Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ОФОРМЛЕННЯ ЗАМОВЛЕННЯ ТА ОПЛАТИ В ЗАКЛАДАХ ХАРЧУВАННЯ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-2
Цехмістренко К.В.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Кобилін О.А.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Цехмістренко Катерині Василівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка вебзастосунку для оформлення замовлення та оплати в закладах харчування

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 28 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, офіційна документація мови Kotlin, офіційна документація мови JavaScript.

4. Перелік питань, що потрібно опрацювати в роботі
1. Огляд підходів для організації проєктування сучасних вебзастосунків.
2. Організація взаємодії між інтерфейсом користувача та адміністративною частиною застосунку.
3. Постановка задачі та її реалізація.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми замовлення та розрахунку в закладах харчування, постановка задачі, реалізація.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-12.04.24	
3	Аналіз літератури з досліджуваної проблеми	12.04.24-23.04.24	
4	Аналіз технічних засобів	24.04.24-26.04.24	
5	Постановка задачі	28.04.24-01.05.24	
6	Програмна реалізація	02.05.24-20.05.24	
7	Оформлення пояснювальної записки	21.05.24-26.05.24	
8	Перевірка на плагіат	27.05.24	
9	Рецензування	28.05.24	
10	Підготовка презентації та доповіді	29.05.24-02.06.24	
11	Занесення роботи в електронний архів	06.06.24	
12	Попередній захист кваліфікаційної роботи	06.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Кобилін О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 67 с., 1 табл., 22 рис., 30 джерел.

АВТОМАТИЗАЦІЯ ПРОЦЕСІВ, ВЕБЗАСТОСУНОК, ЕЛЕКТРОННЕ МЕНЮ, VISUAL STUDIO CODE, FIREBASE, JAVASCRIPT, HTML, CSS, KOTLIN.

Об’єктом роботи є система оформлення та оплати замовлень для закладів харчування.

Метою роботи є розробка вебзастосунку для зручності замовлення та сплати онлайн та мобільного застосунку для статистики.

Проведено дослідження, в рамках якого було проаналізовано існуючі сервіси для прийому замовлення та оплати. Розглянуто можливі засоби для реалізації об’єкту роботи у вигляді вебзастосунку на базі JavaScript та Firebase з використанням допоміжних бібліотек та фреймворків.

У результаті роботи здійснена програмна реалізація вебзастосунку та мобільного застосунку для оформлення замовлення та оплати в закладах харчування.

AUTOMATION OF PROCESSES, WEB APPLICATIONS, ELECTRONIC MENU, VISUAL STUDIO CODE, FIREBASE, JAVASCRIPT, HTML, CSS, KOTLIN.

The object of the work is a system for placing and paying for orders for catering establishments.

The purpose of the work is to develop a web application for the convenience of ordering and paying online and a mobile application for statistics.

A study was conducted to analyze existing services for ordering and payment. Possible means for realizing the object of work in the form of a web application based on JavaScript and Firebase with the use of auxiliary libraries and frameworks are considered.

As a result of the work, the software implementation of a web application and a mobile application for ordering and payment in catering establishments was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

QR – двовимірний штрих-код

Front-end – візуальна частина проєкту

Back-end – частина програмного забезпечення, яка працює на сервері і відповідає за логіку додатку

ISO (International Organization for Standardization) – це міжнародна організація, яка розробляє та публікує міжнародні стандарти

HTML (HyperText Markup Language) – це стандартна мова розмітки, яка використовується для створення веб-сторінок і веб-додатків

CSS (Cascading Style Sheets) – це мова стилів, яка використовується для опису зовнішнього вигляду HTML-документів

Cloud Firestore – гнучка, масштабована база даних NoSQL

СУБД – система управління базами даних

NoSQL (Not Only SQL) – тип систем управління базами даних

HTTP API (Hypertext Transfer Protocol Application Programming Interface) – набір визначених правил, які дозволяють програмам взаємодіяти один з одним через протокол HTTP

MVI – архітектурний патерн у розробці програмного забезпечення

Serverless – модель обчислень з динамічним керуванням виділення машинних ресурсів

ВСТУП

Цифрова трансформація значно покращила клієнтський сервіс. Наприклад, використання QR-кодів для реклами, донесення інформації та покращення інтерактивності вже стало стандартом.

QR-код являє собою вдосконалену версію всім нам відомого штрих-коду. Даний винахід можна використовувати для багатьох видів даних, а його популярність настала завдяки легкості розпізнавання подібного шифру камерою телефона. Тож, здавалося б найпростішим способом для усунення паперового меню є варіант надання його у вигляді фото за допомогою коду. Але передача зображень таким методом не прижилася у загальному вжитку.

В статтях про інновації в ресторанному світі за 2013 рік розглядалися звичні для нас зараз QR-коди, як спосіб рекламування та додавання інтерактивності, а електронні меню, як інструмент для розв’язання проблем з помилками при замовленні або зменшення ризиків виникнення конфліктних ситуацій між гостем та персоналом тощо. Та вже у 2019-2020 роках, за часів пандемії відбувається різка зміна в підході до ведення бізнесу – якщо до цього вище згадані нововведення використовувалися одиницями, то тепер це стає необхідністю, яка допомагає втриматися на ринку. Після ослаблення карантинних обмежень у клієнтів лишився запит на зменшення ризиків для власного здоров’я, крім того, вони оцінили зручність використання QR-кодів, які містять посилання на сайт з меню.

Актуальність роботи полягає в тому, що такий вид електронного меню є безпечнішим для клієнта, що контактує з власним телефоном, а також універсальним для бізнесу. Більше не потрібно робити фото меню для розміщення в соціальних мережах та на гугл мапах, постійно передруковувати його через зміну цін, інгредієнтів чи з будь-яких інших причин. Все що потрібно клієнту – лишити посилання на сторінці, мапах та на столиках у вигляді QR [1].

Варто зазначити, що онлайн формат дозволяє клієнту більш досконало вивчити меню, яке знаходиться завжди у відкритому доступі. Відкритість та доступність інформації також є великим плюсом, оскільки сайт вміщає в собі набагато більше інформації, яка може знадобитися відвідувачу для того, щоб зробити вибір. Наприклад, перелік алергенів що містить певна страва або опис застосунків якими можна кастомізувати свою їжу.

Використання подібних інновацій дозволяє збільшити кількість даних, що можуть знадобитися при аналізі. Умовно, при розробці нового меню, можна зважати не лише на найпопулярніший продукт, а і на «сіру зону», продукцію, якою люди цікавилися, проте не замовляли, та модифікувати її.

Вебзастосунок реалізує оновлений підхід до розробки інтерфейсу користувача, наприклад, інтуїтивно зрозумілі процеси за допомоги принципів UI/UX [2].

1 АНАЛІЗ ІСНУЮЧИХ ВЕБЗАСТОСУНКІВ ДЛЯ ОФОРМЛЕННЯ ТА ОПЛАТИ ЗАМОВЛЕННЯ В РЕСТОРАНІ ДЛЯ ПОСТАНОВКИ ЗАДАЧІ, СПОСОБИ РЕАЛІЗАЦІЇ

1.1 Огляд вебзастосунків для оформлення та оплати замовлення в ресторані

Діджіталізація змінила підхід до розробки бізнес-моделі та відкрила нові можливості та канали для створення прибутку. Для підвищення конкурентоспроможності компаніям постійно необхідно шукати та створювати нові рішення.

Основними стратегічно важливими факторами у підвищенні ефективності, є процеси оформлення та оплати замовлень. Сьогодні існує широкий вибір вебзастосунків, спрямованих на полегшення цих процесів і покращення користувацького досвіду. Сьогодні на ринку існує два типи реалізації меню.

Перший тип вебзастосунків – це платформи замовлення їжі, які працюють на основі вебсайтів або мобільних застосунків. Такі сервіси дозволяють клієнтам з легкістю переглядати меню, але робити замовлення за допомоги офіціанта.

Другий тип – це комплексні рішення, які поєднують у собі функції замовлення, оплати, подекуди доставлення та управління клієнтською базою. Ці засоби дозволяють ресторанам не лише приймати замовлення, а й ефективно керувати усім процесом обслуговування клієнтів з погляду автоматизації.

Окрім зручності для клієнтів, вебзастосунки для оформлення та оплати замовлень в ресторанах пропонують власникам бізнесу значні переваги. Вони дозволяють оптимізувати роботу персоналу, зменшуючи помилки та підвищуючи ефективність обслуговування [3]. Такі інструменти також

дозволяють збирати аналітику про замовлення та взаємодію з клієнтами, що допомагає у плануванні бізнес-стратегії та збільшенні лояльності клієнтів.

1.1.1 OddMenu

OddMenu – це сервіс для створення електронного меню за QR-кодом, що приносить значну користь для ресторанів, кафе та барних бізнесів, а також для їх клієнтів. Платформа пропонує широкий набір функцій для оптимізації роботи з меню, включаючи можливість редагування, додавання фото, створення меню декількома мовами та генерацію QR-кодів [4].

Сервіс спрямований на покращення якості обслуговування та зручності користувачів в закладах харчування. Для клієнтів, OddMenu - це сучасний спосіб ознайомлення з меню через QR-код, що не вимагає встановлення застосунків на їхні телефони. Це робить процес замовлення та ознайомлення з асортиментом більш зручним і швидким.

Для власників ресторанів, кафе та барів, OddMenu стає платформою для покращення взаємодії з клієнтами та збільшення обсягів продажів. Вони можуть легко редагувати та оновлювати своє меню, додавати акції та інші актуальні дані без зайвих витрат часу та ресурсів.

Використання цифрового QR-меню допомагає підвищити рівень задоволеності клієнтів, оскільки вони можуть легко знайти інформацію про страви та залишити свої відгуки безпосередньо з меню. Фотографії та детальний опис страв, а також зручність замовлення через QR-коди, спонукають гостей до більшого обсягу замовлень та підвищують середній чек.

OddMenu дозволяє ефективно керувати меню без великих витрат на оновлення або покупку дорогих терміналів. Це сприяє економіці часу та фінансів для власників закладів.

Узагальнюючи, сервіс OddMenu відкриває нові можливості для закладів, допомагаючи їм підвищувати якість обслуговування, залучати нових клієнтів та збільшувати обсяги продажів за допомогою створення цифрового меню на базі їх середовища.

1.1.2 Expirenza

Expirenza – це інноваційний сервіс, який забезпечує зручну оплату в закладах громадського харчування за допомогою QR-кодів. Клієнти можуть не лише оплачувати рахунок, але й залишати чайові, писати відгуки, переглядати меню та користуватися іншими зручностями [5].

На квітень 2024 року Expirenza успішно приєднала до своєї платформи понад 1300 закладів по всій Україні, де працює понад 28 000 офіціантів і 3000 кур'єрів. Заклади розташовані у різних містах, включаючи Київ, Харків, Одесу та Львів.

Основні переваги використання Expirenza:

– сервіс пропонує привабливі умови для підприємців, зберігаючи комісію на рівні 1.3% для українських карток і 2% для іноземних. Встановлення та обслуговування сервісу абсолютно безплатні;

– Expirenza дозволяє вам створити цифрове меню без зусиль. Ви можете вибрати один з пропонованих форматів або завантажити свої фотографії, додати опис та вказати вартість. Менеджери сервісу допоможуть вам з цим або зроблять це за вас;

– гості можуть швидко оплатити свій рахунок, сканувавши QR-код і обравши опцію «Оплата». Це спрощує процес оплати та дозволяє швидше обслуговувати клієнтів;

– система Expirenza автоматично розподіляє гроші за бар, кухню та чайові на відповідні рахунки. Власники закладів можуть отримувати оцінки та зворотний зв’язок від гостей для поліпшення обслуговування;

– платформа надає особистий кабінет з аналітичними засобами для керування банківським рахунком, відгуками гостей та аналізу діяльності;

– Expirenza дозволяє створити електронне меню з можливістю приймати замовлення безпосередньо з нього.

Загалом, Expirenza створює інноваційне та зручне середовище для клієнтів та власників закладів харчування, спрощуючи процес оплати та забезпечуючи покращення якості обслуговування.

1.1.3 ChoiceQR

ChoiceQR – це сучасне рішення для ресторанів, яке дозволяє створювати цифрове меню. Цей сервіс забезпечує гостям миттєвий доступ до всіх пропозицій закладу з яскравими фотографіями, відео та детальним описом страв. Крім того, ChoiceQR можна використовувати як сайт-візитку, де гості можуть знайти робочі години, контактні дані, пароль від Wi-Fi та інші корисні відомості. Усі елементи цифрового меню інтерактивні, що дозволяє гостям легко знаходити необхідну інформацію і залишати відгуки безпосередньо через цей сервіс [6].

У меню вже вбудовані інструменти для підвищення продажів, такі як «Щасливі години», «Бізнес-ланчі» та рекламні банери з новинками або акціями. ChoiceQR гнучко налаштовується під ваш бренд та стиль. Меню виглядає органічно та зручно для гостей, допомагаючи їм запам’ятати ваш бренд.

Сервіс дозволяє збирати відгуки від клієнтів безпосередньо в меню. Це дозволяє вам оперативно реагувати на негатив і уникати поганих відгуків у соцмережах.

Гості можуть додавати страви до «Списку бажань» для зручності та економії часу. Це сприяє збільшенню середнього чека, оскільки клієнти замовляють більше, а персонал витрачає менше часу на обслуговування.

ChoiceQR надає зручну панель адміністратора з інтуїтивним інтерфейсом та ключовими даними зі статистики для ефективного управління.

Загально, ChoiceQR є потужним інструментом для створення ефективного цифрового меню, яке допомагає підвищити продажі, поліпшити взаємодію з клієнтами та збільшити впізнаваність вашого бренду.

1.2 Форма реалізації цифрового меню

У цьому розділі розглянемо можливі методи для реалізації не паперового меню. Існує безліч варіантів відходу від паперових меню, наприклад, планшети та інтерактивні екрани.

Крім вже зазначеного QR-коду, меню може бути доступне через технологію NFC (Near Field Communication). Гості можуть скористатися своїм смартфоном з підтримкою NFC для отримання доступу до меню, просто доторкнувшись до відповідного пристрою.

Деякі ресторани встановлюють планшети на столиках, де гості можуть переглядати цифрове меню, вибирати страви та надсилати замовлення безпосередньо зі свого місця. Також ресторани використовують інтерактивні екрани або проєкції для відображення цифрового меню на стінах або інших поверхнях у закладі. Гості можуть взаємодіяти з цими екранами, вибирати страви та здійснювати замовлення. Вищезазначене – це скоріше способи представлення, оскільки планшет та інша техніка лише відтворюють певну інформацію.

Отже, самих форм реалізації є три: у вигляді мобільного застосунку, у вигляді вебзастосунка та картинки. Надалі ми розглянемо лише перші два методи.

1.2.1 Реалізація у вигляді мобільного застосунку

Ресторани можуть створювати власні мобільні застосунки, які надають гостям доступ до цифрового меню. Вони можуть бути завантажені з магазинів (таких як App Store або Google Play) і дозволяють гостям переглядати, робити замовлення, залишати відгуки та здійснювати оплату.

Перевагами впровадження мобільного застосунку для бізнесу є зниження навантаження на персонал, автоматизація процесів замовлення та доставлення, персоналізація взаємодії з клієнтом, бонусні програми, оптимізація робочих процесів, збільшення середнього чека, розширення цільової аудиторії, аналіз робочих процесів та підвищення іміджу компанії.

