МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Центр заочної, дистанційної та вечірньої форм навчання

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

11 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформаційна технологія проєктування мобільного додатку для закладу ресторанного господарства»

здобувача групи ІН.мз - 21с Шаргіна Ярослава Володимировича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Ярослав ШАРГІН

(підпис)

Керівник
асистентка кафедри комп'ютерних наук,
к.ф.-м.н. Ольга ШУТИЛЄВА _____

(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.мз – 21с Шаргін Ярослав Володимирович

1. Тема роботи: «Інформаційна технологія проєктування мобільного додатку для закладу ресторанного господарства»
затверджую наказом по СумДУ від *«20» листопада 2023 р. № 1308-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 13 грудня 2023 року*
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз предметноъ області. 2) Аналіз аналогів для додатку та визначення найважливіших концепцій. 3) Визначення оптимального програмного рішення для реалізації поставленої задачі. 4) Програмна реалізація додатку. 5) Оформлення пояснювальної записки до кваліфікаційної роботи..
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 2023 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз предметної області.*		
2	*Аналіз аналогів для додатку та визначення найважливіших концепцій.*		
3	*Визначення оптимального програмного рішення для реалізації поставленої задачі.*		
4	*Програмна реалізація додатку.*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи.*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 64 стор., 16 рис., 1 таб., 4 додатки, 27 використаних джерел.

Обґрунтування актуальності теми роботи: У сучасному світі інформаційні технології стають невід'ємною частиною будь-якої сфери, включаючи ресторанний бізнес. Завдяки розвитку мобільних технологій, ресторани можуть пропонувати свої послуги більш ефективно та зручно. Розроблення мобільного додатку для ресторанів представляє собою важливий та актуальний крок у поліпшенні сервісу та забезпеченні комфорту для клієнтів.

Об'єкт дослідження: Процес взаємодії клієнтів з ресторанами через мобільні додатки.

Предмет дослідження: Технологія мобільного додатку для ресторанів, що дозволяє користувачам з легкістю отримувати інформацію про послуги, меню, спеціальні пропозиції, та можливості бронювання.

Мета роботи: Розробка та впровадження мобільного додатку для ресторанів, спрямованого на поліпшення якості обслуговування та оптимізацію взаємодії з клієнтами.

Методи дослідження: Вивчення та аналіз існуючих мобільних додатків для ресторанів, проектування інтерфейсу користувача, розробка функціоналу, тестування та оцінка ефективності додатку.

Результати: Досліджено технології проєктування мобільного додатку, розроблено мобільний додаток, який дозволяє користувачам отримувати швидкий доступ до інформації про ресторани, зручно бронювати столики, переглядати меню та акції, що призводить до підвищення задоволеності клієнтів та зростання лояльності до ресторану.

ІНФОРМАЦІЙНА СИСТЕМА, API,
NODE JS, REACT NATIVE, БАЗА ДАНИХ, АНДРОЇД

ЗМІСТ

ВСТУП

Інтернет став невід'ємним елементом нашого щоденного життя. Практично кожен з нас щоденно звертається до всесвітньої мережі для здобуття інформації, проведення фінансових операцій або для комунікації. У цьому контексті, ресторанний бізнес також адаптувався до цифрової ери, створюючи мобільні додатки, які служать не тільки для замовлення їжі, але й як зручний інструмент для взаємодії з клієнтами.

У епоху цифровізації, мобільні додатки для ресторанів стають незамінними помічниками як для бізнесу, так і для клієнтів. Вони не лише полегшують процес замовлення та доставки їжі, але й створюють платформу для інтерактивного спілкування з відвідувачами. Через додатки можна отримувати зворотний зв'язок від клієнтів, що допомагає вдосконалювати сервіс і меню. Красивий дизайн та інформативність мобільних додатків підвищують шанси на те, що ресторан буде сприйматися як якісний і популярний.

Окрім того, в умовах глобальних викликів, як-то пандемії, ці додатки надають ресторанам можливість підтримувати зв'язок зі своєю аудиторією, незважаючи на обмеження в переміщенні. Це сприяє підтримці постійного потоку клієнтів, навіть коли фізичний доступ до закладу обмежений.

Додатково, мобільні додатки ресторанів відкривають можливості для особистісного маркетингу, дозволяючи надсилати персоналізовані пропозиції та знижки, а також інформувати про особливі події та новинки меню. Це не тільки збільшує продажі, але й створює почуття ексклюзивності та приналежності серед клієнтів.

Таким чином, мобільні додатки для ресторанів стають важливим інструментом у сучасному бізнесі. Вони дозволяють компаніям залишатися на зв'язку зі своїми клієнтами, пропонуючи їм зручність, інновації та персоналізований досвід, що є ключовими утримуючими факторами в цій галузі.

1. ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

1.1 Аналіз предметної області

У сучасному світі, де Інтернет є невід'ємною частиною нашого життя, значення мобільних додатків для ресторанів стрімко зростає. Ці додатки, як цифрові візитні картки, надають ключову інформацію про ресторан, включаючи типи кухні, тривалість обслуговування та унікальні характеристики закладу. Вони пропонують зручний спосіб для спілкування з клієнтами, дозволяючи їм зв'язуватися безпосередньо з менеджерами або обслуговуючим персоналом.[6]

На головній сторінці таких додатків зазвичай представлена інформація про актуальні пропозиції, новини та акції, що дозволяє клієнтам швидко ознайомитися з найсвіжішою інформацією. Особливу увагу приділяють дизайну головної сторінки, включаючи логотип ресторану, який має бути яскравим, легко запам’ятовуватися, а також відображати концепцію закладу.[20]

Наступна частина додатка часто присвячена детальному опису послуг, що пропонуються рестораном. Це може включати меню, фотографії страв та опис особливостей кухні. Форма зворотного зв'язку в цій секції дозволяє клієнтам легко залишати замовлення або задавати запитання.

Останній розділ додатка зазвичай містить контактну інформацію, мапу для зручності орієнтування до ресторану, а також посилання на соціальні мережі та інші ресурси, які можуть бути корисні для клієнтів, які хочуть зв'язатися з рестораном або його представниками.

Розвиток такого мобільного додатку для ресторану також може включати розділ для онлайн-бронювання столиків. Ця функція є надзвичайно корисною для клієнтів, які планують свій візит заздалегідь. Вона може містити календар з доступними датами та часом, а також можливість вибору

конкретного столика або зони у ресторані. Це не лише збільшує зручність для клієнтів, але й оптимізує робочий процес персоналу ресторану.[17]

Ще одним важливим аспектом є інтеграція системи відгуків та оцінок у додатку. Клієнти можуть залишати відгуки про свій досвід, страви, обслуговування та загальну атмосферу ресторану. Це не тільки допомагає іншим користувачам прийняти рішення про візит, але й надає ресторану цінну зворотну інформацію для поліпшення своїх послуг.

Додатково, можна включити секцію з рецептами або кулінарними порадами від шеф-кухаря ресторану. Це не тільки розширює функціонал додатку, але й підвищує інтерес клієнтів до кухні ресторану, а також створює додаткову вартість, оскільки надає користувачам можливість спробувати створити щось схоже вдома.

Ще один корисний елемент - це інтеграція з програмами лояльності та акційними пропозиціями. Клієнти можуть отримувати спеціальні знижки, бонуси або накопичувати бали за кожен візит або замовлення через додаток. Це спонукає клієнтів частіше відвідувати ресторан і використовувати додаток для замовлення послуг.

На завершення, важливо включити інтегровані платіжні системи, що дозволяють користувачам безпосередньо через додаток здійснювати оплату за замовлення. Це забезпечує велику зручність і безпеку платіжних транзакцій, а також прискорює процес обслуговування у ресторані. Враховуючи ці різноманітні аспекти, розробка мобільного додатку для ресторану може значно підвищити задоволеність клієнтів та ефективність роботи закладу.

1.2 Аналіз аналогів

Мобільні додатки для ресторанів набули великої популярності в сучасному цифровому світі. Зі зростанням обсягу інформації, розробка таких додатків вимагає використання передових технологій для їх створення та

наповнення контентом.[23] Ці додатки можуть містити розгорнуту інформацію про ресторан, його меню та особливості обслуговування.

