткочасна концентрація нервових процесів викликає сильне стомлення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи на тему “ Інтеграція та управління бізнес акаунтом facebook в мобільному додатку на прикладі “Easy Accounting for Social Biz””, був отриманий значний досвід та знання у створенні програмного забезпечення для мобільних пристроїв. Розробка даного функціоналу дозволила закріпити навички роботи з сучасними технологіями та інструментами, що використовуються в індустрії розробки як мобільних додатків так і серверів.

Були освоєні та застосовані різні методики тестування. Досвід роботи над кваліфікаційною роботою дав змогу закріпити навички розробки клієнтської, а також серверної програмного забезпечення.

Розробка інтеграції Facebook API в мобільний додаток посприяла поглибленню знань про основні принципи створення програмного забезпечення, такі як проєктування архітектури, організація коду, використання патернів проєктування та принципів чистої архітектури. Закріплення цих навичок та знань стане основою для подальшого розвитку у сфері розробки мобільних додатків та серверів і забезпечить глибоке розуміння процесу створення якісного програмного забезпечення.

Отже, розробка кваліфікаційної роботи принесла не лише практичний досвід у створенні програмного забезпечення, але й додала цінності до академічного та професійного розвитку. Отримані навички, знання та досвід покладають підставу для успішної кар'єри розробника програмного забезпечення та сприятимуть подальшому розвитку у цій галузі.

ПЕРЕЛІК ДЖЕРЕЛ

1 Офіційна документація GraphQL: веб-сайт. URL: https://graphql.org/learn

2 Офіційна документація Hasura. URL: https://hasura.io/docs

3 Офіційна документація Docker. URL: https://hub.docker.com/_/docker-docs

4 Клієнт-серверна архітектура з використанням ресурсного серверу Hasura. URL: https://hasura.io/blog/hasura-for-existing-graphql-rest-apis-architect-guide

5 Механізм підписок(Subscription) у базі даних PostgreSQL та серверу Hasura. URL: https://hasura.io/docs/latest/subscriptions/postgres/index

6 Mutation запити до серверу Hasura. URL: https://hasura.io/learn/graphql/intro-graphql/graphql-mutations

7 Webhook mode Hasura. URL: https://hasura.io/learn/graphql/hasura-authentication/webhook-mode

8 Guide to Implementing BLoC Architecture in Flutter – Wednesday Engineering Blog URL: https://www.wednesday.is/writing-articles/guide-to-implementing-bloc-architecture-in-flutter

9 Instagram Message API for Instagram. URL: https://developers.facebook.com/products/messenger/messenger-api-instagram

10 Karate Framework Tutorial: Automated API Testing With Karate. URL: https://www.softwaretestinghelp.com/api-testing-with-karate-framework

11 Docker compose up | Docker documentation. URL: https://docs.docker.com/engine/reference/commandline/compose_up

12 Testing Hasura GraphQL APIs with Karate. URL: https://hasura.io/blog/testing-hasura-graphql-apis-with-karate

13 Огляд архітектури Flutter URL: https://docs.flutter.dev/resources/architectural-overview

14 Dart Futures and Streams – Kodeco URL: https://www.kodeco.com/32851541-dart-futures-and-streams

15 Офіційна документація Facebook URL: https://developers.facebook.com/docs/graph-api/get-started

16 Сторінка Meta Bussines Suite на платформі Play Market, URL: https://play.google.com/store/apps/details?id=com.facebook.pages.app

17 Сторінка WatsApp Bussines на платформі Play Market, URL: https://play.google.com/store/apps/details?id=com.watsapp.w4b

18 В.І. Голінько, М.Ю. Іконніков, Я.Я. Лебедєв, «Охорона праці в галузі інформаційних технологій»

19 Пойда В.Ю, Курс лекцій «Охорона праці в галузі»

20 Клієнт-серверна архітектура з використанням ресурсного серверу Hasura. URL: https://hasura.io/blog/hasura-for-existing-graphql-rest-apis-architect-guide/

21 Hasura GraphQL Engine Permissions and Roles Examples. URL: https://hasura.io/docs/latest/auth/authorization/permissions/common-roles-auth-examples/

22 Introduction to virtual tables to PostgreSQL. URL: https://hasura.io/docs/latest/schema/postgres/postgres-guides/views/

23 Zaccagnino C. Programming Flutter: Native, Cross-Platform Apps the Easy Way. 1st Ed. North Carolina : Pragmatic Bookshelf, 2020. 370 с.