Проте така реалізація передбачає вагомий інвестиційний вклад, оскільки потрібно реалізовувати два застосунки задля того, щоб не втратити частину клієнтів. Мова про те, що застосунок під систему Android не підходить для користувачів IOS. Також варто зважати на потребу в постійній підтримці та регулярному оновленні, оскільки операційні системи регулярно оновлюються, а платформи App Store та Google Play вимагають постійну актуалізацію під найновіші версії операційних систем задля забезпечення безпеки. Наразі є варіант використовувати умовний Flutter для створення універсального застосунку, це допоможе трохи скоротити витрати. Але застосунок скоріше за все доведеться додатково рекламувати.

Варто зауважити, що у мобільних застосунків є значна перевага в умовах поганого інтернет з’єднання, оскільки вони здатні зберігати дані в офлайн режимі. З погляду на утримання клієнтів, мобільні застосунки також мають деякі переваги. Застосунок завжди «під рукою» на телефоні, що

робить доступ до інформації швидким і зручним. Крім того, вони можуть використовувати push-сповіщення для нагадування про акції, знижки й нові пропозиції, що сприяє збереженню клієнтів і підвищує їхню участь.

1.2.2 Реалізація у вигляді вебсайту

Створення цифрового меню у вигляді мобільного сайту має низку переваг, що робить його більш привабливим для ресторанного бізнесу порівняно з мобільними застосунками.

По-перше, мобільний сайт має зручний і відомий користувачам інтерфейс, що є стандартним для більшості людей. Користувачам не потрібно завантажувати та встановлювати застосунки, що забезпечує швидший доступ до інформації про ресторан.

Друга перевага полягає в тому, що мобільні сайти є сумісними з будь-якими пристроями, незалежно від їх операційної системи. Це дозволяє залучити більше аудиторії й зробити взаємодію з рестораном більш доступною.

Окрім того, мобільні сайти мають ширші охоплення порівняно з мобільними застосунками. Для користувачів достатньо перейти за посиланням, щоб переглянути меню ресторану, що спрощує початок взаємодії з бізнесом і не потребує додаткових дій.

Утримання клієнтів також стає більш ефективним за допомогою мобільних сайтів. Користувачі завжди можуть швидко звертатися до сайту через браузер на своєму пристрої, а також отримувати push-сповіщення з акціями і пропозиціями, що сприяє підтримці інтересу і залучає до бізнесу.

Остання перевага мобільних сайтів - це зменшення витрат на виходження на ринок. Порівняно з розробкою та підтримкою мобільних застосунків, створення мобільного сайту вимагає менших фінансових і часових затрат, оскільки достатньо оплатити лише доменне ім’я та хостинг.

Отже, внаслідок зручного інтерфейсу, ширшого охоплення аудиторії, збереження інтересу клієнтів і зниження витрат, мобільні сайти стають більш привабливим і ефективним інструментом для ресторанних закладів порівняно з мобільними застосунками.

1.3 Порівняльний аналіз зручності вебзастосунків для оформлення та оплати замовлення в ресторані

В даному розділі складено порівняльну характеристику у вигляді таблиці де представлено для огляду три сайти створені за допомогою агрегаторів від OddMenu, Expirenza та ChoiceQR.

OddMenu пропонує нам переглянути демонстраційний варіант, що дуже зручно, оскільки одразу можна продивитися весь функціонал зі сторони клієнта. Це значно пришвидшує процес пошуку сервісу, який допоможе реалізувати ідею, бо не потрібно шукати сайти ресторанів, що вже використовують подібне рішення від певної компанії. Проте варто зазначити, що інтерфейс демо версії досить кострубато зроблений, та навіть в такому вигляді інтуїтивно зрозумілий, тому користувачі можуть легко зрозуміти, як його використовувати, головну сторінку застосунку наведено на рисунку 1.1.

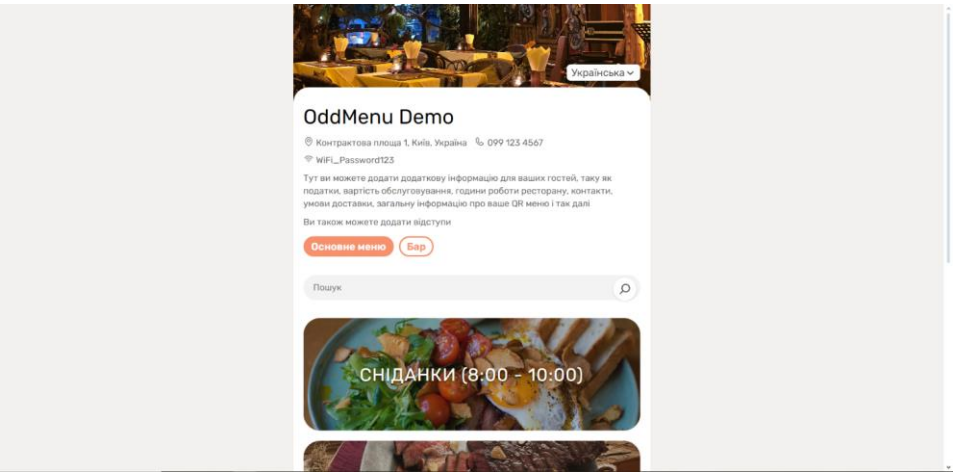

Рисунок 1.1 – Скріншот вигляду UI демо версії вебзастосунку від сервісу OddMenu [4]

Expirenza є одним з лідерів ринку, тож візуальна частина їх застосунків на рівень вища за те, що ми бачимо на рисунку 1.1. Сайт лаконічний та інтуїтивно зрозумілий, вся детальна інформація про заклад прихована та не відвертає увагу від меню, є можливість пошуку страви за назвою, але відсутній пошук за інгредієнтом, проте головною особливістю є наявність самостійної оплати.

Також можна відмітити увагу до деталей, для кожної страви є список певних алергенів, що містяться в страві (рисунок 1.2), за потреби можна додати страву до «закладок», щоб швидше перелічити список офіціанту або замовити самостійно, якщо заклад увімкнув дану функцію. З мінусів зауважу, що переглядаючи список обраних нами позицій не прораховується загальна вартість.

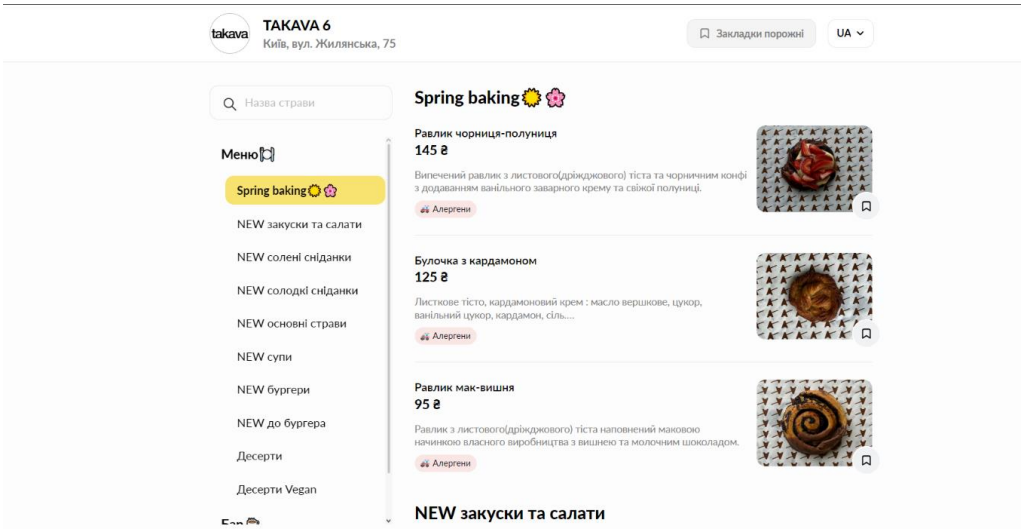

Рисунок 1.2 – Скріншот вигляду UI сайту для TAKAVA від Expirenza [7]

ChoiceQR одні з перших, хто почав впроваджувати електронні меню. Тому наразі їх роботи можна зустріти чи не найчастіше в ресторанах, барах та кафе. Дизайн інтуїтивно зрозумілий, проте навантажений, деякі елементи повторюються декілька разів, що не має в собі сенсу, але візуально завантажує простір (рисунок 1.3).

З ключових особливостей є наявність «Популярне», де можна переглянути ходові позиції, які найчастіше замовляють, це може допомогти гостю з вибором. Є можливість поділитися стравою, тобто надіслати посилання на певну позицію, що доволі зручно. Алергени ніяк не виділені, проте прописаний повний склад. Також присутня можливість додавання в «обране», проте так само як і в Expirenza немає можливості переглянути суму обраного.

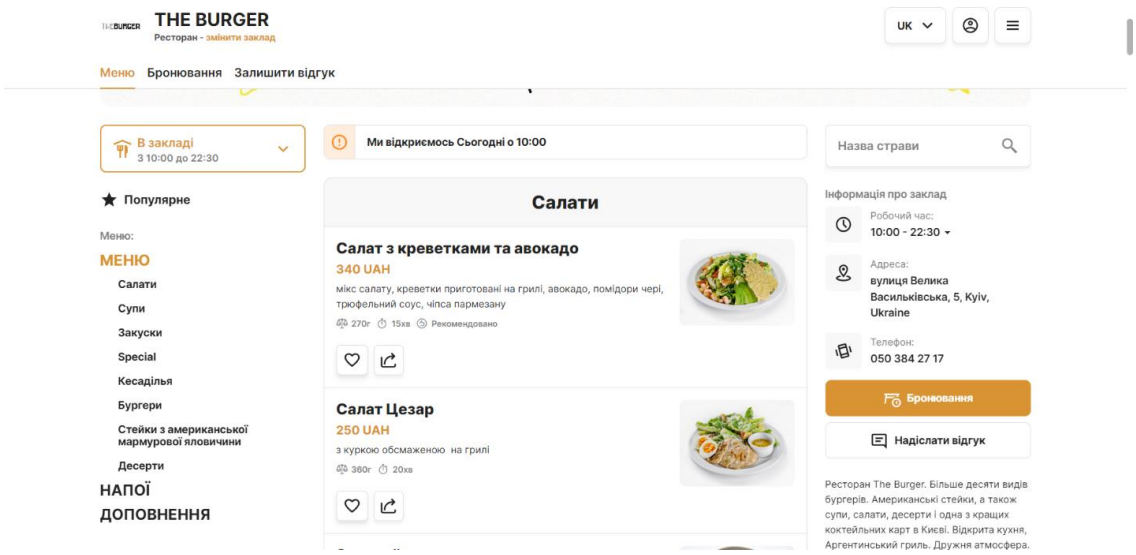

Рисунок 1.3 – Скріншот вигляду UI сайту для The Burger від ChoiceQR [8]

Для зручності оформимо дані у вигляді таблиці 1.1.

Таблиця 1.1 – Порівняння обраних застосунків по критеріям

Критерій	**OddMenu**	**Expirenza**	**ChoiceQR**
1	**2**	**3**	**4**
Адаптивність	Так	Так	Так
Інтерфейс	Простий та порожній	Простий та зрозумілий	Простий та перенавантажений
Інтеграція з іншими системами	Ні	Так	Так

Продовження таблиці 1.1

1	2	3	4
Тариф	10$ за місяць	Немає фіксованої щомісячної вартості, підключення безкоштовне, плата 1,3% від суми рахунку	Тариф «Standard» обійдеться в 1200 грн на місяць
Наявність системи відгуків	Так	Так	Так
Наявність реєстрації	Ні	Ні	Так

1.4 Постановка задачі

Тож у контексті підвищеного попиту на цифрові технології у сфері обслуговування, створення такого вебзастосунка є актуальним для підвищення ефективності роботи закладів, забезпечення кращого досвіду для клієнтів та зниження оперативних витрат.

Метою роботи є створення вебзастосунка для оптимізації та автоматизації процесу оформлення замовлення та оплати в закладах харчування.

Об’єктом роботи є система оформлення та оплати замовлень для закладів харчування.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз наявних застосунків для оформлення та оплати замовлень для закладів харчування;

– розробити та реалізувати зручний інтерфейс користувача для інтуїтивно зрозумілого використання сайту;

– розробити архітектуру програми, створити логіку для обробки та управління замовленнями, включаючи створення, відстеження та аналіз замовлення;

– створити та інтегрувати базу даних для збереження інформації про продукцію ресторану;

– розробити застосунок для адміністрації бізнесу та робітників закладу, щоб надати можливість відстеження.

2 АНАЛІЗ ТА ОБГРУНТУВАННЯ ОБРАНИХ ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ ЗАСТОСУНКУ ДЛЯ ОФОРМЛЕННЯ ТА ОПЛАТИ ЗАМОВЛЕННЯ В РЕСТОРАНІ

2.1 Аналіз використаних інструментів для реалізації застосунку

Інтерфейс користувача виступає основним засобом, за допомогою якого відбувається взаємодія між людиною і цифровими системами, сприяючи ефективному виконанню різних завдань. Ця взаємодія передбачає не лише фізичну участь користувачів, але їхнє сприйняття та концептуальне залучення, що підкреслює роль інтерфейсу користувача. До того ж користувацький досвід охоплює емоційні та психологічні реакції користувачів до, під час і після використання програми, як визначено в ISO 9241-210.

ISO 9241-210 – міжнародний стандарт, що визначає принципи ергономіки людино-системної взаємодії для створення вебсайтів та програмного забезпечення, які зосереджені на забезпеченні користувачам позитивного досвіду взаємодії. Він акцентує на розробці систем, орієнтованих на користувача, включаючи усі аспекти досвіду користувача – емоції, вподобання, сприйняття та поведінку, до, під час та після використання системи [9].

А одним з фундаментальних аспектів зручного інтерфейсу є швидкість реакції сайту на дію користувача для цього виконують відокремлення логіку роботи та збереження від самого інтерфейсу, а також створюють багатомодульну систему [10].

Частину для робітників закладу було вирішено розроблювати під ОС Android. Java та Kotlin наразі є конкурентами у світі розробки під дану систему, оскільки першою мовою написано безліч старих застосунків, а друга стрімко набирає обертів на сучасному ринку. Тому було вирішено

обрати саме Kotlin для створення частини, що призначена для робітників закладу ресторанного господарства. Це статично типізована мова програмування, яка працює на віртуальній машині Java. Мова була офіційно представлена у 2011 році та згодом у 2017 році стала однією з офіційних мов для розробки Android за підтримки Google. Kotlin став популярним вибором для бізнесу що потребують вже мають проєкти під Android завдяки простоті використання, безпеці та сумісності з Java.

Kotlin має сучасний синтаксис, який спрощує код і робить його більш читабельним. Він включає такі корисні функції, як виведення типів, лямбда-вирази та функції розширення, які дозволяють писати більш виразний і лаконічний код, ніж Java.

Однією з головних сильних сторін Kotlin є система роботи з нульовими значеннями. Мова використовує строгу систему типів для уникнення помилок нуля (NullPointerExceptions) на етапі компіляції, що значно знижує ризик виникнення помилок під час виконання програми.

Kotlin розроблено для повної сумісності з Java. Це означає, що ви можете використовувати всі наявні бібліотеки та фреймворки Java у своїх проєктах на Kotlin, і навпаки, код на Kotlin можна легко викликати з Java. Така інтероперабельність робить перехід на Kotlin менш болючим для наявних проєктів.

Kotlin вводить корутини як сучасний підхід до асинхронного програмування, що дозволяє писати асинхронний код так само легко, як і синхронний. Корутини в Kotlin допомагають уникнути «пекла зворотного виклику» і забезпечують більш зручний спосіб управління асинхронними процесами, такими як мережеві запити, обробка файлів або взаємодія з базами даних.