Також важливою є естетика мобільного додатку. Привабливий дизайн є ключовим фактором, що впливає на вибір користувачів, адже багато хто цінує зручний інтерфейс та приємне візуальне сприйняття.

Перед початком розробки мобільного додатку для ресторанів було проведено дослідження аналогічних програм: «Піцерія куб», «Corleone» та «KUCHENCHEF».

Кожен з запропонованих додатків відрізняється сучасним дизайном та зручним меню для навігації між різними розділами та привабливою головною сторінкою (рис. 1.1, рис 1.2, рис 1.3). Аналізуючи головну сторінку мобільного додатку для ресторану, можна виділити наступні ключові особливості:

- Інформативність: Наявність чіткої інформації про години прийому замовлень відразу привертає увагу користувача, надаючи важливу інформацію для планування замовлення.
- Інтуїтивність: Навігаційне меню розташоване в центральній частині екрану, що дозволяє користувачеві легко вибирати категорії страв без потреби в довгому пошуку або прокручуванні.
- Доступність: Використання іконок з текстовими мітками робить інтерфейс зрозумілим для широкого спектра користувачів, незалежно від їх віку чи технічної освіченості.
- Візуальна консистенція: Однорідний стиль елементів інтерфейсу та колірна гама сприяють створенню гармонійного візуального сприйняття та підвищують впізнаваність бренду.
- Функціональність: Наявність нижньої навігаційної панелі забезпечує швидкий доступ до основних функцій додатку, таких як перегляд меню, профіль користувача, кошик покупок, історія замовлень та додаткові налаштування.

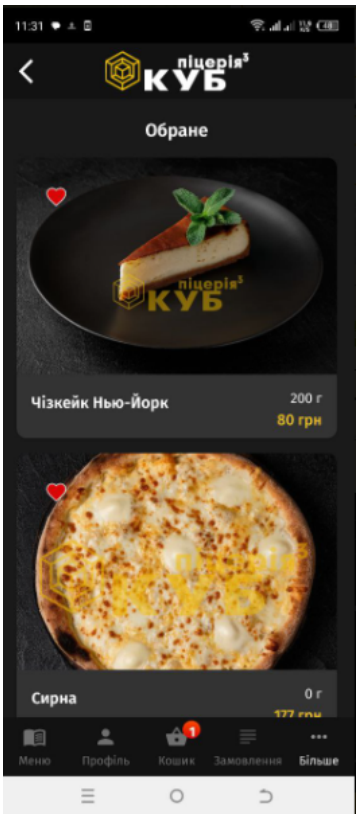

Рисунок 1.1 – Головна сторінка додатку «Піцерія куб»

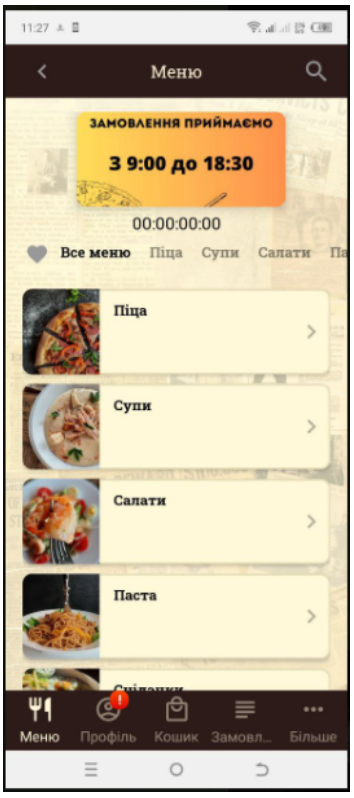

Рисунок 1.2 – Головна сторінка додатку «Corleone»

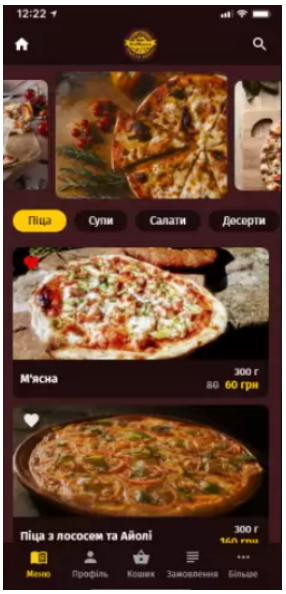

Рисунок 1.3 – Головна сторінка додатку «KUCHENCHEF»

На карточці товару відображається фото продукту, щоб користувач розумів, що саме він замовляє, також назва, короткий опис, вага та ціна обраного блюда та можливість додати в обране або в кошик. У ресторані «KUCHENCHEF» є можливість додати к товару ще набір додатків, додатковий сир або зелень до піци (рис. 1.4, рис. 1.5, рис. 1.6).

Рисунок 1.4 – Карточка товару у додатку «Піцерія куб»

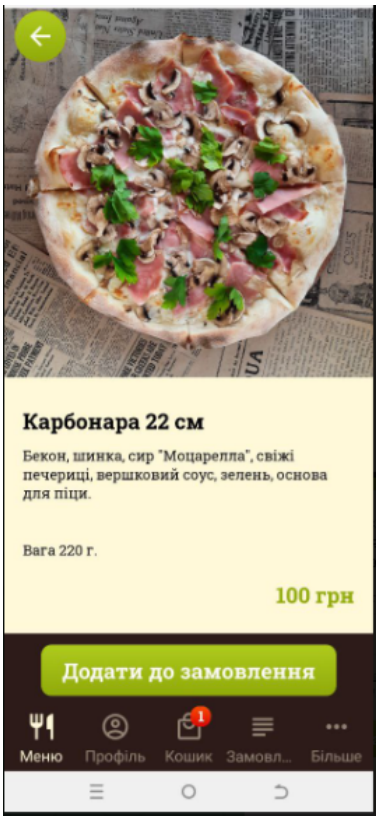

Рисунок 1.5 – Карточка товару у додатку «Corleone»

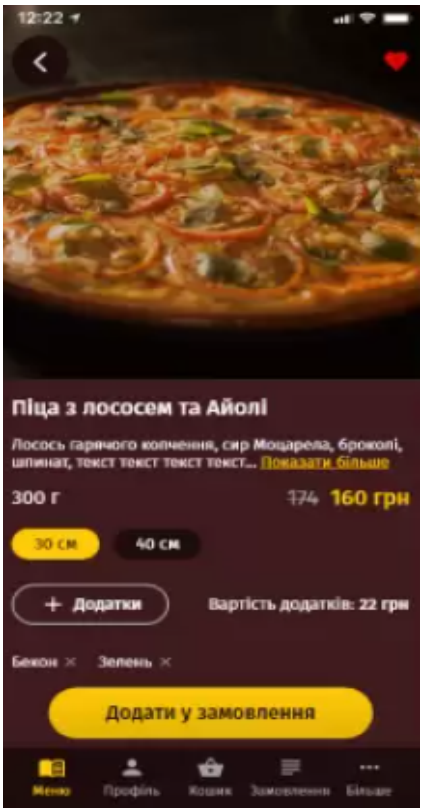

Рисунок 1.6 – Карточка товару у додатку «KUCHENCHEF»

Також обов’язковою частиною є сторінка-кошик, в ній відображені всі продукти які користувач хочу замовити, можливість переходу до оформлення замовлення, логіка для збільшення кількості товару та функції очистку всього кошика або видалення одного продукту (рис. 1.7, рис. 1.8, рис. 1.9).