24 Огляд мови Dart – Dart.dev URL: https://dart.dev/overview

25 Kadavy, D., Kagan, N. Design for Hackers. Berkeley: New Riders, 2011. 352 с.

26 OAuth 2 Simplified - Aaron Parecki URL: https://aaronparecki.com/oauth-2-simplified

27 Napoli, M. L. Beginning Flutter: A Hands On Guide to App Development. 1st Edition. Birmingham: Wrox, 2019. 528 с.

28 Dart & Flutter: What Does copyWith Do? URL: https://developer.school/tutorials/dart-flutter-what-does-copywith-do

29 Zakas N. Android Security Internals: An In-Depth Guide to Android's Security Architecture. San Francisco: No Starch Press, 2014. 432 с.

30 Miller, C., Blazakis D., DaiZovi D. iOS Hacker's Handbook. Indianapolis: Wiley, 2012. 408 с.

31 Biessek A. Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2. Birmingham: Packt Publishing, 2020, 512 с.

ДОДАТКИ

Додаток А

Лістинг файлу «insta_bussiness_account.dart»

```
import 'package:hive/hive.dart';
import 'package:json_annotation/json_annotation.dart';

part 'insta_business_account.g.dart';

@JsonSerializable()
@HiveType(typeId: 6)
class InstaBusinessAccount {
  @HiveField(0)
  final String id;
  @HiveField(1)
  final String name;
  @HiveField(2)
  @JsonKey(name: "profile_picture_url")
  final String? profilePictureUrl;
  @HiveField(3)
  @JsonKey(name: "username")
  final String userName;

  InstaBusinessAccount(
      {required this.id,
      required this.name,
      this.profilePictureUrl,
      required this.userName});

  factory InstaBusinessAccount.fromJson(Map<String, dynamic> json) =>
      _$InstaBusinessAccountFromJson(json);

  Map<String, dynamic> toJson() => _$InstaBusinessAccountToJson(this);
}
```

Додаток Б

Лістинг файлу «facebook_repository.dart»

```
import 'package:flutter_facebook_auth/flutter_facebook_auth.dart';
import 'package:inst_accountant/api/artemis_client.dart';
import'package:inst_accountant/api/facebook_repository/models/fb_login/fb
_user_page.dart';
import'package:inst_accountant/api/facebook_repository/models/fb_login/fb
_user_pages_response.dart';
import'package:inst_accountant/api/facebook_repository/models/insta_conve
rsations/insta_message.dart';
import'package:inst_accountant/api/facebook_repository/models/instagram_m
edia_response.dart';
import 'package:inst_accountant/api/graphql/graphql_api.dart';

enum FBAuthenticationStatus {
  unknown,
  authenticated,
  unauthenticated,
}

abstract class FacebookRepository {
  final ArtemisClient _client;

  ArtemisClient get client => _client;

  const FacebookRepository({required ArtemisClient client}) : _client =
client;

  Stream<FBAuthenticationStatus> get status;

  Stream<List<InstaStorageMessageResponse>> get messages;

  Future<LoginStatus> loginViaFacebook(
```

```
      {LoginBehavior loginBehavior});

  Future<FBUserPagesResponse?> getFBUserPages({int limit = 20, String?
next});

  Future<void> chooseFBUserPage(FBUserPage fbUserPage);

  Future<FBUserPage?> getChosenFBUserPage();

  Future<void> deleteFacebookBoxesValues();

  Future<InstaBusinessAccountResponse> getInstagramBusinessAccount();

  Future<InstagramMediaResponse> getInstagramMedias(
      {int limit = 20, String? next});

  Future<InstagramConversationResponse> getInstagramConversations(
      {int limit = 20, String? nextUrl});

  Future<InstaMessagesResponse> getInstagramMessages(
      {required String? conversationId, int limit = 20, String? next});

  Future<void> logOut();

  Future<InstaBusinessAccount?> get instaBusinessAccount;

  Future<String?> getInstagramUserProfilePicUrl({required String
userId});

  Future<String?> get instagramBusinessId;

  void dispose();

  Future<void> sendMessage(
      {required String recipientId, required String message});
```

```
  Future<void> sendReceiptAsImageMessage({required String
recipientId,required String receiptUrl});

  Future<void> subscribeToMessages(
      {required String socialId, required String senderId});

  Future<List<InstaStorageMessageResponse>>
getInstagramMessagesFromStore(
      {required String recipientInstaId, required List<String>
messagesIds});

  void unSubscribeToMessages();

  Future<bool> get isMessagePermissionGranted;

  Future<bool> get isLoginPermissionGranted;

  Future<bool> get isPostsPermissionGranted;

  Future<bool> get isSignedToInstagram;

  Future<bool> get isInstagramBasicGranted;
}
```