Завдяки офіційній підтримці Google та широкому прийняттю спільнотою розробників, Kotlin постійно розвивається та вдосконалюється. Розробники мають доступ до безлічі ресурсів для навчання та підтримки,

включаючи документацію, навчальні посібники, курси та активні спільноти в таких місцях, як StackOverflow, GitHub та Reddit.

Існує два варіанти створення користувацьких інтерфейсів в Android-застосунках за допомоги XML чи Jetpack Compose [11]. XML – мова розмітки, що використовується для структурування даних у форматі, який читабельний як для людей, так і для машин. XML дозволяє користувачам визначати свої власні теги та структуру документа, що робить її зручною для обміну складними даними між різними системами та платформами. XML широко використовується у розробці вебслужб, конфігураційних файлів, а також для представлення даних у багатьох програмних застосунках на базі Java та Kotlin.

Jetpack Compose своєю чергою це сучасний фреймворк для створення користувацьких інтерфейсів на Android, який використовує декларативний підхід до програмування UI. Він дозволяє розробникам більш ефективно і інтуїтивно створювати динамічні інтерфейси, використовуючи Kotlin як основну мову програмування. Compose спрощує реалізацію реактивних інтерфейсів, інтегруючи вбудоване управління станом, що автоматично оновлює UI при зміні даних. Використання Compose може значно зменшити кількість коду та підвищити швидкість розробки завдяки перевикористанню компонентів та модульності. Також, Compose підтримує тісну інтеграцію з іншими частинами Android Jetpack, полегшуючи розробку сучасних і ефективних застосунків [12].

Також у роботі було використано Orbit MVI – це бібліотека для Kotlin, яка допомагає реалізувати архітектурний патерн MVI в Android застосунках, спираючись на реактивні шаблони. Orbit спрощує управління станом та побічними ефектами, забезпечуючи чітку та консистентну організацію коду, що значно зменшує ймовірність помилок. Бібліотека використовує Kotlin корутини для спрощення асинхронного коду, роблячи його більш зрозумілим і доступним для розробників. MVI (Model-View-Intent) – це архітектурний патерн у програмуванні, який наголошує на однонапрямному потоці даних та

ясному розділенні відповідальності між компонентами системи, роботу схематично зображено на рисунку 2.1.

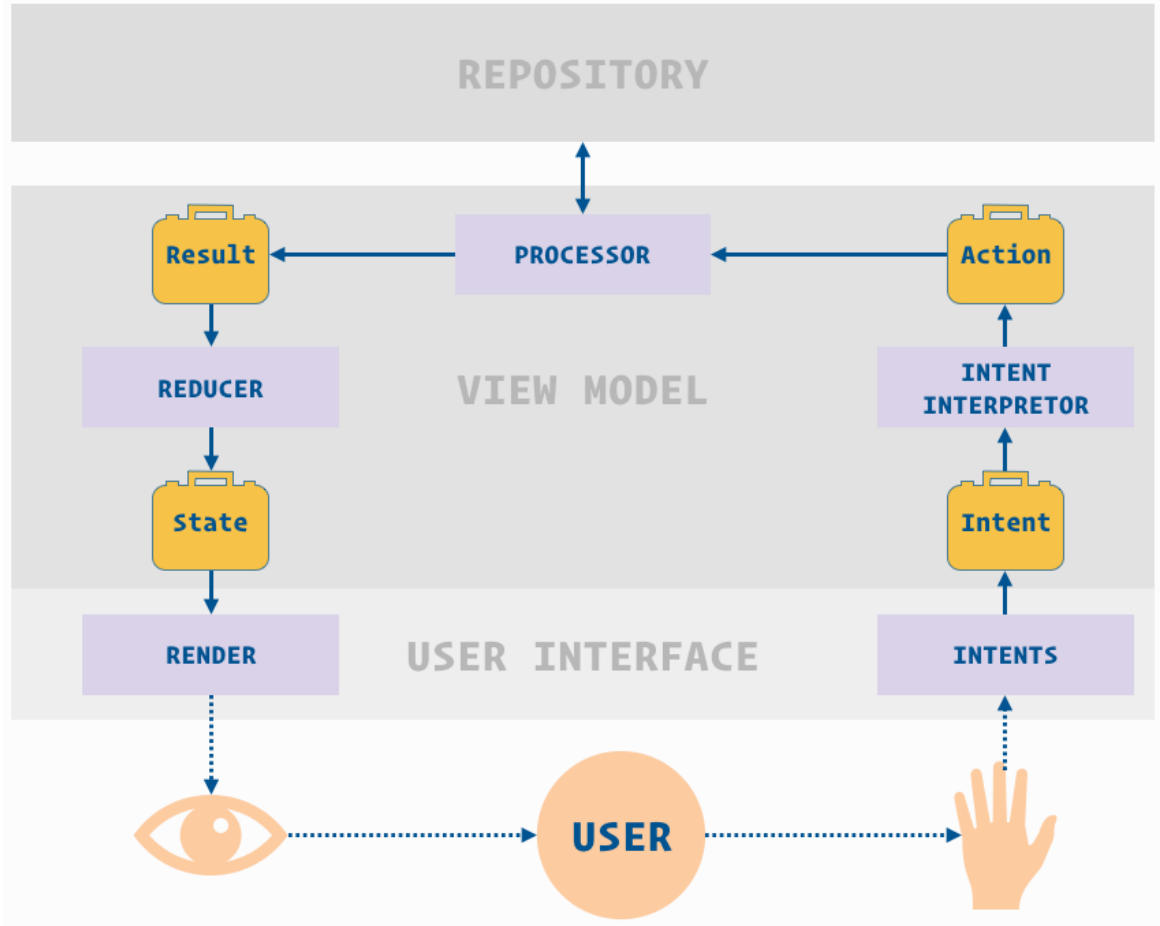

Рисунок 2.1 – Схематичне пояснення принципу роботи патерну MVI [13]

Цей патерн ділить архітектуру застосунку на три основні частини: Model (Модель), View (Представлення) та Intent (Намір). Model є незмінним станом застосунку, який оновлюється через чисті функції відповідно до вхідних дій (Intents), що їх визначає користувач. View відображає стан моделі, але не взаємодіє напряму з даними або логікою бізнесу, лише відправляє Intents на основі дій користувача. Intent в цьому контексті визначається як дія чи подія, що ініціює зміну стану, яка передається в Model.

Також у роботі застосовуються Jetpack Hilt та Compose Destinations. Hilt – це бібліотека для залежностей від Google, яка спрощує інтеграцію залежностей у проєктах Android, використовуючи Dagger 2. Hilt автоматизує

багато аспектів управління залежностями, включно з Android життєвими циклами, зменшуючи кількість шаблонного коду та збільшуючи читабельність і легкість обслуговування коду. Jetpack Hilt призначений для спрощення використання залежностей Dagger в Android-застосунках, роблячи його більш доступним і легким для інтеграції [14]. Hilt забезпечує стандартизований спосіб інтеграції Dagger dependency injection в Android-застосунки. Ця стандартизація дозволяє уникнути поширених помилок і спрощує процес конфігурації. Автоматично генерує компоненти Dagger та адаптує їх до життєвих циклів класів Android, зменшуючи потребу розробників вручну створювати та керувати життєвими циклами компонентів. Hilt розроблений для безперебійної роботи з фреймворком Android, надаючи заздалегідь визначені компоненти та розширення, які відповідають архітектурі Android, такі як дії, фрагменти та сервіси. Стандартизуючи конфігурацію та надаючи вбудовану підтримку для тестування, Hilt дозволяє легко замінювати модулі під час тестування, що сприяє більш надійному та простому тестуванню [15].

Compose Destinations - це бібліотека маршрутизації для Jetpack Compose, яка спрощує навігацію в програмах, написаних за допомогою Compose. Вона полегшує розробникам управління потоками навігації за допомогою анотацій і автоматично згенерованого коду, знижуючи ризик помилок і спрощуючи масштабування проєкту. Compose Destinations використовує безпеку типів Kotlin, щоб зменшити кількість помилок під час виконання та забезпечити перевірку шляхів навігації під час компіляції. Бібліотека використовує анотації для визначення маршрутів навігації, що робить навігаційний граф простішим для розуміння та керування, ніж традиційні підходи, які використовують XML або декларативний код. Compose Destinations від самого початку підтримує глибокі зв’язки, що полегшує керування складними навігаційними сценаріями, які включають посилання на застосунки [16].

Jetpack Hilt та Compose Destinations - це значний прогрес у розробці Android, особливо з погляду зменшення наскрізного коду та спрощення складних аспектів розробки застосунків. Hilt справляється з тонкощами ін’єкції залежностей, в той час, як Compose Destinations займається навігацією, що робить їх важливими інструментами для сучасної практики розробки Android. Ці бібліотеки не лише підвищують продуктивність розробників, але й покращують загальну якість та зручність обслуговування застосункі.

Клієнтську сторону було реалізовано за допомоги трьох інструментів: HTML, CSS, JavaScript. Кожен з цих інструментів відіграє свою ключову роль у створенні структури, дизайну та функціональності вебсайту.

HTML становить основу будь-якого вебсайту. Це не програмна мова, а мова розмітки, яка використовується для структурування вмісту на вебсторінках. За допомогою HTML створюються основні елементи сторінки, такі як заголовки, параграфи, списки, посилання, зображення та форми. Теги та атрибути HTML визначають, як елементи повинні бути організовані та якими даними вони володіють [17].

CSS використовується для визначення стилю та візуального оформлення вебсайту. Це мова стилів, яка дозволяє розробникам задавати кольори, шрифти, розміри, відступи, рамки та багато інших аспектів візуального представлення HTML-елементів. CSS також забезпечує адаптивний дизайн, дозволяючи сайтам правильно відображатися на різних пристроях та розмірах екранів. Використовуючи класи, ідентифікатори та селектори, CSS здійснює «каскадне» застосування стилів, що робить його дуже потужним інструментом для дизайнерів [18].

JavaScript – це мова програмування, яка використовується для додавання інтерактивності на вебсайтах. З JavaScript можна створювати динамічні оновлення контенту, інтерактивні карти, анімовані графіки, слайд-шоу, форми з валідацією даних та багато іншого. JavaScript може взаємодіяти з HTML та CSS для маніпуляції елементами сторінки, змінюючи їх

відповідно до дій користувача, таких як кліки мишкою, наведення курсора, натискання клавіш тощо. Завдяки таким бібліотекам, як jQuery, React, Angular та іншим, можливості JavaScript стають ще ширшими [19].

JavaScript універсально сумісна з усіма сучасними веббраузерами без необхідності встановлення додаткових плагінів, що робить її однією з найдоступніших і найпоширеніших мов програмування у світі. Вона безперешкодно працює на різних платформах і пристроях, забезпечуючи узгоджену поведінку та представлення даних.

З появою Node.js JavaScript розширив свою сферу застосування з програмування на стороні клієнта (браузерів) на програмування на стороні сервера. Це дозволяє розробникам використовувати JavaScript для створення повно стекових засотсунків, що означає, що одна і та ж мова може бути використана як для front-end, так і для back-end розробки, спрощуючи процес розробки й зменшуючи необхідність перемикання між мовами.

Екосистема JavaScript містить в собі велику кількість бібліотек та фреймворків, кожна з яких призначена для спрощення різних аспектів веброзробки. Наприклад:

– React використовується для створення користувацьких інтерфейсів з акцентом на декларативному програмуванні та компонентній архітектурі.

– Angular надає повний фреймворк для розробки динамічних односторінкових застосунків з потужними функціями зв’язування даних та модуляризації.

– Vue.js захоплює своєю простотою та гнучкістю, сприяючи швидкій розробці прогресивних користувацьких інтерфейсів.

– jQuery спрощує навігацію HTML-документами, обробку подій, анімацію та Ajax-взаємодію для швидкої веброзробки.

JavaScript підтримує асинхронне програмування за допомогою зворотних викликів, обіцянок та синтаксису async/await. Це особливо корисно в таких операціях, як отримання даних з сервера, де ви не хочете

блокувати основний потік. Асинхронний JavaScript, наприклад, AJAX, дозволяє оновлювати вебсторінки асинхронно, обмінюючись даними з вебсервером у фоновому режимі.

Для ефективної роботи також застосовуться Live Server. Це дуже корисний інструмент для розробників вебсайтів, який спрощує процес розробки за допомогою автоматичного перезавантаження сторінок при внесенні змін у код. Це розширення, яке часто використовується з редакторами коду, такими як Visual Studio Code. Live Server може використовуватися для розробки як статичних HTML-сторінок, так і динамічних вебзастосунків, що використовують серверні скрипти або технології типу AJAX. Зазвичай Live Server легко інтегрується в більшість сучасних розробницьких середовищ і не вимагає складної настройки. У Visual Studio Code, наприклад, достатньо лише встановити розширення і запустити Live Server через контекстне меню або командний рядок. Також плагін запускає локальний розвитковий сервер, що дозволяє тестувати вебзастосунки в умовах, близьких до продакшну, але без необхідності розгортання на віддалені сервери.

Для зберігання даних було розглянуто два типи баз даних: реляційні та нереляційні, схематично зображені на рисунку 2.2. Реляційні бази даних (СКБД) є дуже потужним інструментом для управління даними, особливо в середовищах, де необхідна сувора узгодженість, надійність і структурованість запитів до даних. Кожна таблиця в реляційній базі даних має ключ (зазвичай первинний ключ), який однозначно ідентифікує кожен рядок. Реляційні бази даних підтримують мову SQL (Structured Query Language - мова структурованих запитів) для маніпулювання даними. Однією з ключових переваг реляційних баз даних є підтримка транзакцій, що дозволяє виконувати групу операцій як єдину одиницю роботи, яка або повністю завершується, або не виконується взагалі. Це забезпечує атомарність, узгодженість, ізольованість і довговічність (властивості ACID), які є критично важливими для фінансових систем, систем управління

ресурсами підприємства та інших застосунків, де надійність даних є життєво важливою. Реляційні бази даних вимагають попередньо визначеної схеми бази даних перед зберіганням даних. Схема визначає структуру даних, включаючи таблиці, поля, типи даних для кожного поля і зв'язки між таблицями. Це забезпечує суворий порядок і дозволяє застосовувати типові обмеження безпеки та цілісності, мінімізуючи помилки в даних і підвищуючи загальну якість даних.