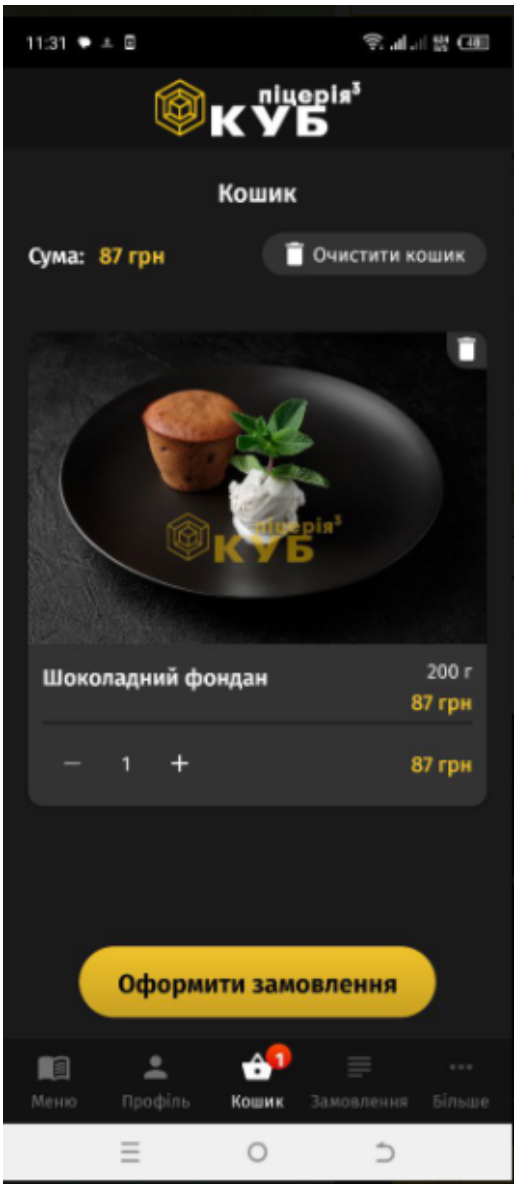

Рисунок 1.7 – Сторінка кошику у ресторані «Піцерія куб»

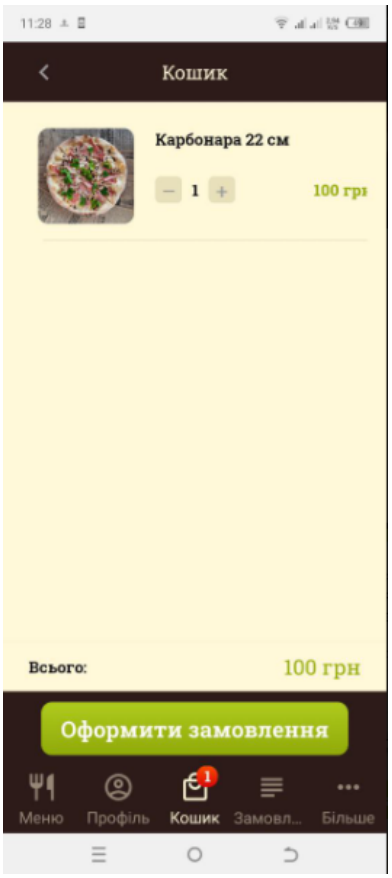

Рисунок 1.8 – Сторінка кошику у ресторані «Corleone»

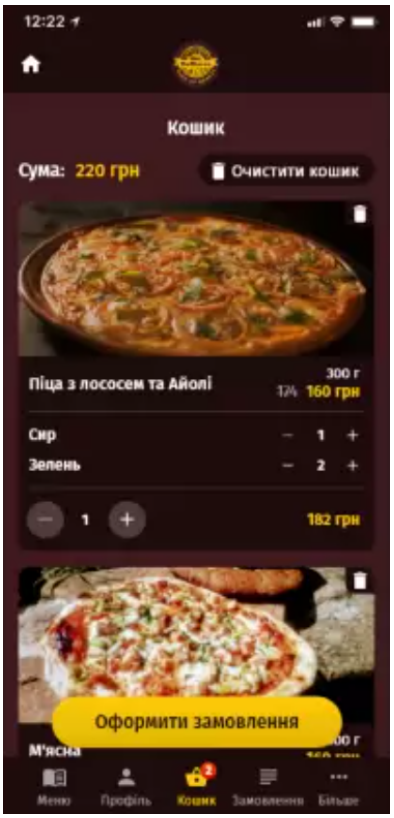

Рисунок 1.9 – Сторінка кошику у ресторані «KUCHENCHEF»

Найважливішою частиною додатку ресторану є сторінка з оформленням замовлення. На цій стоянці користувач повинен заповнити свої дані для доставки, або якщо самовивіз то можно додати коментар та вибрати спосіб оплати, для ресторану «KUCHENCHEF» можно обрати час приготування, коли замовник може прийти і забрати вже готове замовлення (рис 1.10, рис 1.11, рис 1.12).

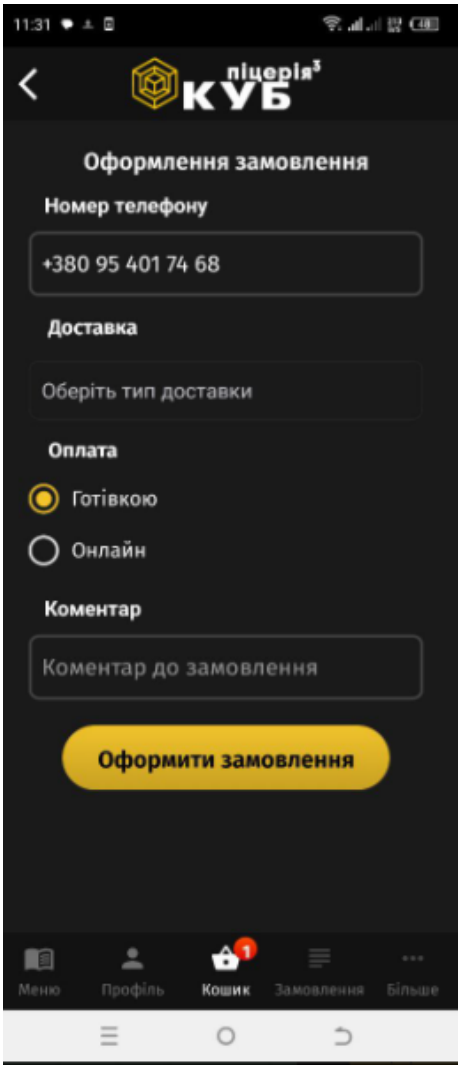

Рисунок 1.10 – Сторінка оформлення замовлення у ресторані «Піцерія куб»

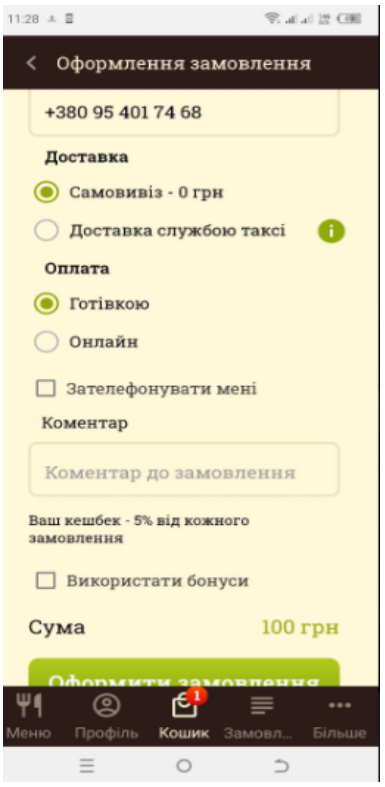

Рисунок 1.11 – Сторінка оформлення замовлення у ресторані «Corleone»

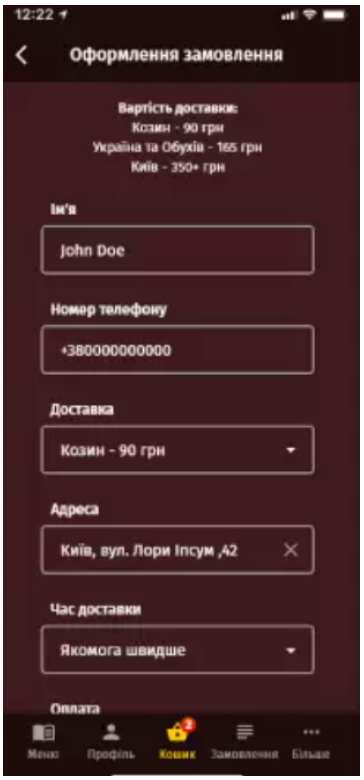

Рисунок 1.12 – Сторінка оформлення замовлення у ресторані «KUCHENCHEF»

Мобільні додатки ресторанів, які було проаналізовано, містять в собі витончену сторінку історії замовлень. Вона пропонує користувачам зручний інструмент для перегляду їхніх попередніх замовлень, організованих за хронологією від найновіших до найстаріших. Ця функція забезпечує короткий опис кожного замовлення, що дозволяє користувачам легко відстежувати свою історію замовлень блюд, які вони пробували, статуси своїх замовлень та з великою ймовірністю замовлять у майбутньому. Сторінка історії замовлень стає своєрідним журналом кулінарних уподобань користувача, відображаючи не тільки їх вибір страв, але й динаміку смакових уподобань часу (рис 1.13, рис 1.14, рис 1.15).