Додаток В

Лістинг файлу «subscriptions.graphql»

```
subscription messages($recipient_insta_id: String, $limit: Int = 20,
$offset: Int = 0) {
    insta_messages(where: {_and: {recipient_insta_id: {_eq:
$recipient_insta_id}}, }, limit: $limit, offset: $offset) {
        id
        message
        message_id
        recipient_insta_id
        created_at
    }
}

subscription messages_stream($recipient_insta_id: String,
$sender_insta_id: String, $created_at: timestamptz) {
    insta_messages_stream(batch_size: 10, cursor: {initial_value:
{created_at: $created_at}}, where: {_and: {recipient_insta_id: {_eq:
$recipient_insta_id}, sender_insta_id: {_eq: $sender_insta_id}}}) {
        id
        message
        message_id
        recipient_insta_id
        created_at
        deleted_at
    }
}
```

Додаток Г

Налаштування webhooks

```
// Adds support for GET requests to our webhook
app.get('/webhook', (req, res) => {
  console.log("Got GET /webhook ");
  // Your verify token. Should be a random string.
  const VERIFY_TOKEN = process.env.VERIFY_TOKEN;

  // Parse the query params
  let mode = req.query['hub.mode'];
  let token = req.query['hub.verify_token'];
  let challenge = req.query['hub.challenge'];

  // Checks if a token and mode is in the query string of the request
  if (mode && token) {

    // Checks the mode and token sent is correct
    if (mode === 'subscribe' && token === VERIFY_TOKEN) {

      // Responds with the challenge token from the request
      console.log('WEBHOOK_VERIFIED');
      res.status(200).send(challenge);

    } else {
      // Responds with '403 Forbidden' if verify tokens do not match
      res.sendStatus(403);
    }
  }
});

// Creates the endpoint for your webhook
app.post('/webhook', (req, res) => {
    let body = req.body;
```

```
    console.log(`\u{1F7EA} Received webhook:`);
    console.dir(body, { depth: null });

    // Check if this is an event from a page subscription
    if (body.object === "instagram") {

      // Return a '200 OK' response to all requests
      res.status(200).send("EVENT_RECEIVED");

      // Iterate over each entry - there may be multiple if batched
      body.entry.forEach(async function(entry) {

        if (!("messaging" in entry)) {
           console.warn("No messaging field in entry. Possibly a webhook
test.");
          return;
        }

        // Iterate over webhook events - there may be multiple
        entry.messaging.forEach(async function(webhookEvent) {
          console.log(`\u{1F7EA} Parsed insta message`);
          // Discard uninteresting events
          if (
            "message" in webhookEvent &&
            webhookEvent.message.is_echo === true
          ) {
            console.log("Got an echo");
            return;
          }

          // Get the sender IGSID
          let recipient_insta_id = entry.id;
          let sender_insta_id = webhookEvent.sender.id;
          let message_id = webhookEvent.message.mid;
          let message = JSON.stringify(webhookEvent);
```

```
          console.log(`\u{1F7EA} Before saving ` + recipient_insta_id + sender_insta_id + ' ' + message_id + ' ' + message);
          // execute the Hasura operation
          const { data, errors } = await execute({ recipient_insta_id, sender_insta_id, message_id, message }, HASURA_OPERATION);

          // if Hasura operation errors, then throw error
          if (errors) {
                console.warn("Got an error while saving message!" + JSON.stringify(errors[0]));
            return;
          }

          return;
        });
      });
    } else if (body.object === "page") {
      // Catch if the event came from Messenger webhook instead of Instagram
      console.warn(
        `Received Messenger "page" object instead of "instagram" message webhook.`
      );
      res.sendStatus(404);
    } else {
      // Return a '404 Not Found' if event is not recognized
      console.warn(`Unrecognized POST to webhook.`);
      res.sendStatus(404);
    }
});

const HASURA_OPERATION = `
mutation MyMutation($recipient_insta_id: String, $sender_insta_id: String, $message_id: String, $message: String) {
          insert_insta_messages_one(object:         {recipient_insta_id: $recipient_insta_id,   sender_insta_id:   $sender_insta_id,   message_id: $message_id, message: $message}) {
```

```
    id
  }
}
`;

// Subscribe to Instagram messages
app.post('/subscribe_to_insta_page', (req, res) => {
  // Send the HTTP request to the Messenger Platform
  const { page_id, page_access_token } = req.body.input;
  request({
    'uri': `https://graph.facebook.com/${page_id}/subscribed_apps`,
        'qs':   {   'subscribed_fields':   'messages',   'access_token':
page_access_token },
    'method': 'POST',
  }, (err, _res, _body) => {
    if (!err) {
      return res.status(200).json({
        message: "Subscribed to messages!"
      });
    } else {
      return res.status(400).json({
        message: 'Unable to subscribe to messages:' + err
      });
    }
  });
});
```