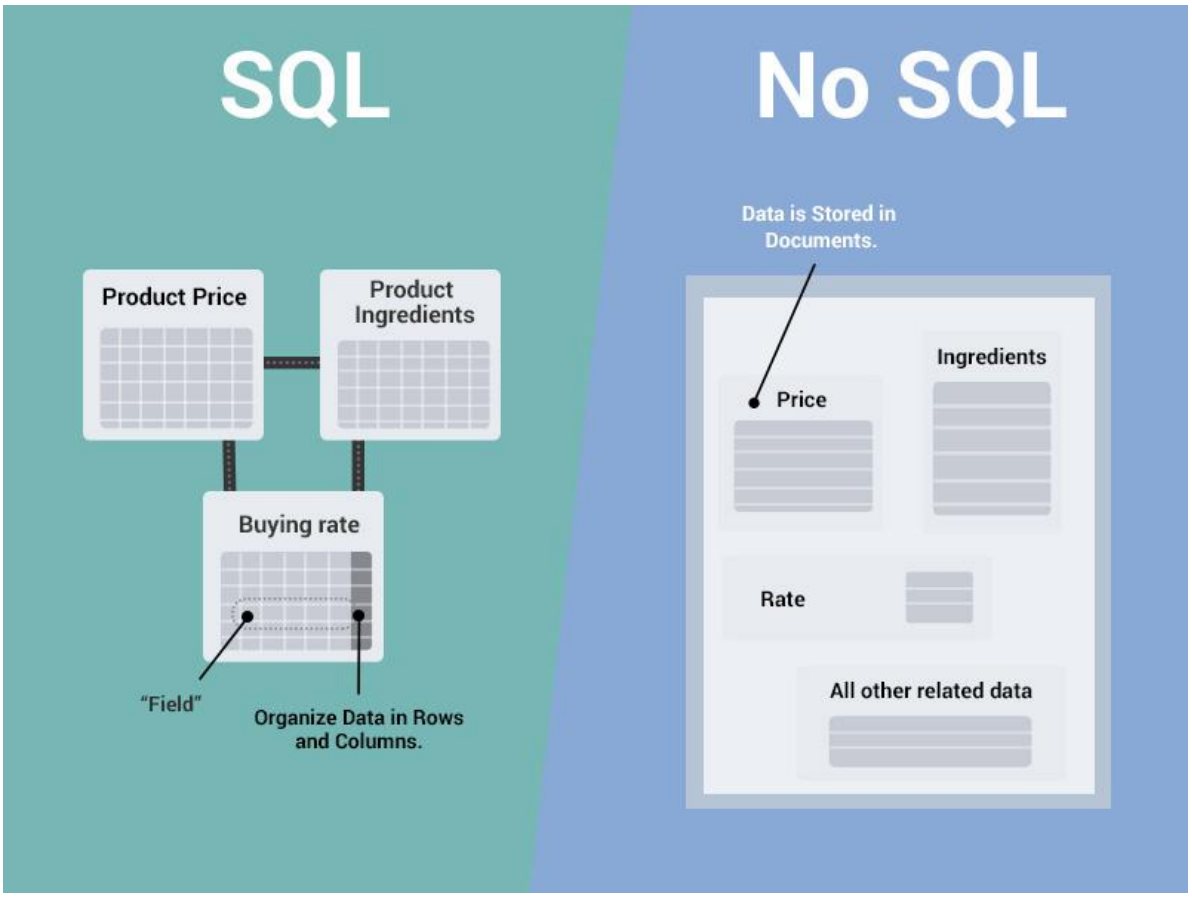

Рисунок 2.2 – Візуальне зображення різниці між двома видами баз даних [20]

Хоча реляційні бази даних є високоорганізованими та узгодженими, вони стикаються з проблемами при горизонтальному масштабуванні, що передбачає додавання нових серверів для обробки зростаючих навантажень. Однак сучасні досягнення в технології СУБД призвели до таких удосконалень, як розподілені архітектури баз даних і хмарні рішення, які

підвищують їхню масштабованість і продуктивність без шкоди для властивих їм властивостей ACID.

Завдяки підтримці SQL реляційні бази даних чудово справляються зі складними запитами. SQL дозволяє створювати складні запити і маніпулювати даними, включаючи об’єднання декількох таблиць, виконання підзапитів і агрегування даних, що робить реляційні бази даних особливо придатними для застосунків, які вимагають складного аналізу даних і звітності [21].

Реляційні бази даних суворо забезпечують цілісність даних за допомогою таких обмежень, як зовнішні ключі, обмеження на перевірку та унікальні обмеження. Ці механізми гарантують, що дані відповідають визначеним правилам, запобігаючи аномаліям і підтримуючи точність та узгодженість даних протягом їхнього життєвого циклу.

Отже, реляційні бази даних незамінні в сценаріях, які вимагають ретельного управління даними, складних запитів і непохитної цілісності даних. Їхній структурований підхід і надійна підтримка транзакцій роблять їх ідеальними для критично важливих бізнес-застосунків у різних галузях, включаючи фінанси, охорону здоров’я та уряд. З розвитком технологій реляційні бази даних продовжують адаптуватися, пропонуючи більш масштабовані рішення і зберігаючи свою актуальність в умовах мінливих потреб у зберіганні даних.

Своєю чергою нереляційні бази даних (NoSQL) не використовують табличну модель, а замість цього використовують більш гнучку, часто документоорієнтовану структуру. Вони добре підходять для роботи з великими масивами розподілених даних і часто використовуються для великих вебзастосунків. Оскільки NoSQL не потребують фіксованої схеми, що дозволяє розробникам легко змінювати структуру даних без закриття бази даних або зміни всього коду програми. Ця гнучкість ідеально підходить для сучасних застосунків, які можуть швидко розвиватися відповідно до вимог користувачів або ринку.

Бази даних NoSQL ідеально підходять для горизонтального масштабування, що означає можливість керувати зростаючими обсягами даних і запитів, додаючи більше серверів до пулу ресурсів. Це контрастує з вертикальним масштабуванням, яке частіше використовується в реляційних базах даних, і яке часто є дорожчими і складним у реалізації.

Багато баз даних NoSQL підтримують реплікацію даних і автоматичне відновлення, забезпечуючи високу доступність і стабільність сервісу навіть у разі апаратних або мережевих збоїв. Це особливо важливо для застосунків, які повинні бути доступні цілодобово.

Бази даних NoSQL часто є більш економічно вигідними для дуже великих наборів даних і розподілених систем, оскільки вони оптимізовані для використання на недорогому готовому обладнанні або в хмарних сервісах, що знижує витрати на зберігання великих обсягів даних.

Було обрано базу даних Firebase. Ця платформа розроблена Google для створення мобільних та вебзастосунків, надає широкий спектр інструментів та сервісів, які допомагають розробникам в управлінні, розвитку та масштабуванні їхніх програм. Для розробки Firebase пропонує два різновиди хмарних сховищ, а саме Firebase Realtime Database та Cloud Firestore. Обидва ці сервіси надають можливість створення та управління базами даних в режимі реального часу, забезпечують масштабованість та легкість використання, але вони мають ряд ключових відмінностей, що роблять кожен з них принагідним для різних сценаріїв використання.

Firebase Realtime Database – це база даних NoSQL, яка зберігає дані у форматі JSON (рис. 2.3) та дозволяє синхронізувати інформацію між користувачами в реальному часі з дуже низькою затримкою. Це особливо корисно для застосунків, що потребують швидкого обміну даними між користувачами, таких як чати або колаборативні платформи. Коли дані змінюються, вони миттєво синхронізуються з усіма підключеними клієнтами. Є підтримка мільйонів одночасних з’єднань, хоча це може стати дороговартісним у великих масштабах. Підтримує простий доступ через

REST API та SDK для основних платформ, включаючи Android, iOS та JavaScript.

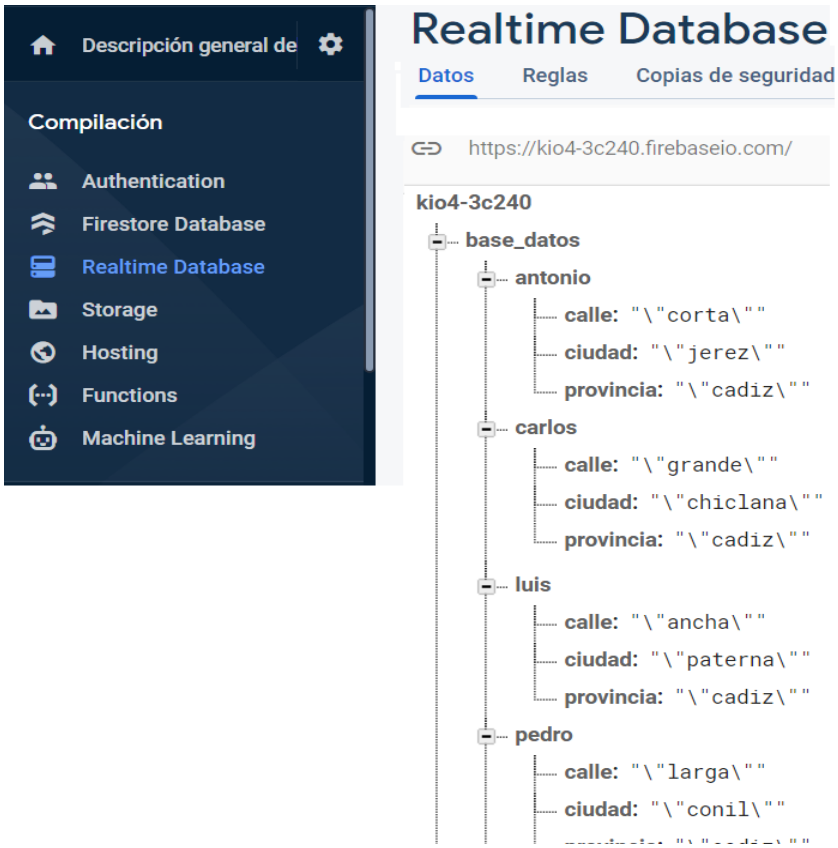

Рисунок 2.3 – Приклад структурованості даних в Realtime Database [22]

Для цього проєкту було обрано Cloud Firestore (рис.2.4), оскільки це більш сучасна база даних, яка забезпечує синхронізацію даних в реальному часі, але пропонує додаткові можливості та покращену масштабованість порівняно з Realtime Database.

Дані організовані у колекції та документи, що полегшує структурування складних даних і розширює можливості запитів. Firestore оптимізований для великих масштабів з автоматичним шардінгом даних і ефективнішим управлінням ресурсами. Підтримує транзакції на кількох документах та колекціях, а також складніше згруповані запити. Firestore надає широку підтримку офлайн-режиму, дозволяючи користувачам продовжувати працювати з застосунками навіть без мережевого з’єднання.

Як вже зазначалося вище, Firebase має багато інструментів та сервісів, що поліпшують роботу з базами даних, тому в даній роботі було використано Firebase Authentication. Сервіс від Firebase, який надає повний набір рішень для аутентифікації користувачів, що дозволяє легко інтегрувати безпечний і гнучкий метод входу в мобільні та вебзастосунки.

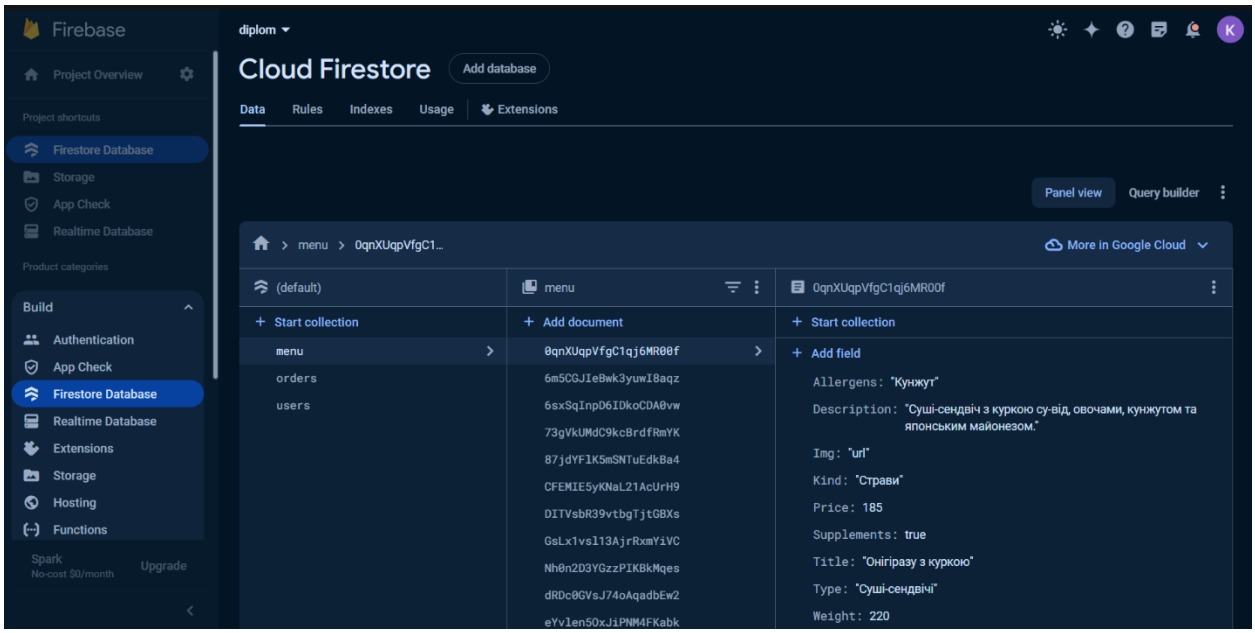

Рисунок 2.4 – Структуризація даних в Cloud Firestore

Firebase Authentication підтримує багато популярних постачальників ідентифікаційних даних, таких як Google, Facebook, Twitter, Apple та багато інших. Також підтримується автентифікація за допомогою електронної пошти та пароля, автентифікація за номером телефону (включаючи одноразові SMS-паролі) та анонімна автентифікація. Автентифікація легко інтегрується з іншими службами Firebase, такими як Firestore, Realtime Database і Cloud Storage, дозволяючи використовувати облікові дані користувача для налаштування політик безпеки доступу до даних. Firebase Authentication керує сеансами користувачів, дозволяючи розробникам легко відстежувати активність сеансів і виконувати такі операції, як вихід з усіх сеансів або перевірка токенів доступу. Сервіс захищає ідентифікаційні дані користувачів за допомогою сучасних механізмів шифрування та безпечного

зберігання [23]. Також підтримується відповідність регуляторним стандартам, таким як GDPR. Firebase Authentication підтримує багатофакторну автентифікацію, що додає додатковий рівень безпеки з другим етапом перевірки після введення пароля.

2.2 Підхід до реалізації back-end проєкту

Існує багато підходів до розробки програмних застосунків, якими керуються програмісти та інженери при створенні програмного забезпечення. Ці парадигми впливають на стилі кодування, дизайн архітектури програмного забезпечення та методології тестування [24].

Монолітна архітектура визначає структуру програмного застосунку як єдиний, неподільний блок, де всі компоненти (інтерфейс користувача, бізнес-логіка, база даних тощо) тісно інтегровані в один виконуваний файл. Такий підхід має кілька важливих переваг:

– Простота розробки, бо усі частини застосунку знаходяться в одному місці, що спрощує розробку та тестування.

– Єдине розгортання, оскільки всі компоненти розгортаються разом, що зменшує складність управління версіями та конфігурацією.

– Однак, монолітна архітектура має і свої недоліки:

– Складність масштабування, через те що важко масштабувати окремі частини застосунку, оскільки вони всі тісно пов'язані між собою.

– Ускладнене оновлення однієї частини застосунку може вимагати повторного розгортання всього застосунку, що може призвести до простоїв.

Архітектура мікросервісів передбачає поділ програми на менші незалежні сервіси, кожен з яких виконує окрему функцію і взаємодіє з іншими через чітко визначені інтерфейси, зазвичай через HTTP API. Кожен

мікросервіс можна розробляти, розгортати, модифікувати та масштабувати незалежно від інших, що забезпечує більшу гнучкість та стійкість [25].

Контейнеризація – це технологія, яка дозволяє упаковувати програми разом з усіма їхніми залежностями в контейнери, забезпечуючи узгоджене виконання на будь-якому обладнанні, незалежно від локального середовища. Контейнери ізольовані один від одного і від хост-системи, що підвищує безпеку і полегшує масштабування і розгортання застосунків, пропонуючи високу ефективність і портативність.