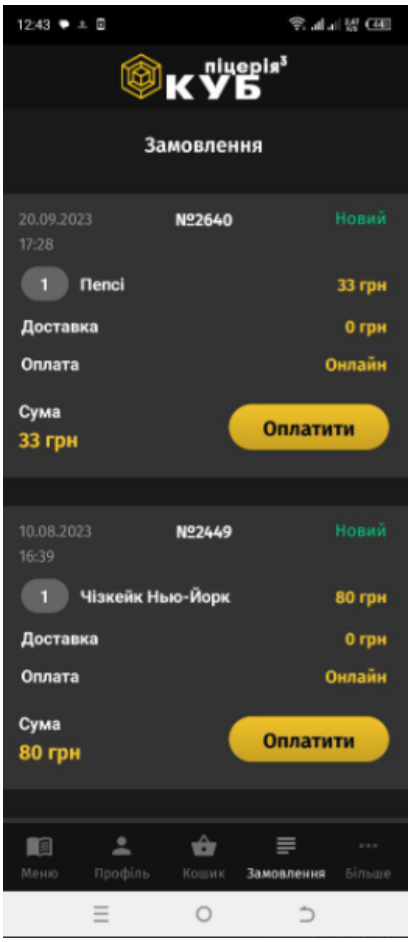

Рисунок 1.13 – Сторінка перегляду історії замовлень у ресторані «Піцерія куб»

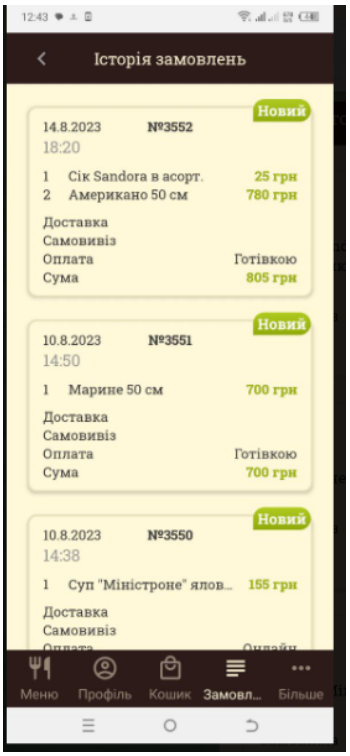

Рисунок 1.14 – Сторінка перегляду історії замовлень у ресторані «Corleone»

Рисунок 1.15 – Сторінка перегляду історії замовлень у ресторані «KUCHENCHEF»

Сторінка контактів у мобільному додатку ресторану пропонує користувачам всю необхідну інформацію для зв'язку. Вона містить докладні відомості, такі як фізична адреса закладу, що допомагає відвідувачам легко знайти місце для наступної кулінарної зустрічі. Електронна пошта ресторану доступна для зручного вирішення будь-яких запитань чи особливих побажань. Сторінка також інтегрована з соціальними мережами, надаючи можливість слідкувати за актуальними новинами та акціями, а номер телефону гарантує безпосередній зв'язок з обслуговуючим персоналом для миттєвого бронювання столу чи уточнення деталей замовлення.

Крім того, деякі ресторани включають на цій сторінці посилання на Умови користувацької угоди, що надає користувачам повну інформацію про правила та регуляції використання послуг додатку, забезпечуючи прозорість та впевненість у використанні сервісу (рис 1.16, рис 1.17, рис 1.18).

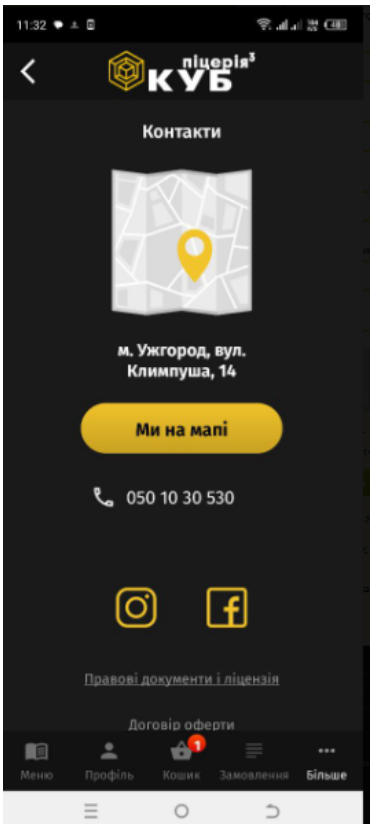

Рисунок 1.16 – Сторінка контактів у ресторані «Піцерія куб»

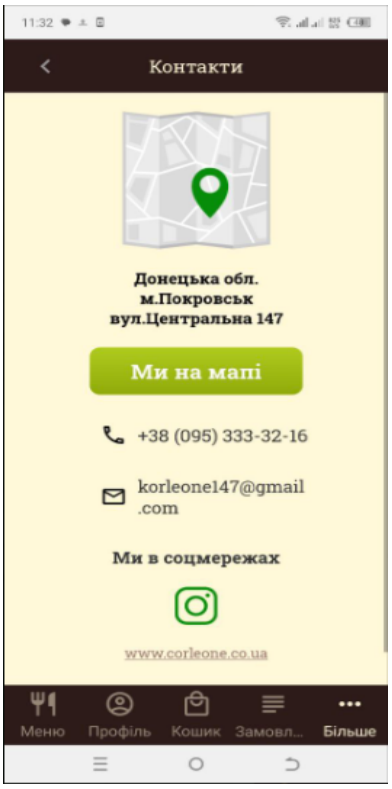

Рисунок 1.17 – Сторінка контактів у ресторані «Corleone»

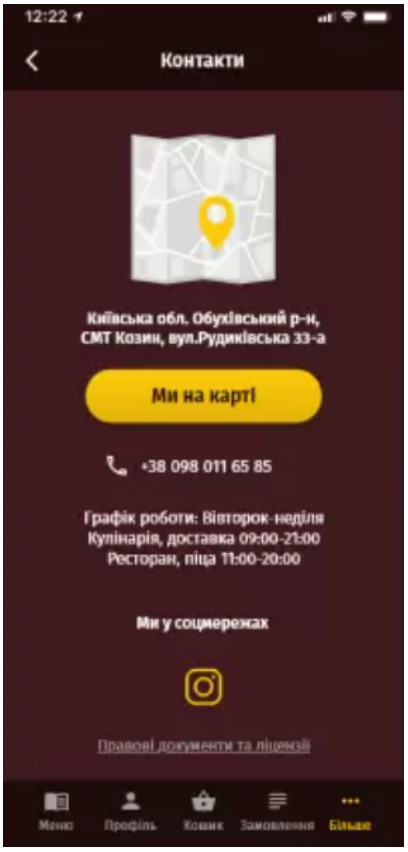

Рисунок 1.18 – Сторінка контактів у ресторані «KUCHENCHEF»

У ході аналізу аналогів було виявлено, що у ресторані «Піцерія куб» є можливість бронювання столиків. Сторінка бронювання столика в мобільному додатку ресторану створена для забезпечення максимальної зручності планування візиту. Вона має зрозумілий інтерфейс, де користувачі можуть ввести свій номер телефону, забезпечуючи ресторану надійний канал зв'язку для підтвердження бронювання. Є поля для вибору кількості гостей, що дає можливість ресторану ефективно готуватися до прийому відповідної кількості відвідувачів.

Крім того, на сторінці присутній зручний календар та годинник, які дозволяють вибрати бажану дату та час візиту. Це допомагає уникнути непорозумінь та забезпечити, що стіл буде готовий саме тоді, коли клієнт прибуде. Після введення всієї необхідної інформації, користувач просто натискає кнопку для надсилання запиту на бронювання, після чого може очікувати на дзвінок або текстове повідомлення з підтвердженням від ресторану. Ця функція забезпечує ефективний і гладкий процес бронювання, знижуючи потенційний стрес при плануванні спеціальних подій чи зустрічей.