Додаток Ґ

Перша теза до кваліфікаційної роботи

УДК 004.4

В. О. Крушельницький
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

Інтеграція та управління бізнес акаунтом Facebook *та Instagram* в мобільному додатку на прикладі “Easy Accounting for Social Biz”

V. O. Krushelnystkyi

Integration and management of Facebook and Instagram business accounts in the mobile application using the example of "Easy Accounting for Social Biz"

Розглянемо процес інтеграції Facebook API зі сторони клієнта. Функції для отримання данних зі сторони клієнта можна поділити на 2 категорії: функції взаємодії із нашим сервером і функції для взаємодії із Facebook API. Функції першої категорії будуть виконувати роль підписки на WebHooks, які ми реалізовуєм на сервері. Функції другої категорії будуть реалізовані за допомогою сервісу FacebookSDK, який значно спростить задачу, надавши готовий макет запиту. До цих функцій належать: отримання токену авторизації в Facebook, отримання доступу до сторінки Instagram, прив’язаної до Facebook, отримання списку клієнтів та інформація про них, отримання списку товарів та їхній опис.

Вихідна інформація від Facebook API може відрізнятись, в залежності від запиту. Тому всю необхідну інформацію ми будемо зберігати в локальній базі данних – Hive. Її функціонал являє собою зберігання данних за системою: назва боксу > ключ боксу. Таким чином ми зможемо розділити необхідні нам данні і оптимізувати доступ до них. Періодичність надходження залежить від кількості операцій пов’язаних з обліковим записом Facebook з нашого мобільного додатку.

Частина звязку із джерелами данних, розроблена в окремих класах «репозиторіях». Саме в цих класах ми і будемо писати наші інтеграційні запити, а також будемо звязуватись із локальною базою данних. Все що буде необхідно для бізнес логіки – це отримати доступ до класу- репозиторію. Цей клас буде розроблено за допомогою паттерну «Singleton»

Висновок. Розглянуто розробку інтеграціїFacebook API в мобільний додаток із сторони клієнта. Реалізовано функціонал взаємодії.