Безсерверна архітектура – це парадигма розробки програмного забезпечення, за якої розробники можуть створювати та запускати програми без необхідності явного керування серверами. Це не означає, що сервери відсутні, натомість управління інфраструктурою перекладається на постачальників хмарних послуг, які динамічно розподіляють ресурси. Безсерверні архітектури часто використовують «Функції як послугу» (FaaS), де код виконується у відповідь на події і автоматично масштабується в залежності від попиту.

Ключові особливості безсерверних застосунків включають:

– безсерверні платформи автоматично масштабують їх, додаючи або видаляючи ресурси відповідно до поточного навантаження. Це полегшує обробку мінливих обсягів запитів без необхідності платити за невикористані ресурси;

– з безсерверною технологією ви платите лише за час виконання вашого коду та ресурси, які він використовує протягом цього часу. Це може значно знизити витрати порівняно з традиційними хмарними або локальними серверними рішеннями;

– розробники можуть швидко писати функції, тестувати їх локально або безпосередньо в хмарі та миттєво розгортати зміни.

– абстрагуючись від базової інфраструктури, безсерверна архітектура дозволяє розробникам зосередитися виключно на написанні та розгортанні

коду, не турбуючись про операційне середовище. Це призводить до скорочення циклу розробки.

– безсерверна технологія за своєю суттю є подієво-орієнтованою, обробляючи запити по мірі їх надходження без необхідності в попередньо виділених ресурсах. Це робить її ідеальною для програми зі змінним трафіком і тих, що реагують на події в реальному часі.

– хмарні провайдери забезпечують високу доступність і відмовостійкість безсерверних застосунків. Це зменшує потребу розробників в управлінні цими аспектами і забезпечує більшу надійність.

– безсерверні обчислення також можуть сприяти екологічній стійкості за рахунок оптимізації використання ресурсів та зменшення енергії, необхідної для простою серверів.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ПОСТАВЛЕНОЇ ЗАДАЧІ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Обґрунтування вибору середовища програмної реалізації

Для розробки відповідного інтерфейсу існують платформи для дизайну, в роботі було використано Figma. Це популярний інструмент для розробки інтерфейсів, який забезпечує співпрацю в реальному часі та широкий спектр можливостей для створення, прототипування та передачі дизайнів. Інтерактивні прототипи можна використовувати для проведення тестувань з реальними користувачами, збираючи зворотний зв'язок до фінальної розробки продукту.

На рисунку 3.1 зображений початковий дизайн для застосунку, яким буде користуватися персонал.

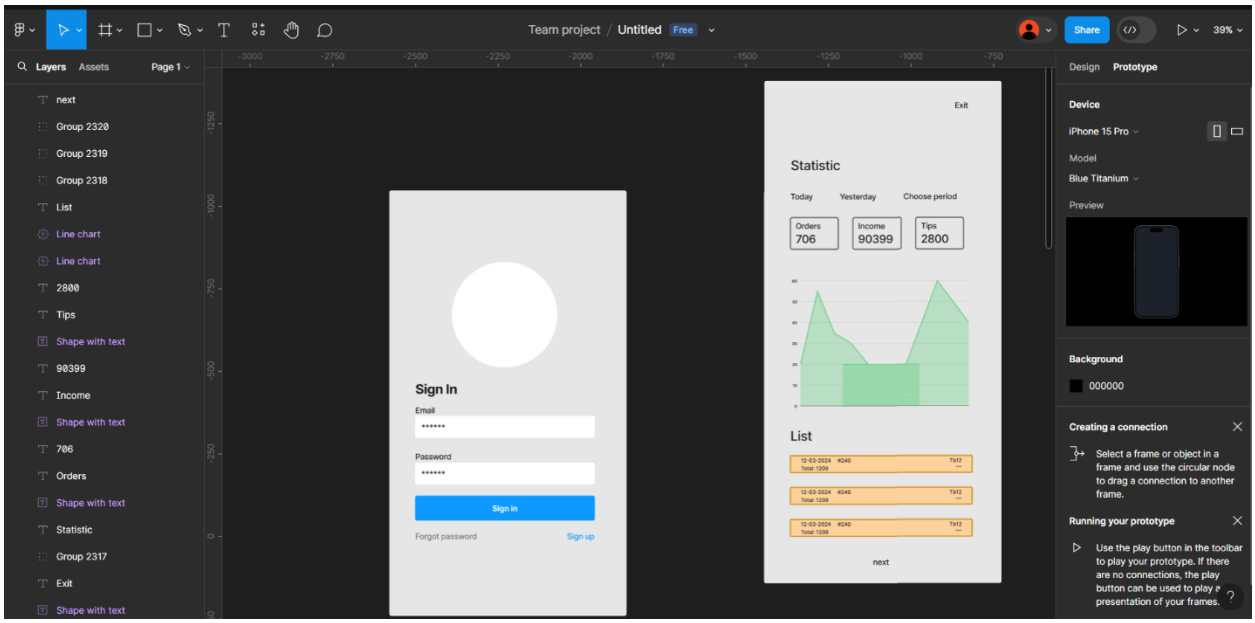

Рисунок 3.1 – Налаштування прототипу для телефону на операційній системі IOS

Figma підтримує інтеграцію з багатьма іншими інструментами та сервісами, такими як Slack, JIRA, GitHub та інші платформи для ведення проєктів та комунікацій. Крім того, Figma має відкритий API, який дозволяє розробникам створювати власні плагіни та автоматизувати рутинні задачі або додавати нові функції [26]. Для початку було вирішено реалізовувати застосунок для телефонів на базі операційної системи Android, оскільки дана ОС є найпопулярнішою у використанні [27].

Процес розробки відбувався в середовищі Android Studio, оскільки це офіційне середовище розробки для Android, засноване на платформі IntelliJ IDEA (рис. 3.2). Середовище пропонує зручний інтерфейс, який структуровано для покращення робочого процесу та продуктивності. Такі функції, як перетягування макетів, потужний редактор макетів та розширений редактор коду, допомагають спростити процес розробки.

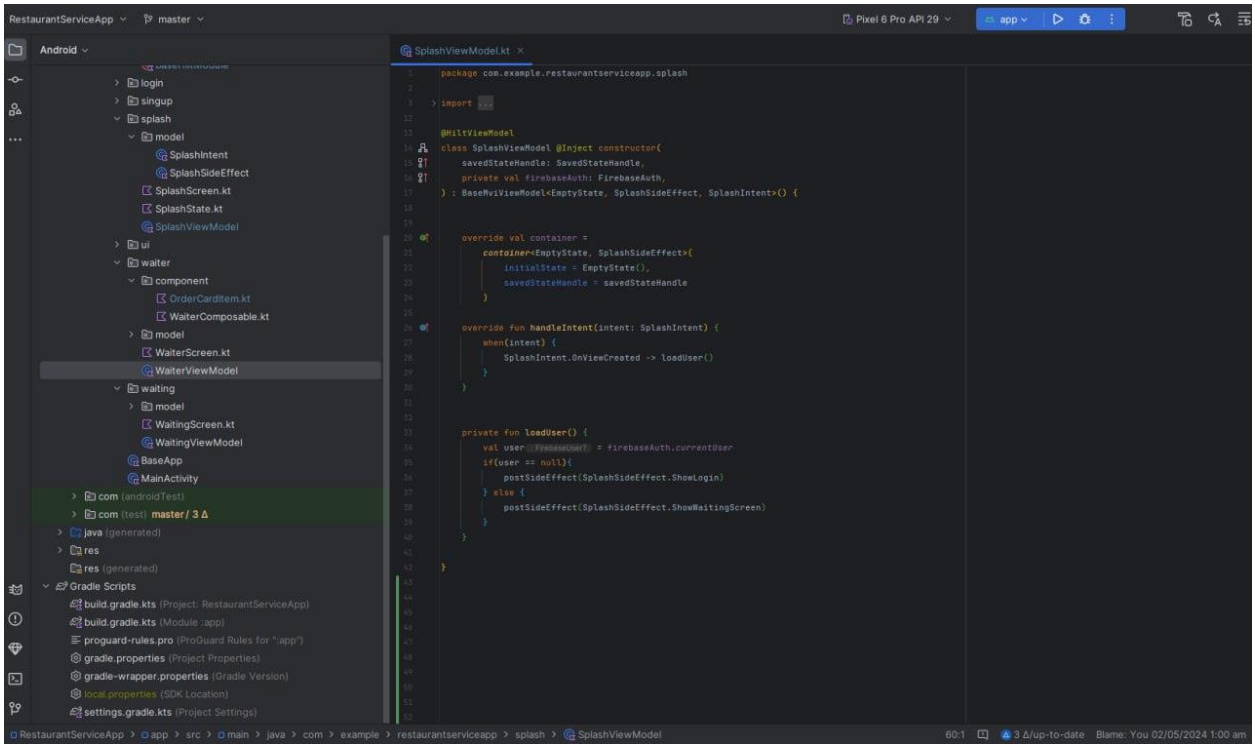

Рисунок 3.2 – Інтерфейс редактора Android Studio

Вбудований емулятор запускає застосунки Android у середовищі віртуального пристрою, імітуючи різні конфігурації пристроїв та версії

Android. Це дуже важливо для тестування програм на різних пристроях Android без використання фізичного обладнання. Використовує Gradle як систему збірки, яка автоматизує такі завдання, як збірка програми, управління залежностями та налаштування збірок для різних конфігурацій пристроїв.

До складу IDE входить інтегрований Android SDK Manager, який спрощує процес управління інструментами Android SDK, платформами та іншими компонентами, необхідними для розробки програми. Профайлер Android надає інформацію в реальному часі про використання процесора, пам’яті та мережі вашого застосунку, допомагаючи ефективно оптимізувати продуктивність програми.

Інструмент Layout Inspector дозволяє розробникам досліджувати та налагоджувати інтерфейс користувача програми, надаючи детальний огляд ієрархії макетів програми. Аналізатор APK може зменшити розмір APK-файлу, надаючи інформацію про вміст APK-файлу, допомагаючи розробникам зрозуміти вплив кожного компонента на кінцевий розмір збірки. Android Studio легко інтегрується з Google Cloud Platform, спрощуючи включення хмарних сервісів Google в застосунках, таких як служби автентифікації та бази даних.

Для розробки клієнтської частини було обрано Visual Studio Code. Це безкоштовний редактор коду з відкритим вихідним кодом, розроблений компанією Microsoft. Він універсальний, продуктивний та має широкий набір функцій, які підтримують різні мови програмування та завдання розробки. VS Code розроблений як легкий редактор, який швидко запускається та ефективно працює навіть при роботі з великими кодовими базами (рис. 3.3). Він доступний для Windows, macOS та Linux, що забезпечує однаковий досвід розробки на різних операційних системах.

Забезпечує інтелектуальне завершення коду на основі типів змінних, визначень функцій та імпортованих модулів. Ця функція підвищує швидкість і точність кодування. Також присутня безшовна інтеграція з Git’ом дозволяє

розробникам керувати контролем версій безпосередньо з редактора. Підтримується постановка, фіксація, розгалуження та перегляд відмінностей.

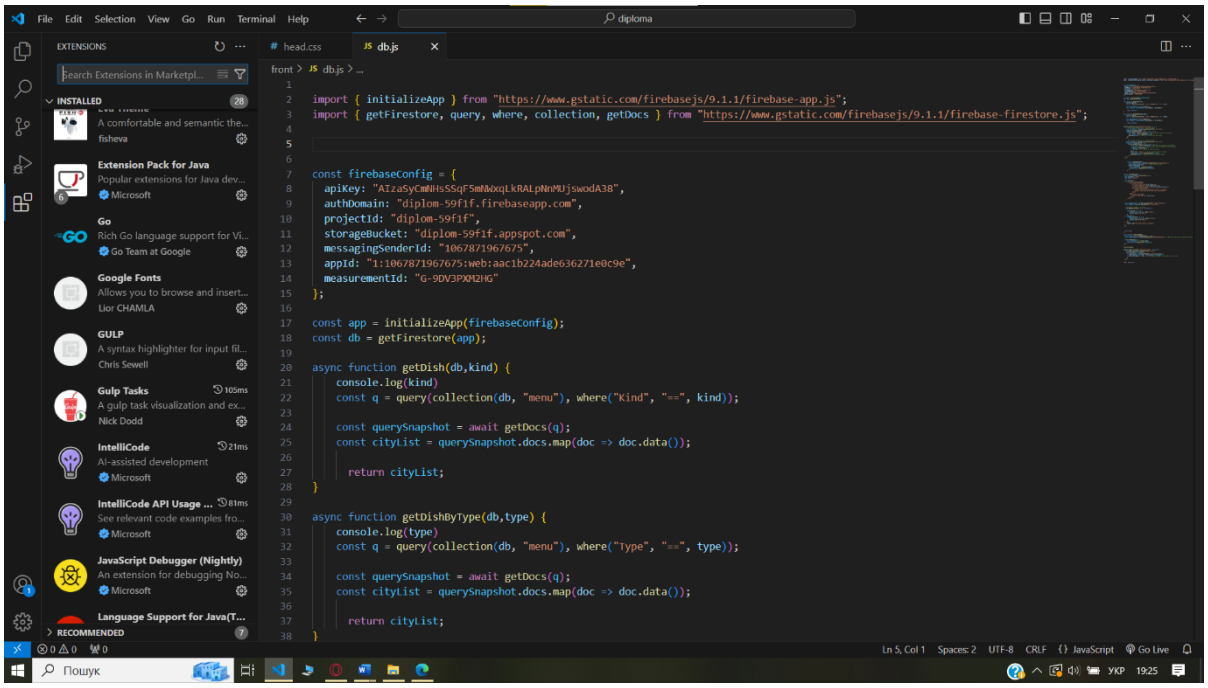

Рисунок 3.3 – приклад розширень для роботи з VS Code

VS Code має багату екосистему розширень, доступних через Visual Studio Code Marketplace. Це допомагає налаштувати своє середовище розробки за допомогою тем, відладчиків, мовних пакетів та додаткових інструментів. Інтеграція з Emmet забезпечує швидкі робочі процеси HTML і CSS, дозволяючи швидко писати код за допомогою абревіатур, які розширюються до повноцінних фрагментів коду.

Доступні фрагменти коду для різних мов і фреймворків, що полегшує швидке і послідовне створення шаблонів коду. Багатофункціональність VS Code відображається в його здатності підтримувати широкий спектр видів діяльності, мов і фреймворків. Ця універсальність робить його придатним для різних типів розробників, включаючи веб-розробників, аналітиків даних, інженерів DevOps тощо. VS Code підтримує багато мов програмування з коробки, включаючи JavaScript, TypeScript, Python, Java, C++ та інші. Додаткова підтримка мов може бути додана за допомогою розширень.

Розширення та плагіни забезпечують підтримку популярних фреймворків та бібліотек, таких як React, Angular, Vue.js, Django, Flask та багатьох інших, пропонуючи адаптовані інструменти та функції для кожного з них.

VS Code можна налаштувати під різні робочі процеси розробки. Будь то фронтенд-розробка, бекенд-розробка або повностекова розробка, VS Code надає необхідні інструменти та розширення.

Для розробки програмного забезпечення було використано JavaScript та Kotlin – кожна з мов призначена для певної сфери розробки. JavaScript в основному використовується для веброзробки, покращуючи інтерактивність та функціональність вебсторінок, тоді як Kotlin використовується для розробки застосунків, особливо для Android.

JavaScript є основою інтерактивності та динамічного контенту сайту, що в свою чергу створюється за допомоги HTML та CSS. Вона використовується як для розробки на стороні клієнта, так і на стороні сервера. JavaScript чудово підходить для створення адаптивних вебзастосунків завдяки своїй моделі, заснованій на подіях, яка дозволяє скриптам реагувати на дії користувача, такі як кліки, заповнення форм і завантаження сторінок.