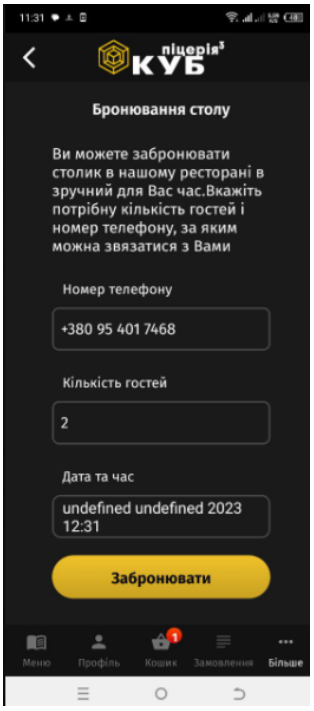

Рисунок 1.19 – Сторінка бронювання столиків у ресторані «Піцерія куб»

Порівняльний аналіз додатків в таблиці 1.1

Таблиця 1.1 – Порівняльний аналіз додатків

Критерії	«Піцерія куб»	«Corleone»	«KUCHENCHE»
Приваблива головна сторінка	+	+	+
Список категорій	+	+	+
Список товарів до кожної категорії	+	+	+
Зручне навігаційне меню	+	+	+
Сторінка кошику	+	+	+
Сторінка оформлення замовлення	+	+	+
Можливість обрати час доставки	-	-	+
Сторінка контактів	+	+	+
Історія замовлень з сортування від нових до старих	+	+	+
Сторінка для бронювання столиків	+	-	-

Аналізуючи розглянуті аспекти мобільного додатку для ресторану, можна сформулювати наступні основні розділи та функції, які є обов'язковими для забезпечення високого рівня користувацького досвіду:

- Головна сторінка: Це вітальна сторінка, яка забезпечує швидкий доступ до всіх функцій додатку, до меню, списку товарів та навігації.

- Меню ресторану: Блок з детальним переліком страв, поділених на категорії, яка дає можливість користувачам легко переглядати асортимент і вибирати блюда.
- Карточка товару: Важлива для демонстрації детальної інформації про кожну страву, включаючи фото, назву, вагу, ціну, та кнопку для додавання у кошик чи замовлення.
- Історія замовлень: Функція, що дозволяє користувачам переглядати свої попередні замовлення, впорядковані від найновіших до старих, що сприяє лояльності та зручності повторних замовлень.
- Сторінка контактів: Тут користувачі можуть знайти всю необхідну контактну інформацію, включаючи адресу, електронну пошту, соціальні мережі, номер телефону та посилання на користувацьку угоду.
- Бронювання: Розділ, який дозволяє відвідувачам забронювати столик, вказавши свій номер телефону, кількість гостей та бажану дату та час візиту.

Кожен з цих розділів забезпечує зручність та ефективність взаємодії з рестораном, створюючи цілісний та інтуїтивно зрозумілий користувацький інтерфейс, що є ключовим для приваблення та утримання клієнтів в сучасному цифровому просторі.

1.3 Визначення середовища розробки

Для створення сучасного і функціонального мобільного додатку ресторану використовується гнучкий підхід до розробки, який включає в себе застосування передових технологій. React Native – це інноваційний фреймворк, розроблений Facebook, що дозволяє розробникам створювати нативні мобільні додатки для iOS та Android за допомогою JavaScript та React.] Ця технологія забезпечує високу продуктивність

при одночасному спрощенні процесу розробки, дозволяючи писати код один раз і запускати його на обох платформах, забезпечуючи при цьому гладку і нативну роботу додатка. Це рішення ідеально підходить для ресторанів, що прагнуть забезпечити високу якість взаємодії з клієнтом через мобільні пристрої, надаючи їм можливість легко переглядати меню, робити замовлення та бронювати столики.[7]

Node.js виступає в ролі серверної платформи для обробки зовнішніх запитів, управління базами даних і логікою додатку. Це середовище виконання для JavaScript, яке дозволяє використовувати JS для серверного програмування, пропонуючи при цьому масштабованість і асинхронність в обробці запитів. Завдяки цьому, веб-сервер може обробляти велику кількість запитів одночасно, не блокуючи потік виконання. Завдяки своїй асинхронній природі, Node.js забезпечує ефективну обробку великої кількості запитів, що є надзвичайно важливим для ресторанів з високим потоком онлайн-замовлень.[4] Ця технологія відкриває двері для інтеграції з різноманітними базами даних та зовнішніми сервісами, як-от системи CRM, платіжні шлюзи, а також системи лояльності і відгуків.[26]

Використання цих двох технологій разом дозволяє створити мобільний додаток, який не тільки швидко працює та відповідає всім сучасним стандартам інтерфейсу і користувацького досвіду, але й є легко масштабованим та легким у підтримці. Реактивний інтерфейс з React Native забезпечить користувачам плавність переходів та інтерактивність елементів, тоді як Node.js на серверній стороні забезпечить стабільну роботу додатку, високу швидкість обробки даних та безпеку інформації. Комбінація React Native та Node.js створює міцну основу для розробки мобільних додатків, які можуть бути легко адаптовані до змінюваних потреб бізнесу та його клієнтів. Разом ці технології формують екосистему, що забезпечує не тільки швидкість

та ефективність роботи, але й надає розробникам інструменти для створення справді вражаючого користувацького інтерфейсу та взаємодії, яка буде відповідати найвищим стандартам індустрії гостинності.[5]

1.4 Постановка задачі. Технічне завдання

1.4.1 Призначення інформаційної системи

Мобільний додаток призначено для клієнтів ресторану з метою поліпшення сервісу для клієнтів, підвищення лояльності та автоматизації роботи ресторану по прийманню замовлень з мобільного додатку.

1.4.2 Мета створення інформаційної системи

Автоматизувати процеси взаємодії клієнтів з рестораном через веб-сайт, забезпечивши зручний і ефективний сервіс для онлайн-замовлень та бронювання.

1.4.3 Цільова аудиторія

Клієнти, які бажають швидко отримати інформацію про ресторан, його пропозиції, замовити страви онлайн або забронювати стіл.

2. Вимоги до інформаційної системи

2.1 Вимоги до інформаційної системи в цілому

2.1.1 Вимоги до структури

Додаток має бути інтуїтивно зрозумілим і забезпечувати легкий навігаційний досвід, з чіткою структуризацією розділів і функцій для замовлення страв і бронювання столиків.

2.1.2 Вимоги до персоналу

Персонал, який займається підтримкою веб-сайту, повинен мати базові технічні навички управління веб-контентом без необхідності глибоких знань в області ІТ.

2.1.3 Вимоги до збереження інформації

Інформація про меню, замовлення та бронювання повинна зберігатися в базі даних із забезпеченням конфіденційності та безпеки даних клієнтів.

2.1.4 Вимоги до розмежування доступу

Неавторизовані юзери не мають доступу до оформлення замовлення та додавання в обране, але мають змогу подивитися всі розділи додатку, додавати в кошик та бронювати столик.

2.2 Вимоги до функцій, виконуваних додатком

Додаток повинен забезпечувати можливість здійснювати замовлення користувачем. Користувач авторизується в системі, після чого, заповнює форму своїх особистих даних. Користувач може ознайомитися з каталогом продукції даного ресторану, зробити пошук і фільтрацію по або ж вибрати відповідну категорію. Користувачеві буде доступна можливість додати товар до обраного, а також робити замовлення на кілька товарів (страв). Далі користувач може додати товар в корзину, задати кількість товару для замовлення та оформити замовлення обравши онлайн спосіб оплати або готівко, а також опцію доставки або самовивозу. При процесі замовлення відстежує статус замовлення, може переглянути поточні активні акції. Управління додатком виконується через спеціальну адмін-панель.

Додаток має носити насамперед функціональний, а також інформаційний характер для кінцевого користувача. Функціональне та інформаційний напрямок цього додатка має розкриватися у вигляді наступних можливостей:

- Багатомовність (українська, англійська)
- Відображення категорій і товарів до кожної категорії
- Функціонал оформлення замовлень
- Особистий кабінет клієнта, історія замовлень
- Каталог з можливістю вибору по категорії, перегляд карток товарів, додавання у кошик
- Список переглянутих та акційних товарів (та банерів), на головному екрані

- Можливість додати товари в обране
- Онлайн оплата замовлення (LiqPay)
- OTP код для авторизації / реєстрації
- Push повідомлення (про зміну статусу замовлення, а також нових акційних страв)

2.3 Вимоги до видів забезпечення

2.3.1 Вимоги до інформаційного забезпечення

Технічна клієнтська частина: JS, CSS, HTML5, TypeScript, React Native.