Література

1. Zaccagnino C. Programming Flutter: Native, Cross-Platform Apps the Easy Way. 1st Ed. North Carolina : Pragmatic Bookshelf, 2020. 370 c.
2. Zakas N. Android Security Internals: An In-Depth Guide to Android's Security Architecture. San Francisco: No Starch Press, 2014. 432 c.
3. Guide to Implementing BLoC Architecture in Flutter – Wednesday Engineering Blog https://www.wednesday.is/writing-articles/guide-to-implementing-bloc-architecture-in-flutter
4. Kadavy, D., Kagan, N. Design for Hackers. Berkeley: New Riders, 2011. 352 c.
5. Огляд мови Dart – Dart.dev. https://dart.dev/overview
6. Dart Futures and Streams – Kodeco. https://www.kodeco.com/32851541-dart-futures-and-streams
7. Огляд архітектури Flutter. https://docs.flutter.dev/resources/architectural-overview

Додаток Д

Друга теза до кваліфікаційної роботи

УДК 004.4

В. О. Крушельницький
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

Інтеграція та управління бізнес акаунтом Facebook *та Instagram* в мобільному додатку на прикладі “Easy Accounting for Social Biz”

V. O. Krushelnystkyi

Integration and management of Facebook and Instagram business accounts in the mobile application using the example of "Easy Accounting for Social Biz"

Розглянемо процес інтеграції Facebook API зі сторони сервера. Робота серверної частини обробляється Hasura GraphQL Engine. Спочатку потрібно визначити схему GraphQL, яка описує доступні типи даних та запити. Схема визначає поля, які можна запитувати, а також способи отримання даних. Після визначення схеми можна створити запит, який відповідає необхідним потребам. Запит до серверу вказується мовою запитів GraphQL.

Після створення запиту потрібно надіслати його на сервер GraphQL Engine. Зазвичай використовуються HTTP-запити, такі як POST або GET. Сервер GraphQL Engine отримує запит та обробляє його згідно з визначеною схемою GraphQL. Він перетворює стрічку зі скриптом на мові GraphQL у звичайний SQL запит та звертається до бази даних або іншого джерела даних, отримує результат та формує відповідь.

Після обробки сервер GraphQL Engine повертає відповідь на клієнтську частину у вигляді об’єкту json. Відповідь може містити дані, які запитувались, або повідомлення про помилку, якщо запит був недійсним або виникла проблема під час обробки. Після отримання відповіді можна обробити дані згідно з потребами. Відповідно є можливість використовувати отримані дані для відображення на користувацькому інтерфейсі або подальшої обробки.

Використовувати Hasura GraphQL Engine ми будемо тільки для того, щоб зберігати данні про переписку користувача із клієнтами. Вхідні дані будуть однакові із вихідними(id клієнта, id користувача, дата повідомлення, текст повідомлення), оскільки немає причин зберігати лишню інформацію.

Для реалізації передачі повідомлень в реальному часі між клієнтом та продавцем використано механізм підписок (subscription) у GraphQL та механізм WebHooks. Підписки дозволять клієнтській частині підписатися на певні події або оновлення даних та отримувати повідомлення в реальному часі при виникненні цих подій. WebHooks, у свою чергу, дозволяють серверу надсилати дані до клієнтської частини, щоб повідомляти про події у режимі реального часу.

Висновок. Розглянуто розробку інтеграції Facebook API в мобільний додаток зі сторони серверу. Реалізовано функціонал взаємодії.

Література

8. Офіційна документація GraphQL: веб-сайт. URL: https://graphql.org/learn
9. Офіційна документація Hasura. https://hasura.io/docs
10. Клієнт-серверна архітектура з використанням ресурсного серверу Hasura. https://hasura.io/blog/hasura-for-existing-graphql-rest-apis-architect-guide/
11. Механізм підписок(Subscription) у базі даних PostgreSQL та серверу Hasura. URL: https://hasura.io/docs/latest/subscriptions/postgres/index/
12. Webhook mode Hasura. URL: https://hasura.io/learn/graphql/hasura-authentication/webhook-mode/ Instagram Message API for Instagram. URL: https://developers.facebook.com/products/messenger/messenger-api-instagram/