Kotlin – це сучасна мова програмування зі статичною типізацією, розроблена компанією JetBrains. Вона повністю офіційно підтримується Google для розробки застосунків на Android. Kotlin зменшує кількість шаблонів, роблячи код більш читабельним та легшим у підтримці, ніж Java. Такі функції, як нульова безпека та незмінність, роблять програми на Kotlin менш схильними до помилок, таких як виключення за нульовим вказівником.

Відмінна підтримка в Android Studio та інших IDE від JetBrains, що забезпечує безперебійний процес розробки. Kotlin включає такі можливості, як функції розширення, функції вищого порядку та виведення типів, які дозволяють писати більш виразний та гнучкий код.

І JavaScript, і Kotlin пропонують унікальні переваги, пристосовані до їхніх відповідних областей: JavaScript для створення багатих, інтерактивних

вебпрограм, а Kotlin для розробки надійних мобільних застосунків, що легко підтримуються, що робить їх найкращим вибором у своїх галузях.

3.2 Програмна реалізація клієнтської частини застосунку

У даній роботі реалізація інтерфейсу користувача виконується за допомоги мови гіпертекстової розмітки HTML та каскадних таблиць стилів CSS. HTML забезпечує структуру сторінки, тоді як CSS відповідає за презентацію, дозволяючи задавати елементи дизайну, такі як макети, кольори та шрифти. Ця комбінація дозволяє розробляти візуально привабливі та функціонально багаті користувацькі інтерфейси.

Перша сторінка представляє собою декілька блоків інформації, що можуть знадобитися клієнту, дизайн інтерфейсу чистий і сучасний, з чіткою візуальною ієрархією, яка допомагає в навігації. На рисунку 3.4 можна побачити у верхній частині сторінки розміщено назву та адресу ресторану «Київ, вул. Грінченка Бориса, 5А» жирним шрифтом, що робить його одразу помітним.

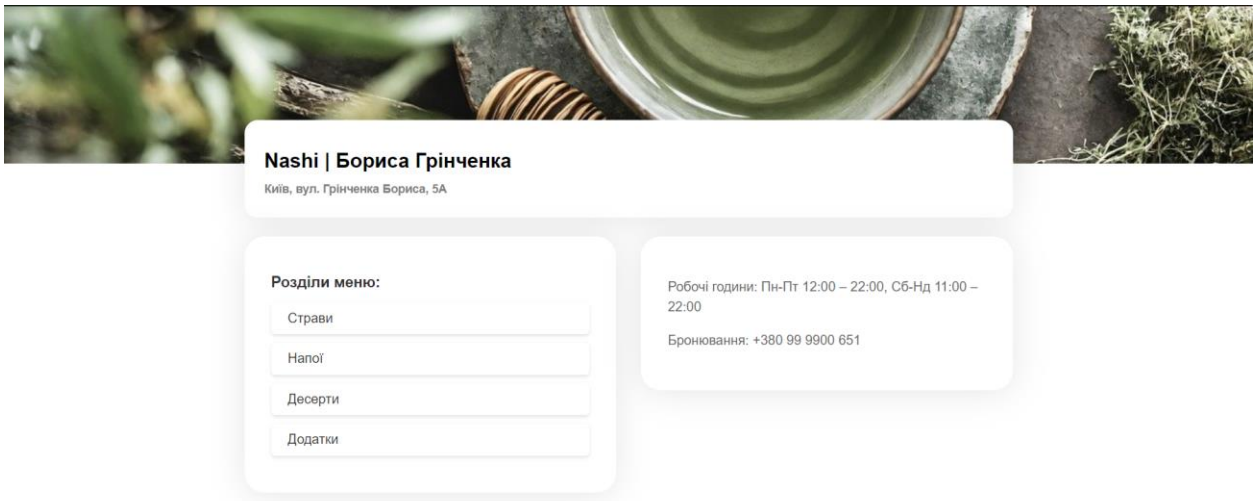

Рисунок 3.4 – Інтерфейс початкової сторінки сайту для користувача

Далі розміщена основне навігаційне меню, структуроване по вертикалі. Воно включає в себе різні розділи меню: «Страви», «Напої», «Десерти» та «Додатки». Кожен розділ є клікабельним, що веде до більш детальної інформації. З правого боку є блок з операційною інформацією про ресторан, що включає в себе години роботи: «Пн-Пт: 12:00 - 22:00, Сб-Нд: 11:00 - 22:00.» та номер телефону для бронювання столиків «+380 99 9000 651».

Після ознайомлення з інформацією, клієнт обирає потрібний йому розділ меню та переходить до нього за допомогою кнопки з відповідною до назвою. Після переходу на нову сторінку ми бачимо обраний нами розділ меню, а також зеленим кольором підкреслений активний наразі підрозділ «Рамени», що має в собі перелік страв (рис. 3.5).

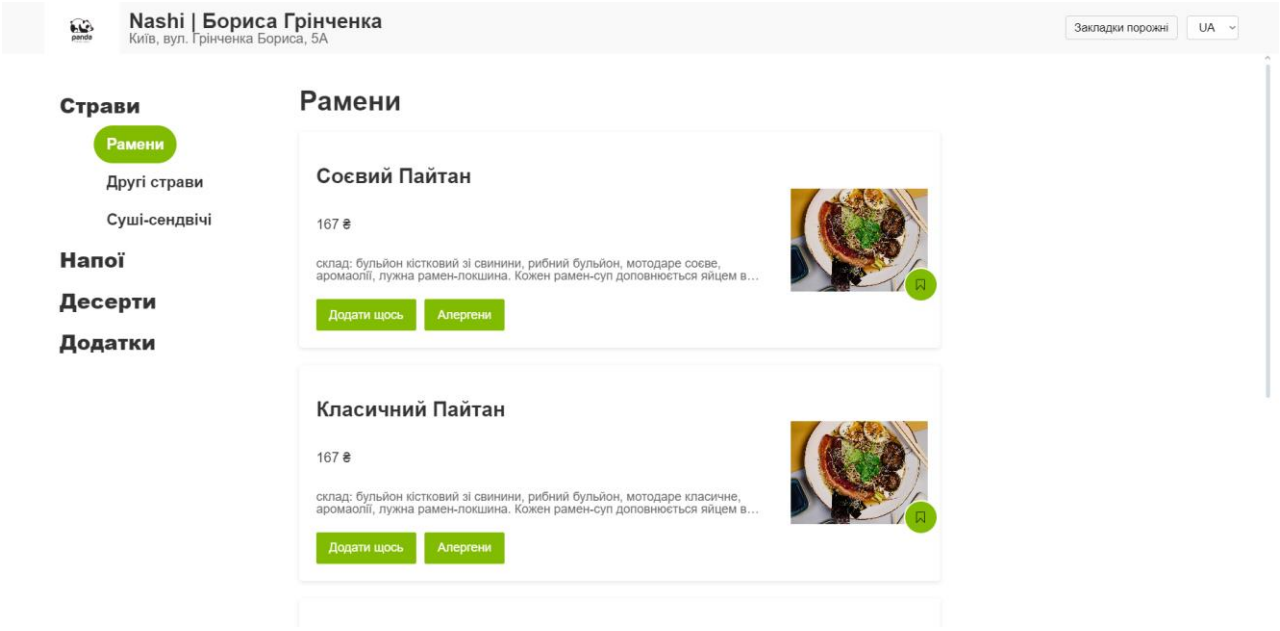

Рисунок 3.5 – Інтерфейс сторінки меню

При наведенні курсором миші на інші елементи списку підрозділів, відбувається підсвічення обраної користувачем назви, для кращого візуального сприйняття. Також присутні позиції меню, а саме страви, для кожної з них вказаний перелік даних, що приходять з бази даних, оскільки

інгредієнти, назви їжі, фото і все інше може змінюватися з часом. Тож, через мінливість такого роду даних були прийнято рішення не прописувати вручну кожну позицію в меню, а реалізувати це за допомогою JavaScript та Cloud Firestore.

Кожна страва, що зберігається в меню має певні параметри, їх можна спостерігати на рисунку 3.6.

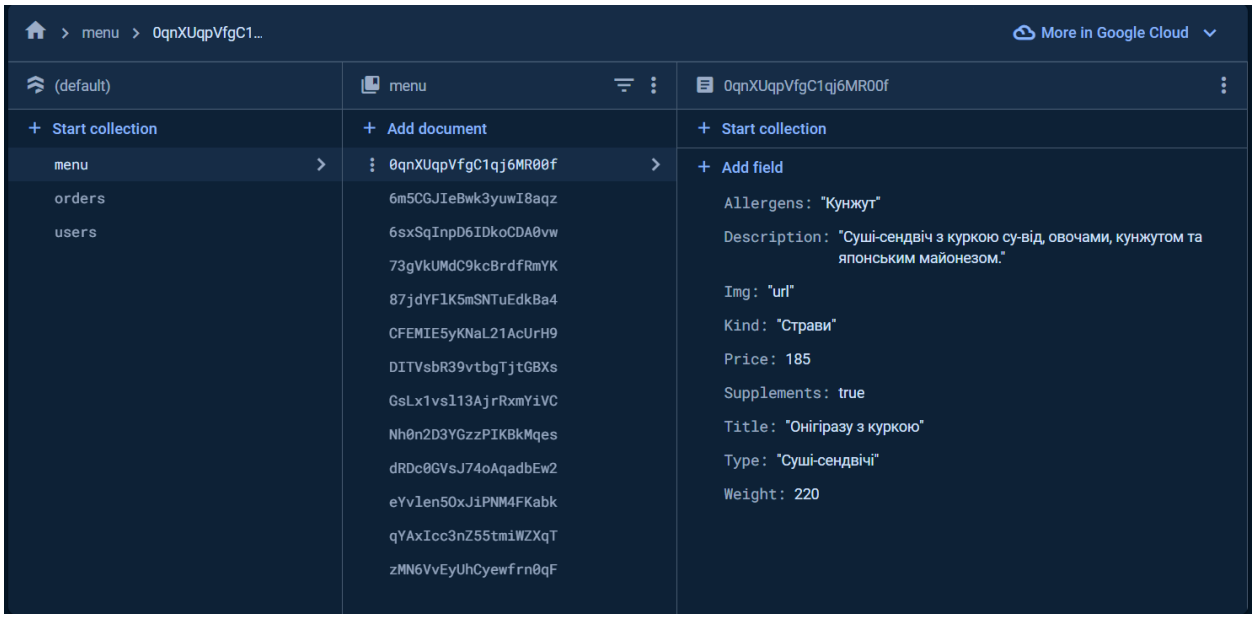

Рисунок 3.6 – Представлення опису страви в базі даних

Для виведення на сторінку страв з меню використовується певний алгоритм, що проходить по базі даних шукаючи елемент що матиме в собі відповідні до обраного клієнтом розділу та підрозділу поля «Kind» і «Type» відповідно. Спочатку ми фільтруємо всі наявні записи в базі даних за наявністю потрібного нам розділу «Kind», а потім шукаємо відповідність до активного наразі поля «Type», після чого виводиться вся потрібна інформація.

Лістинг 3.1 Приклад реалізації функцій перевірки поля «Type»:

```
async function getDishByType(db,type) {
    const q = query(collection(db, "menu"), where("Type", "==", type));
```

```
    const querySnapshot = await getDocs(q);
    const cityList = querySnapshot.docs.map(doc => doc.data());
     return cityList;
}
```

Після подібного сортування відбувається пошук елемента з id «cardsContainer», що в подальшому буде містити в собі всі наші вибрані об’єкти. Використовуючи оператор обходу елементів колекції «foreach», покроково створюємо код в файлі з HTML.

Лістинг 3.2 Приклад реалізації елемента меню:

```
function createCard(db) {
 const card = document.createElement(‘div’);
 card.className = ‘card’;
 card.role = ‘button’;

 card.innerHTML = `
   <div class="card-content">
     <div class="card-text">
       <h2 class="card-title">${db.Title}</h2>
       <p class="card-price-range">${db.Price} ₴</p>
       <p class="card-description truncated">${db.Description}</p>
       <div class="card-actions">
         <button class="card-action-btn">Додати щось</button>
         <button class="card-action-btn">Алергени</button>
       </div>
     </div>
     <div class="card-image">
       <img src="${db.Img}" alt="${db.Title}">
```

```
<button data-v-56ed61c0="" data-v-74224776="" class="basket-button order-mark order-mark--empty"><img src="bookmark-svgrepo-com.svg"></button>
</div>
</div> `;
```

Задля збереження вигляду меню було зроблено автоматичне обмеження для тексту, тобто текст довший за два рядки тексту буде скорочено. Це зроблено для того, щоб зберегти візуальну симетрію. В разі, коли клієнт захоче переглянути повний склад чи опис певної позиції, він може просто натиснути на будь-яку точку, що належить до певної страви і отримати pop-up вікно, де ця інформація вказана повністю (рис. 3.7).

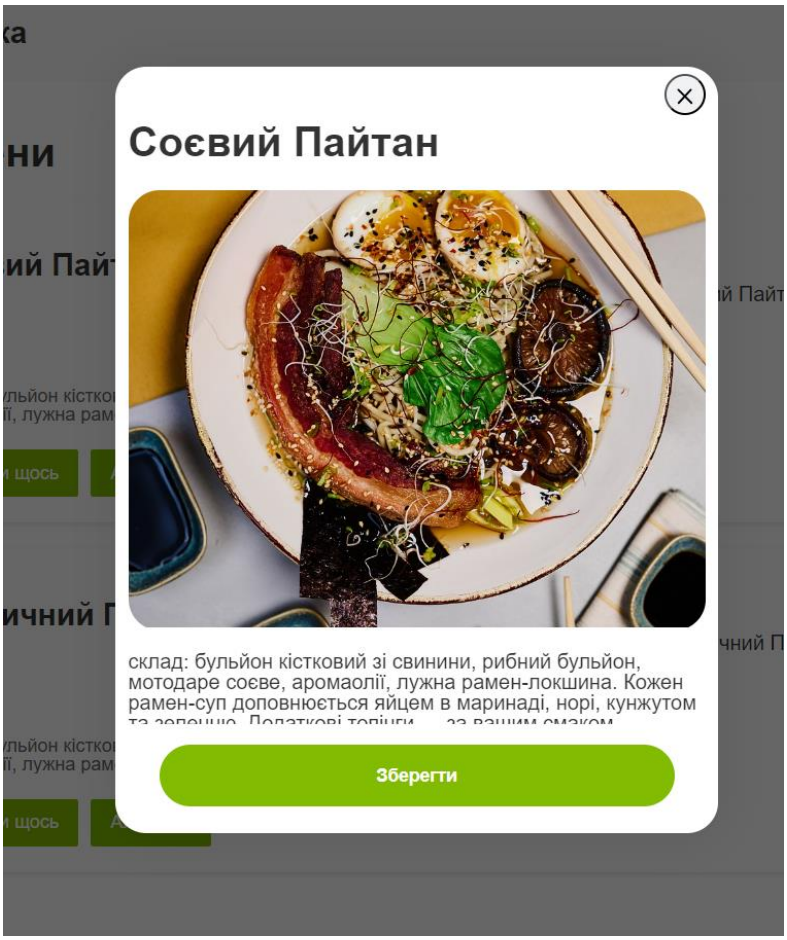

Рисунок 3.7 – Демонстрація модального вікна з інформацією про продукт

Надання доступу до детальної інформації про інгредієнти страви є важливим, оскільки як основні так і допоміжні речовини, такі як ароматизатори, консерванти, і кольорові добавки, можуть викликати алергічні реакції. Проте при великому описі продукту, око людини може не помітити певний інгредієнт, тому було продумано, що при натисканні на кнопку з назвою «Алергени» користувач може дізнатися усі алергени, що наявні в конкретній страві.