Технічна серверна частина: Node.js, MongoDB

2.3.2 Вимоги до лінгвістичного забезпечення

Додаток має бути виконаний українською мовою та англійською.

2.3.3 Вимоги до програмного забезпечення

Мобільний додаток повинен коректно працювати на Android платформах не нижче 9.0 і IOS не нижче 12.0 версій;

Дозволи екранів (Android): mdpi (320x480 px), hdpi (480x800px), xhdpi (720x1280px), xhdpi (768x1280px), xxhdpi (1080x1920px);

Дозволи екранів (IOS): Retina HD 1080x1920, Retina 750x1334, Retina 640x1136 (iphone 5, 5s, SE);

2.3.4 Вимоги до апаратного забезпечення

Не менше 100 МБ вільного місця на диску.

Обов'язкове інтернет-з'єднання (WI-FI, 3G, LTE, EDGE).

2. МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

Кожен мобільний додаток для ресторану містить добре продуману структуру, яка сприяє легкій та інтуїтивно зрозумілий навігації для користувачів. Це означає, що користувачі з легкістю знаходять потрібні функції, такі як перегляд меню, замовлення, перевірка акцій та спеціальних пропозицій, а також бронювання столиків. Клієнти можуть безперешкодно переходити від одного розділу до іншого, зберігаючи відчуття контролю та зручності під час користування додатком.[20]

Успішний мобільний додаток для ресторану вимагає збалансованого підходу до розробки його логічної та фізичної структури. Логічна структура відноситься до візуальної організації додатку, що включає дизайн інтерфейсу, розміщення меню та кнопок навігації, а також послідовність екранів. Ця структура забезпечує зрозумілість та легкість використання, дозволяючи користувачам відчувати себе впевнено під час взаємодії з додатком.

Фізичний рівень структури, з іншого боку, пов'язаний з технічною організацією додатку, включаючи розташування файлів коду, баз даних та інших ресурсів у проекті. Цей аспект розробки зосереджений на ефективності, безпеці та оптимізації роботи додатку, забезпечуючи стабільність і швидкість реакції на запити користувача.

Розробка мобільного додатку для ресторану вимагає ретельного планування як логічної, так і фізичної структури, щоб забезпечити зручність та функціональність для кінцевих користувачів, в той же час підтримуючи технічну ефективність та безпечність додатку.

2.1 Архітектура мобільного застосунку

Мобільна платформа застосунку повинна складатися з 3 основних частин (див. Рис. 2.1):

- Клієнтська програма для різних платформ (Android, IOS);

- Сервер (містить веб-сервіси, які обслуговують запити клієнтських додатків);
- База даних (Використовується для безпосереднього зберігання вмісту сервісів. Якщо необхідна інформація вже міститься в джерелах, до яких надається доступ, використовується API цих віддалених ресурсів, що забезпечує необхідну функціональність).

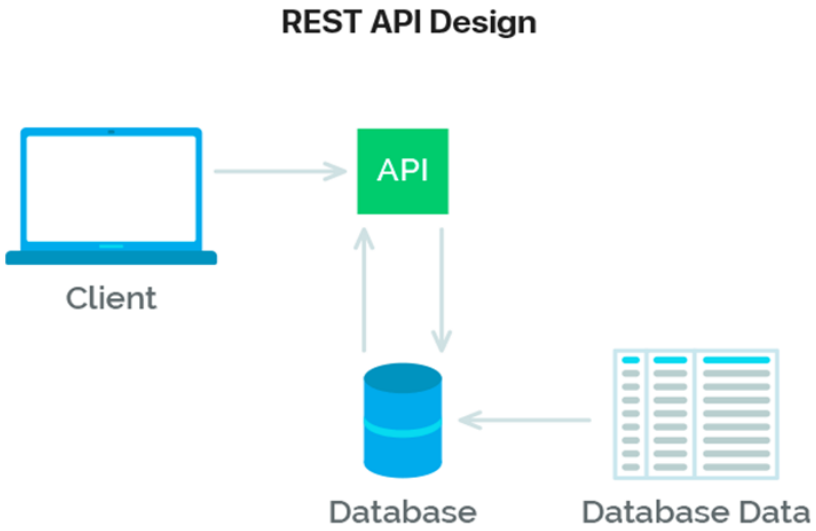

Рисунок 2.1 – Взаємодія мобільного API-додатку і сервера

Таким чином, мобільний додаток є клієнт-серверним додатком, клієнтська частина якого користувач встановлює безпосередньо на мобільному пристрої. В процесі роботи клієнтська програма зв'язується з серверною частиною програми через веб-сервіси. Технологія реалізації програм кросплатформовий JS framework - React Native Framework для розробки повноцінних версій (з інтерпретацією всього функціоналу) додатків для платформ Android і IOS.

2.2 Фізична структура

Коли користувач входить у додаток йому зазвичай завантажується головна сторінка, потім гість обирає на яку сторінку він хоче перейти та в

залежності від сторінки додаток підвантажує необхідні сторінки або блоки. Кожен файл міститься у папці яка зроблена для певної задачі.

Коренева директорія проекту включає папки для нативних платформ: android і ios, які містять специфічні для платформи ресурси та конфігураційні файли. Це дозволяє управляти платформо-залежним кодом і ресурсами в одному місці.

Папка src є основною папкою для джерельних файлів JavaScript та TypeScript, і структурована на декілька піддиректорій:

- @types містить оголошення типів TypeScript для статичної перевірки типів і автодоповнення у редакторі коду.
- assets включає всі статичні файли, такі як зображення, шрифти та інші медіа-ресурси, які використовуються у додатку.
- components містить повторно використовувані компоненти UI, такі як кнопки, карточки страв тощо.
- config зберігає конфігураційні файли, які можуть містити різні налаштування додатку.
- contexts використовується для управління станом додатку на основі React Context API.
- hooks зберігає користувацькі хуки React, які дозволяють використовувати стан і інші можливості React без написання класів.
- navigation відповідає за систему навігації в додатку, включаючи визначення маршрутів і переходів між екранами.
- screens містить всі екрани додатку, кожен з яких представляє певний інтерфейс і функціонал.
- store застосовується для управління глобальним станом додатку з використанням Redux або іншої бібліотеки для управління станом.
- templates може містити шаблони або макети для сторінок чи компонентів.
- types визначає загальні типи і інтерфейси, використовувані в додатку.

- utils включає в себе допоміжні функції та утиліти, які можуть використовуватися в різних частинах додатку.

Крім цього, на кореневому рівні знаходяться конфігураційні файли, такі як .buckconfig, .eslintrc.js (для лінтингу коду), .prettierrc.js (для форматування коду), та інші, які використовуються для налаштування середовища розробки та підтримки консистентності коду між різними розробниками.

Фізична структура проекту, є відображенням логічної структури додатку, що робить процес розробки і подальшої підтримки додатку зручним і ефективним. Фізична структура веб-сайту зображена на рисунку 2.2.