Також, для зручності присутня кнопка «Додати щось». При натисканні на неї відкривається перелік додатків доступних до конкретної страви, що дозволить клієнту урізноманітнити страву та підвищити суму чеку.

Біля кожного фото продукту знаходиться кнопка призначена для збереження обраного продукту в «Закладках». Неактивний елемент має сіре забарвлення, а вже після додавання позиції у вибране, він міняє колір на зелений, щоб користувачу було легше орієнтуватися в тому, що він вже обрав (рис. 3.8).

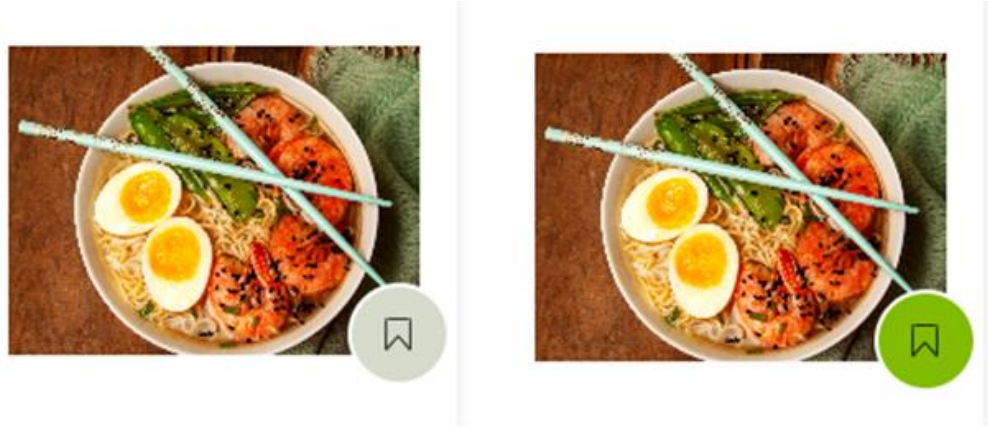

Рисунок 3.8 – Представлення двох станів кнопки збереження

За подібним принципом діє функціональний елемент «Закладки», що розроблений для зручності користувача. При першому візиті, коли клієнт нічого не обрав, кнопка є ледве помітною та не активною (рис. 3.9). При натисканні на неї не відбувається жодних змін на сторінці та переходів на

інші сторінки сайту. Після того, як було обрано одну або декілька позицій з меню, кнопка змінює свій стан та відображає кількість обраних страв (рис. 3.9). Коли елемент активний, він дозволяє перейти до наступної сторінки, де зберігається сформоване замовлення, яке користувач може редагувати.

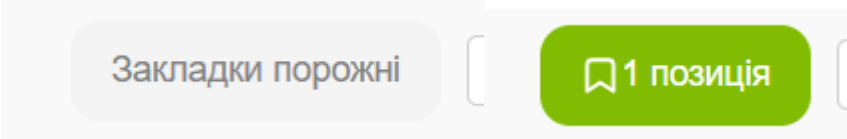

Рисунок 3.9 – Представлення двох станів кнопки «Закладки»

При зміні кількості товарів в сховищі оновлюється стан об’єкту. Дана зміна реалізована за участі LocalStorage, що дозволяє зберігати дані обрані чи введені клієнтом на стороні сайту та дає можливість не втратити їх через проблеми з інтернетом чи після перезавантаження сторінки. Дані зберігаються у браузері користувача, тож все що збережено в LocalStorage, не має терміну дії і залишається доступними навіть після закриття браузера.

Тобто дані зберігаються безстроково, доки не будуть видалені вручну, або не буде виконана дія очищення. В нашому випадку після оплати замовлення воно деактивує дані в сховищі методом clear(). Максимальний обсяг збережених даних для одного домену зазвичай обмежений 5-10 МБ. А самі дані доступні лише для домену, який їх зберіг.

Також було реалізовано кошик, для перегляду, редагування та завершення замовлення. Користувач може додавати товари до кошика, натискаючи на відповідну кнопку. Після додавання товару до кошика, дані про нього (назва, ціна, кількість тощо) зберігаються у LocalStorage браузера. Користувач може змінювати кількість товарів у кошику, збільшуючи або зменшуючи їх кількість. Ця інформація також оновлюється в LocalStorage. Можна видаляти товари з кошика, які більше не потрібні. Після видалення, відповідні дані також видаляються з LocalStorage.

Сторінка з кошиком відображає всі додані товари разом з їхньою інформацією, яка береться з LocalStorage. Користувач може бачити загальну кількість товарів у кошику та їхню загальну вартість.

3.3 Програмна реалізація застосунку для персоналу на базі Android

Операційну систему Android було обрано з міркувань того, що застосунок буде встановлено на планшети та телефони ресторанів, а це частіше за все, сама смартфони на базі Android, так як вони дешевше. Для безпосереднього написання коду було обрано мову Kotlin [28].

Однією з ключових особливостей Kotlin є його механізми для уникнення помилок, пов’язаних з null, які є однією з найпоширеніших причин збоїв у Java-програмах. У Kotlin існують спеціальні конструкції, що допомагають запобігти NullPointerException, що робить код більш надійним.

Також варто відзначити, що код на Kotlin зазвичай коротший і зрозуміліший у порівнянні з Java. Це досягається завдяки більш лаконічному синтаксису та використанню сучасних програмних концепцій, таких як лямбда-вирази, розширювальні функції та інші.

Загалом, Kotlin надає потужні інструменти для розробки Android-застосунків, що дозволяють писати більш чистий, безпечний та ефективний код.

Для написання усієї розмітки у застосунку було обрано бібліотеку Jetpack Compose. Вона дозволяє писати всю розмітку за допомогою Kotlin, використовуючи декларативний підхід, що робить код більш чистим і читабельним [29].

Однією з головних переваг Jetpack Compose є те, що він позбавляє необхідності використовувати XML для створення інтерфейсів, як це було у традиційному підході. Це означає, що весь інтерфейс користувача можна

створити, редагувати та підтримувати в межах одного файлу Kotlin, що значно спрощує роботу розробників.

Jetpack Compose також підтримує інструменти для побудови динамічних інтерфейсів, які можуть автоматично оновлюватися при зміні стану застосунку. Це дозволяє створювати більш інтерактивні та чуйні інтерфейси з меншою кількістю коду. Інтеграція з іншими компонентами Jetpack, такими як ViewModel і LiveData, дозволяє легко працювати з архітектурними компонентами, дотримуючись принципів MVI (Model-View-Intent).

Екрани були побудовані дотримуючись принципів MVI та використовуючи для цього бібліотеку OrbitMvi. MVI – це архітектурний патерн, який допомагає створювати передбачувані та тестовані інтерфейси користувача. Він забезпечує чітке розділення між станами, подіями користувача та відображенням інтерфейсу, що полегшує тестування та підтримку коду.

OrbitMVI допомагає реалізувати MVI-патерн, надаючи інструменти для управління станами та обробки подій. Вона спрощує роботу з асинхронними потоками, дозволяючи декларативно описувати зміни станів та забезпечуючи реактивність інтерфейсу.

3.3.1 Авторизація у застосунок

При вході в застосунок йде перевірка чи був вже авторизований користувач, якщо користувач вже був авторизований, то його ведуть на екран очікування, про який буде пізніше. У разі, якщо користувач ще не авторизований, то відбувається навігація на екран логіну.

Лістинг 3.3. Реалізація перевірки на наявність акаунту:

```
private fun loadUser() {
```

```
    val user = firebaseAuth.currentUser

    if(user == null){
        postSideEffect(SplashSideEffect.ShowLogin)
    } else {
        postSideEffect(SplashSideEffect.ShowWaitingScreen)
    }
}
```

На екрані логіну користувачу необхідно ввести свою пошту і пароль для подальшого доступу до застосунку (рис. 3.10).

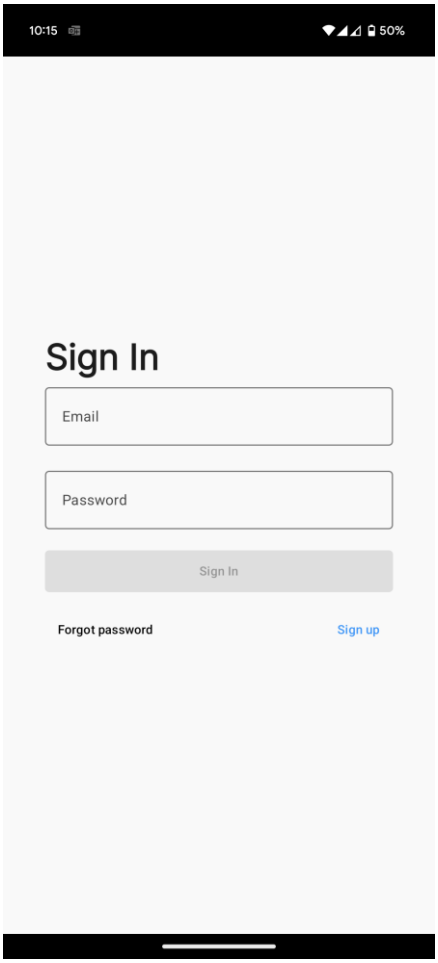

Рисунок 3.10 – Екран логіну у застосунок

Для паролю і для пошти також підключено валідацію, тому кнопка входу буде недоступна для натискання якщо пошта не відповідає шаблону, або пароль має довжину менше 6 символів і в ньому відсутні цифри. Для валідації пошти було використано базовий метод валідації з Android SDK.

Лістинг 3.4 Реалізація валідації пароля і пошти:

```
val isEmailValid = remember(emailValue) { PatternsCompat.EMAIL_ADDRESS.matcher(emailValue).matches() }
val isPasswordValid = remember(passwordValue) { passwordValue.length > 6 && passwordValue.any { it.isDigit() } && passwordValue.any { it.isLetter() } }
```

Після успішної локальної перевірки на логін і пароль дані відправляються на сервіс Firebase Authentication, де вони перевіряються на відповідність наявним акаунтам на сервісі, у разі успішної перевірки користувача переводить на екран очікування, у разі провалу виводить відповідне повідомлення.

Лістинг 3.5 Метод відправки даних на сервер і очікування результату:

```
private fun logInUser(email: String, password: String) {
  viewModelScope.launch {
    firebaseAuth.signInWithEmailAndPassword(email, password)
      .addOnCompleteListener { task ->
        if (task.isSuccessful) {
          // Sign in success, update UI with the signed-in user’s information
          Timber.d("signInWithEmail:success")
          val user = firebaseAuth.currentUser

          postSideEffect(LoginSideEffect.NavigateToWaitingPage)
        } else {
```

```
                // If sign in fails, display a message to the user.
                Timber.w("signInWithEmail:failure", task.exception)

                postSideEffect(LoginSideEffect.ShowErrorMessage)
            }
        }
    }
}
```

Також, якщо у користувача відсутній обліковий запис, він може перейти на екран реєстрації натиснувши на кнопку «Sign up». На екрані реєстрації користувачу необхідно буде вказати повне ім’я, пошту та двічі вказати бажаний пароль (рис 3.11)

Після введення усіх даних користувачу стане доступна копка «Sign up», вона, так само як і на екрані логіну, перебуває в неактивному стані до моменту заповнення усіх полів і проходження всіх інших валідацій.

Лістинг 3.6 Валідація на екрані реєстрації:

```
enabled = emailValue.isNotBlank()
&& PatternsCompat.EMAIL_ADDRESS.matcher(emailValue).matches()
&& passwordValue.isNotBlank()
&& passwordValue == secondPasswordValue
&& fullNameValue.isNotBlank()
&& passwordValue.length > 6
&& passwordValue.any { it.isDigit() }
&& passwordValue.any { it.isLetter() }
```

Далі відбувається відправка даних у сервіс Firebase Authentication та збереження користувача у базі даних, саме другий доволі важливий, так як

саме через цю базу буде відбуватися налаштування доступів користувача і визначення його ролі.

Рисунок 3.11 – Екран реєстрації у застосунку

Лістинг 3.7 Процес реєстрації користувача:

```
private fun onRegisterUser(
userName: String,
email: String,
password: String,
repeatedPassword: String,
```

```
) {

    // Add a new document with a generated id.
    val data = hashMapOf(
        "fullName" to userName,
        "email" to email,
    )

    firestore.collection("users")
        .add(data)
        .addOnSuccessListener { documentReference ->
            Timber.d("DocumentSnapshot written with ID: ${documentReference.id}")
            firebaseAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        // Sign in success, update UI with the signed-in user's information
                        Timber.d("createUserWithEmail:success")
                        postSideEffect(SignUpSideEffect.OnNavigateToWaitScreen)
                    } else {
                        // If sign in fails, display a message to the user.
                        Timber.w("createUserWithEmail:failure", task.exception)
                        postSideEffect(SignUpSideEffect.OnShowError)
                    }
                }
        }
        .addOnFailureListener { e ->
            postSideEffect(SignUpSideEffect.OnShowError)
            Timber.w("Error adding document", e)
        }
}
```

Після успішного збереження і реєстрації користувача переводить на екран очікування статусу.

3.3.2 Основні екрани застосунку

Після успішного логіну чи реєстрації користувач одразу переходить на екран очікування (рис. 3.12), де він очікує доки адміністратор наддасть йому статус, на поточний момент їх два:

– waiter: статус офіціанта;

– admin: статус адміністратора

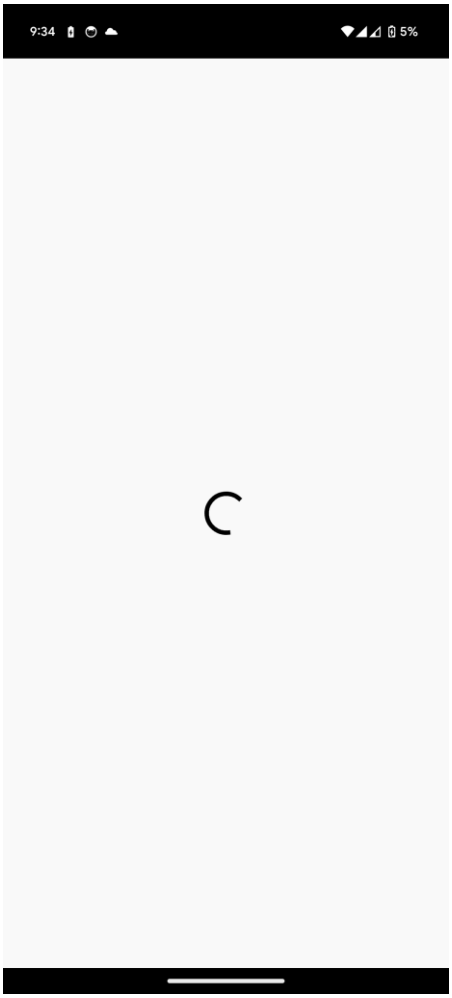

Рисунок 3.12 – Екран очікування на статус

Знаходячись на цьому екрані застосунок використовує механізм подібний до механізму веб-сокетів, тобто постійно знаходиться в статусі очікування відповіді з серверу для продовження. У поточному випадку роль серверу виконує Firestore Database, до бази даних якої підключено застосунок і по пошті користувача надіслано запит на підпис щодо оновлення даних по ньому. Тобто, як тільки дані оновляться, застосунок отримає нову модель даних по цьому користувачу і перевірить чи не оновився його статус, якщо оновився, то цей статус перевіряється на відповідність поточним і користувача переводить на потрібний екран. З недоліків такого підходу можна виділити те, що він не захищений від одруків на стороні адмін-частини, і тому застосунок не зможе розпізнати роль. Ще одним недоліком є той факт, що при умові додавання нових ролей, старі версії застосунку не будуть підтримувати ці ролі, але конкретно це питання вирішується вимогою щодо оновлення застосунку.

Лістинг 3.8 Реалізація підключення до бази даних і підписки на оновлення даних для конкретного користувача:

```
private fun onLoadData() {
    val currentUser = firebaseAuth.currentUser

    if (currentUser != null) {
        firestore.collection("users")
            .whereEqualTo("email", currentUser.email)
            .addSnapshotListener { value, e ->

                if (e != null) {
                    Timber.w("Listen failed.", e)
                    return@addSnapshotListener
                }
```

```
                for (doc in value!!) {
                    doc.getString("role")?.let {
                        if (it == "Admin") {
                            postSideEffect(WaitingSideEffect.OnNavigateToAdmin)
                        } else if (it == "Waiter") {

postSideEffect(WaitingSideEffect.OnNavigateToWaiterPage)
                        } else {
                        }
                    }
        }
        }
        }
        }
```

У разі, якщо користувач має роль «waiter», йому надається доступ до сторінки офіціанта (рис. 3.13). На цій сторінці офіціант може побачити усі поточні замовлення, які в нього є, на графіку подивитися які години були найбільш успішними в плані виручки, та подивитися загальну статистику кількості замовлень і чайових за день. Також, офіціант може переглянути таку саму статистику за будь який інший день, натиснувши на кнопку «Choose period» або переглянути її за вчора, натиснувши на «Yesterday».

Варто зазначити, що офіціант бачить активні замовлення не по принципу закріплення до нього, а по принципу закріплення за столиком, таким чином столик може обслуговуватись одразу двома офіціантами чи навіть більшою кількістю. Чайові в поточній версії будуть розділятися вже власноруч, але в майбутньому можлива автоматизація шляхом розділення, або надання їх тому, хто закрив чек, це може бути на розсуд конкретного ресторану.

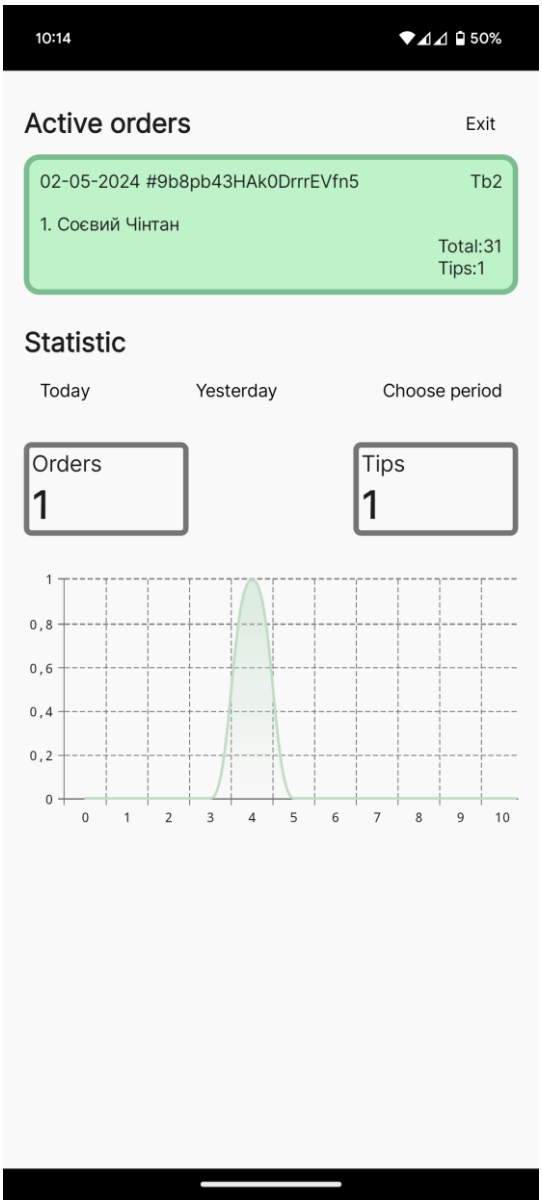

Рисунок 3.13 – Екран застосунку для офіціанта

У майбутніх версіях системи передбачається впровадження більшої автоматизації для оптимізації роботи персоналу та підвищення рівня обслуговування клієнтів. Однією з таких функцій може стати автоматичне розподілення замовлень між офіціантами в залежності від їхньої завантаженості та статусу столиків. Це допоможе рівномірно розподіляти робоче навантаження та зменшити час очікування для клієнтів.

Для визначення які столики належать якому офіціанту так само використовується вже існуючий запис про офіціанта в базі, де в нього з’являється список з номерами столиків (рис. 3.14).

Рисунок 3.14 – Вигляд запису в базі даних, де видно поточні столики офіціанта

Екран адміністратора не дуже відрізняється по функціоналу і інформативності від екрану для офіціанта. Адміністратору так само доступні до перегляду усі замовлення, але для нього немає обмеження щодо столику і щодо того, чи активне замовлення, він бачить їх усі за необхідний день (рис. 3.15).

Також адміністратору доступний графік на якому він може бачити загальну кількість замовлень у певну годину, це допомагає визначати у які години варто мати найбільшу кількість персоналу, а у які можна відпустити їх на обід чи взагалі закінчити їх зміну.

Адміністратор може обрати день, за який хоче переглянути графік і загальну статистику, єдиною відмінністю є те, що він бачить окрім кількості замовлень і чайових ще і загальну виручку без урахування чайових.

Для відображення графіку було використано бібліотеку Vico charts, вона дає дуже зручний і простий в освоєнні функціонал відображення графіків. Єдиним мінусом бібліотеки є доволі складна до розуміння документація, яка використовує багато термінів з графіки, які не кожен розробник знає. Також присутня певна проблема з прикладами, які необхідно актуалізувати через те, що в другій версії дуже багато кардинальних змін.

Попри ці недоліки, Vico charts залишається потужним інструментом для створення візуалізацій даних. Бібліотека підтримує широкий спектр типів

графіків, таких як лінійні графіки, стовпчикові діаграми, кругові діаграми та багато інших [30].

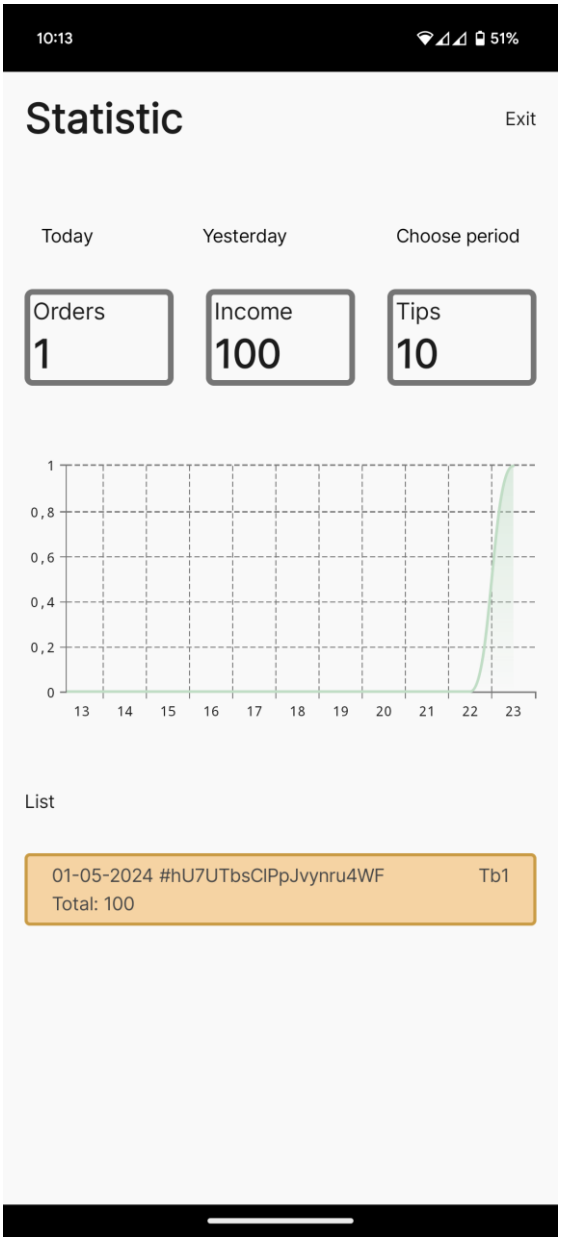

Рисунок 3.15 – Екран адміністратора

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено та реалізовано вебзастосунок для оформлення замовлення та оплати в закладах харчування. Головними перевагами реалізованого сервісу є сучасний, інтуїтивно зрозумілий для користувача. Також було створено мобільний застосунок задля зручності бізнесу, що також є перевагою, оскільки це надає можливість відслідковувати статистку та приймати нові рішення, спираючись на отримані дані.

Для цього були вирішені такі завдання:

– проведено аналіз наявних застосунків для оформлення та оплати замовлень для закладів харчування;

– розроблено та реалізувати зручний інтерфейс користувача для інтуїтивно зрозумілого використання сайту;

– розроблено архітектуру програми, створити логіку для обробки та управління замовленнями, включаючи створення, відстеження та аналіз замовлення;

– створено та інтегрувати базу даних для збереження інформації про продукцію ресторану;

– розроблено застосунок для адміністрації бізнесу та робітників закладу, щоб надати можливість відстеження.

Під час розробки були використані такі технології як: HTML, CSS, JavaScript, Kotlin, Cloud Firestore. А в майбутньому це список може розширитися, оскільки існує багато шляхів розвитку проєкту. Першочерговим є вдосконалення та розширення функціоналу застосунку для бізнесу, а саме додання можливості самостійно створювати та редагувати меню закладу, це зробить навантаження на розробників меншим та зробить утримання проєкту дешевшим.

Результати роботи апробовано у вигляді тез доповідей під час XXVIII Міжнародного молодіжного форуму «Радіоелектроніка і молодь у XXI столітті» [1].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Цехмістренко К. В. (2024). Розробка застосунку для оформлення замовлення та оплати в закладах харчування. Діджіталізація бізнесу. Радіоелектроніка та молодь у XXI столітті: 28 міжнародний молодіжний форум, 136-137.

2. Тітов С. В., & Тітова О. В. (2014). Інформаційно-освітнє середовище навчального закладу: розвиток засобів і способів комунікаційної й інформаційної взаємодії. Вісник Харківської державної академії культури, (43), 144-150.

3. Тітов С. В., & Тітова О. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології. Вісник Харківської державної академії культури. Серія: Соціальні комунікації, (47), 127-134.

4. QR Меню, яке працює на вас. URL: https://oddmenu.com/uk (дата звернення 10.05.2024).

5. Один QR-код для всього. URL: https://expz.monobank.ua/?lang=uk (дата звернення 10.05.2024).

6. Choice: Розумні рішення для ресторанного бізнесу. URL: https://choiceqr.com/uk/ (дата звернення 10.05.2024).

7. TAKAVA. URL: https://expz.menu/259c88f9-ff72-4a7b-934c-ef27df53dbf6 (дата звернення 10.05.2024).

8. The Burger. URL: https://theburger-menu.choiceqr.com/ (дата звернення 10.05.2024).

9. International standard ISO 9241-210 Second edition (2019). URL:https://cdn.standards.iteh.ai/samples/77520/8cac787a9e1549e1a7ffa0171dfa33e0/ISO-9241-210-2019.pdf (дата звернення 13.05.2024).

10. Кобилін О. А., & Творошенко І. С. (2021). Методи цифрової обробки зображень.

11. XML Views or Jetpack Compose: Which is The Best Option For Your Next Project? URL: https://medium.com/@MeenoTeK/xml-views-or-jetpack-compose-which-is-the-best-option-for-your-next-project-ccf38573a82 (дата звернення 13.05.2024).

12. Documentation Jetpack Compose. URL: https://developer.android.com/develop/ui/compose/documentation/ (дата звернення 16.05.2024).

13. The Contract of the Model-View-Intent Architecture. URL: https://proandroiddev.com/the-contract-of-the-model-view-intent-architecture-777f95706c1e (дата звернення 16.05.2024).

14. Kinoshenko D., Kobylin O., Mashtalir S., & Stolbovyi M. (2019, March). Metric video retrieval speedup by irrelevant data elimination. In Eleventh International Conference on Machine Vision (ICMV 2018) (Vol. 11041, pp. 176-183). SPIE.

15. Tvoroshenko, I. S., & Kuznetsov, M. (2021). About the role of testing in process of mobile application development.

16. Lyashenko V., Mohammad A., & Kobylin O. (2015). Experiments with Fusion of Images with Use of Wavelet Transformation in Problems of the Text Information Analysis.

17. HTML: HyperText Markup Language. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення 18.05.2024).

18. CSS: Cascading Style Sheets. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення 18.05.2024).

19. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення 18.05.2024).

20. Mongo DB — The NoSQL database: Installation and basic CRUD operations. URL: https://deepikasrinivasasharma.medium.com/mongo-db-the-nosql-database-installation-and-basic-crud-operations-b121ab2fd415 (дата звернення 20.05.2024).

21. Kobylin O., Vyskrebentseva S., & Petrova R. (2019). Обробка даних, що містять пропуски в задачах кластеризації. Системи управління, навігації та зв'язку. Збірник наукових праць, 5(57).

22. Firebase create realtime database. PathJson extension. Firebase from web page. URL: https://community.appinventor.mit.edu/t/firebase-create-realtime-database-pathjson-extension-firebase-from-web-page/45664 (дата звернення 20.05.2024).

23. Tvoroshenko, I., & Kharchenko, A. (2021). Some aspects of modern development for sign language recognition systems. International Journal of Machine Learning and Cybernetics. Advance online publication.

24. Тітов С.В., Тітова О.В., Чорна О.С. (2022). Опис нескоротних наборів ознак в приблизних множинах з використанням систем числення. Збірник наукових праць Харківського національного університету Повітряних Сил. № 1(71), с. 106-110.

25. Daradkeh Y.I., Tvoroshenko I., Gorokhovatskyi V., Latiff L.A., and Ahmad N. (2021). Development of Effective Methods for Structural Image Recognition Using the Principles of Data Granulation and Apparatus of Fuzzy Logic. IEEE Access, 9, pp. 13417-13428.

26. Daradkeh, Y. I., Gorokhovatskyi, V., Tvoroshenko, I., & Zeghid, M. (2022). Tools for Fast Metric Data Search in Structural Methods for Image Classification. IEEE Access, 10, 124738-124746.

27. Частка ринку Android та iOS: оприлюднено статистику 2022 року. URL: https://root-nation.com/ua/news-ua/it-news-ua/ua-android-ios-statistika-2022/ (дата звернення 21.05.2024).

28. Tvoroshenko, I., & Almakaieva, A. (2020). Application of procedural generation of game content using software algorithms. Proceedings of the 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA), 1-5.

29. Tvoroshenko, I., & Temchur, K. (2021). Features of software application development for food recognition using deep machine learning methods. Technical Sciences: Theoretical and Applied Aspects of the Development of Science, 511.

30. Vico introduction. URL: https://patrykandpatrick.com/vico/wiki/ (дата звернення 26.05.2024).