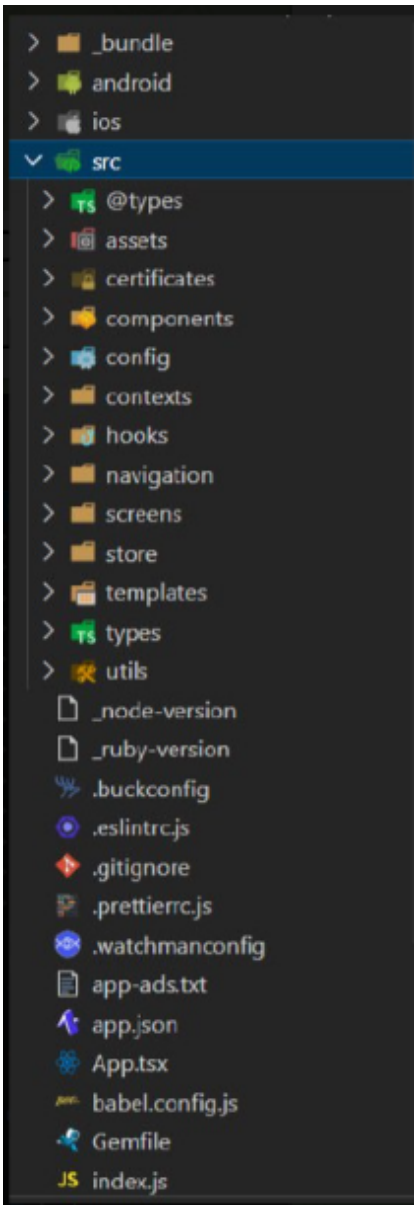

Рисунок 2.2 – Фізична структура веб-сайту

2.3 Реалізація бази даних

Вибір MongoDB для реалізації проекту додатка для ресторану обґрунтовується декількома ключовими перевагами:

- Гнучкість схеми: У ресторанному бізнесі дані часто міняються - нові страви, зміни в меню, сезонні акції. MongoDB дозволяє легко адаптувати і розширювати структуру даних без потреби в переробці всієї схеми бази даних.
- Швидкість і продуктивність: Документо-орієнтований підхід MongoDB забезпечує високу швидкість читання та запису даних, що є критично важливим для додатків, які вимагають швидкого відгуку, наприклад, для онлайн-замовлень або бронювання столиків.
- Підтримка складних запитів: MongoDB дозволяє виконувати складні запити та агрегації, що необхідно для аналізу даних, таких як звіти про продажі, популярні страви, поведінкові патерни клієнтів.
- Інтеграція з іншими технологіями: Легко інтегрується з популярними фреймворками та платформами, які часто використовуються для розробки додатків, наприклад, з Node.js, Express.js, що дозволяє створювати гнучкі та ефективні рішення.
- Зручність у роботі з JSON даними: Оскільки MongoDB працює з даними у форматі, схожому на JSON, це спрощує роботу розробників, особливо при використанні JavaScript або технологій, базованих на JS.
- Надійність та висока доступність: Система реплікації та шардування даних забезпечує високу доступність та надійність, що критично важливо для бізнес-додатків.

Клас Product ефективно поєднує у собі як структурні, і функціональні елементи, необхідні комплексного представлення продукту. Він включає різноманітні поля та асоціації, кожна з яких анотована декораторами @Prop. Ці декоратори сприяють точному визначенню типу даних, забезпечуючи при цьому гнучкість та масштабованість структури даних. Особлива увага у класі Product

приділена визначенню зв'язків з іншими сутностями, такими як категорії, що дозволяє ефективно організувати взаємодію між різними компонентами системи. Крім того, реалізація поля variations демонструє здатність MongoDB зберігати складні і гнучкі структури даних, що є ключовою перевагою при роботі з різноманітними атрибутами продуктів. Визначення основних характеристик продукту name, description, price. Встановлення відносин з іншими сутностями категорії, restaurant, additions (додаток А).

Клас "Categories", який є невід'ємною частиною системи управління контентом у ресторанному додатку, представляє собою детально пророблену модель для категорій страв чи продуктів. Цей клас ефективно моделює схему категорій, необхідну для організації та класифікації різноманітних пунктів меню або товарів у ресторані, що дозволяє користувачам легко навігувати та знаходити бажані страви або продукти. Ключовим елементом у цьому класі є унікальний ідентифікатор "_id" для кожної категорії, що гарантує унікальність кожної записаної категорії в базі даних. Це важливо не тільки для внутрішньої організації даних, але й для підтримки інтеграції з іншими компонентами системи, такими як модулі статистики та аналітики. Поле "title" визначає назву категорії та включає обмеження щодо мінімальної та максимальної довжини, що забезпечує консистентність та чіткість представлення інформації. Це особливо важливо для забезпечення якості користувацького інтерфейсу, де кожна категорія має бути представлена чітко та зрозуміло. Поле "photo" зберігає шлях до зображення кожної категорії, що вносить візуальну привабливість та допомагає користувачам швидше ідентифікувати бажану категорію. Візуалізація є ключовою у сучасних додатках, де зорове сприйняття відіграє вирішальну роль у виборі користувача. Поле "hidden" вказує, чи має бути категорія прихована від кінцевих користувачів, що надає гнучкість у управлінні відображенням категорій, особливо у випадках, коли певні страви або товари тимчасово недоступні чи виведені з обігу. Поле "position" відіграє важливу роль у визначенні порядку, у якому категорії будуть відображатися в додатку. Це дає

змогу ресторанам структурувати пропозицію своїх страв чи продуктів у найбільш привабливий та логічний спосіб, сприяючи кращому користувацькому досвіду (додаток Б).

Клас "Basket" виконує надзвичайно важливу функцію в системі управління електронною комерцією, служачи в якості центрального вузла для зберігання та обробки інформації про товари, вибрані користувачами. Серцем цього класу є унікальний ідентифікатор кошика "_id", який гарантує індивідуалізацію кожного кошика в системі. Це особливо важливо для забезпечення точності та ефективності у процесах покупок та обробки замовлень. Поле "customer" зберігає інформацію про покупця, встановлюючи зв'язок між кошиком та конкретним користувачем, що дозволяє персоналізувати досвід покупки та поліпшити якість обслуговування клієнтів. Важливою складовою класу є масив "items", що містить деталізовану інформацію про кожен продукт у кошику, включаючи ідентифікатори продуктів, їх кількість, варіації та інші специфікації. Це дозволяє створити динамічний та гнучкий механізм для управління товарними позиціями в кошику, адаптованим до змінюваних потреб та виборів покупців. Ефективне управління кошиками є ключовим для забезпечення гладкого та безперебійного процесу покупок, від вибору товарів до оформлення замовлення. Використання класу "Basket" у рамках технологічного стеку NestJS і MongoDB надає розробникам величезну гнучкість та ефективність при обробці даних кошиків. Це дозволяє реалізувати складні та вимогливі функціональні потреби електронної комерції, одночасно підтримуючи високу продуктивність та масштабованість системи. Інтеграція цього класу з іншими компонентами системи, такими як модулі управління продуктами та обробки замовлень, створює міцний фундамент для ефективного та зручного управління електронною комерцією в рамках ресторанного бізнесу або будь-якої іншої галузі, де цифрова торгівля відіграє ключову роль (додаток В).

Клас "Order" є фундаментальним компонентом системи управління замовленнями, побудований на принципах гнучкості та масштабованості, що

дозволяє йому легко адаптуватися до змінних бізнес-процесів. Наприклад, дизайн класу забезпечує плавну інтеграцію нових видів продуктів або модифікацію способів доставки, відповідаючи зростаючим та розвиваючим потребам ресторанного бізнесу. Структура класу відтворює динамічну взаємодію між клієнтом та рестораном, охоплюючи весь процес від вибору страв до доставки замовлення, що дозволяє ресторану точно відстежувати та виконувати замовлення клієнтів. В рамках класу "Order" особлива увага приділена деталям кожного замовлення, включаючи всю необхідну інформацію про обрані продукти, їх кількість, ціну, а також будь-які додаткові компоненти, такі як спеціальні інгредієнти чи доповнення. Такий підхід не тільки забезпечує високу точність у формуванні замовлення, але й оптимізує процес управління запасами. Ключові характеристики класу включають унікальний ідентифікатор замовлення "_id", який слугує для ідентифікації кожного замовлення в системі; поле "customer", що містить інформацію про користувача, який здійснив замовлення; масив "items", який детально описує кожен продукт у замовленні, включаючи ідентифікатор продукту, кількість, варіацію, ціну та додаткові компоненти. Крім того, важливу роль відіграють деталі доставки "delivery", які охоплюють адресу доставки, час доставки та тип доставки (кур'єром або самовивіз). Загальна сума замовлення обчислюється на основі "itemsPrice", "deliveryPrice" та "totalPrice". Клас також включає механізми для відстеження статусу замовлення та платежу ("orderStatus", "paymentStatus"), а також додаткову інформацію, таку як коментарі клієнта, необхідність зв'язку з клієнтом для підтвердження замовлення ("comment", "call") та унікальний ідентифікатор замовлення ("id"). Ця всебічна та деталізована структура класу "Order" робить його незамінним елементом для ефективного управління замовленнями в сучасних ресторанних системах, забезпечуючи високий рівень обслуговування клієнтів та оптимізацію бізнес-процесів.

Клас "Booking" є невід'ємним елементом системи управління бронюваннями в ресторанах, відіграючи ключову роль у процесах резервації

столиків. Ця модель є інструментом для точного відстеження кожного бронювання через унікальний ідентифікатор "_id", який використовує тип Types.ObjectId для забезпечення унікальності кожного запису в базі даних. Це дозволяє ресторану не тільки зберігати важливу інформацію про бронювання, але й ефективно управляти своїми ресурсами. Поле "guests" дозволяє ресторанам планувати заповненість та розподіляти робочий простір, враховуючи очікувану кількість відвідувачів, тоді як наявність контактного номера "phone" забезпечує зв'язок з клієнтами для підтвердження або уточнення деталей бронювання. Поле "date" є ключовим для управління графіком та запасами ресторану, дозволяючи рестораторам точно координувати графік роботи та обслуговування клієнтів. Інтеграція з даними конкретного ресторану через поле "restaurant" надає можливість ресторанам координувати роботу між різними локаціями або відділеннями, підвищуючи ефективність та продуктивність. Клас "Booking" може бути розширений для інтеграції з системами відносин з клієнтами (CRM) та аналітичними інструментами, що значно підвищує ефективність обслуговування клієнтів та оптимізує роботу ресторану. Така інтеграція відіграє важливу роль у цифровізації бізнес-процесів, сприяючи зростанню загальної продуктивності та підвищенню якості обслуговування в галузі гостинності. Використання такого комплексного підходу у класі "Booking" відображає стратегічне бачення розвитку ресторанного бізнесу в епоху цифрових технологій, забезпечуючи високий рівень задоволення потреб клієнтів і забезпечуючи гладке та ефективне функціонування ресторану.

Клас Notification, представляє собою ключовий компонент системи управління повідомленнями в додатках, побудованих на базі MongoDB. Ця модель є невід'ємною частиною інфраструктури сповіщень, забезпечуючи зберігання, обробку та відстеження повідомлень, які надсилаються користувачам. Основною особливістю класу Notification є використання унікального ідентифікатора _id, створеного за допомогою Types.ObjectId. Цей ідентифікатор гарантує унікальність кожного запису повідомлення в базі

даних, що є важливим для точного відстеження та управління повідомленнями. Поле userType визначає тип користувача, до якого належить повідомлення, дозволяючи таргетувати повідомлення для різних груп користувачів. Це може включати різні категорії, такі як "адміністратор", "звичайний користувач" або "гість", залежно від логіки системи. Поле date використовується для зберігання дати створення або запланованої відправки повідомлення, що дозволяє планувати комунікації та забезпечувати своєчасне сповіщення користувачів. Поле playerIds може містити ідентифікатори або токени пристроїв користувачів, до яких повинні бути надіслані повідомлення. Це поле забезпечує націленість повідомлень і дозволяє взаємодіяти безпосередньо з певними користувачами або пристроями. Поле userId відіграє важливу роль у персоналізації повідомлень, вказуючи на конкретного користувача, якому призначено повідомлення. Це забезпечує можливість створення високо персоналізованого контенту, підвищуючи ефективність комунікацій. notificationId є ще одним унікальним ідентифікатором, який може бути використаний для класифікації та ідентифікації конкретних видів повідомлень у рамках системи. Поле show використовується для визначення, чи має бути повідомлення відображене користувачу. Це дозволяє контролювати візуальну присутність повідомлень в інтерфейсі користувача. data є полем з даними у форматі об'єкта, яке може включати будь-яку додаткову інформацію, необхідну для повідомлення, таку як текст сповіщення, зображення або посилання. Автоматично генеровані поля createdAt та updatedAt використовуються для відстеження часу створення та останнього оновлення запису повідомлення.

Клас News, слугує фундаментальною частиною системи управління контентом, спеціалізованою на новинах у контексті ресторанного бізнесу. Цей клас відіграє ключову роль у розповсюдженні інформації про актуальні події, спеціальні пропозиції, оновлення меню та інші важливі аспекти діяльності ресторану. Основним елементом в структурі класу News є унікальний ідентифікатор _id, генерований за допомогою Types.ObjectId. Цей

ідентифікатор є критично важливим для забезпечення унікальності кожної новини в базі даних, дозволяючи точно ідентифікувати та відслідковувати кожен запис.

Поле title представляє заголовок новини, що є ключовим елементом для залучення уваги користувачів. Заголовок має велике значення для первинного враження, тому важливо, щоб він був лаконічним, але водночас інформативним та привабливим. Зв'язок з ресторанами в класі News реалізований через поле restaurants, що містить масив ідентифікаторів Types.ObjectId. Це дозволяє пов'язати кожну новину з одним чи декількома ресторанами, надаючи можливість цілеспрямовано розповсюджувати інформацію серед відповідних закладів. Тіло новини, представлене полем body, є основним вмістом повідомлення. Це поле має велике значення, оскільки в ньому міститься детальна інформація новини, включаючи всі необхідні факти, описи та інші важливі дані. Поле photo використовується для зберігання шляху до графічного зображення, пов'язаного з новиною. Візуальний контент відіграє значущу роль у сучасному медіапросторі, допомагаючи привертати увагу до тексту та підсилюючи його вплив.

3. ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ПРОДУКТУ

Для забезпечення високої якості та надійності програмного забезпечення в процесі тестування було обрано методологію "чорного ящика". Цей підхід дозволяє зосередитися на функціональності та поведінці програми, не вдаючись у внутрішню структуру чи реалізацію коду. Він є особливо ефективним у виявленні розбіжностей між очікуваною та фактичною поведінкою програми, а також у забезпеченні відповідності розробленого продукту зазначеним вимогам та специфікаціям.

Для того, щоб максимально ефективно виявити потенційні вади та забезпечити всебічне тестування, була розроблена детальна набір тест-кейсів. Кожен тест-кейс ретельно продуманий та спрямований на перевірку конкретних аспектів функціонування програми. Це дозволяє систематично охопити всі ключові функції та забезпечити глибоке та ефективне тестування. Тест-кейси охоплюють різні сценарії використання, від стандартних операцій до менш очевидних випадків, що допомагає виявити приховані вади та забезпечити стабільність та надійність програмного продукту.

Використання цього підходу в поєднанні з детально розробленими тест-кейсами гарантує глибоке та всебічне тестування програмного забезпечення, сприяє підвищенню його якості та довіри користувачів до продукту.

Тест-Кейс 1: Пошук за назвою

Мета: Перевірити функціонал поля пошуку на головній сторінці.

Кроки:

- Відкрити головну сторінку додатку.
- Знайти поле пошуку.
- Ввести назву страви або продукту.

Очікуваний результат: Показуються результати, що відповідають введеному запиту.

Тест-Кейс 2: Перевірка горизонтального слайдера категорій

Мета: Переконатися, що горизонтальний слайдер категорій працює коректно.

Кроки:

- Відкрити головну сторінку додатку.
- Перевірити візуальну присутність слайдера.
- Пролистати слайдер вліво та вправо.

Очікуваний результат: Слайдер прокручується, відображаючи різні категорії з фото та назвами.

Тест-Кейс 3: Взаємодія з категорією

Мета: Перевірити, що вибір категорії відображає відповідний список товарів.

Кроки:

- Відкрити головну сторінку додатку.
- Натиснути на будь-яку категорію у горизонтальному слайдері.

Очікуваний результат: Відкривається сторінка або розділ з переліком товарів, які належать до обраної категорії.

Тест-Кейс 4: Перевірка інформації про товар

Мета: Перевірити, чи коректно відображається інформація про товари (фото, ціна, скидка, назва, вага).

Кроки:

- Відкрити розділ з товаром після вибору категорії.
- Перевірити наявність та коректність інформації для кожного товару.

Очікуваний результат: Інформація про кожний товар відображається чітко та коректно, включаючи наявність знижки, якщо вона є.

Тест-Кейс 5: Вертикальний скрол товарів

Мета: Перевірити функціональність вертикального скролу в списку товарів.

Кроки:

- Відкрити розділ з товаром після вибору категорії.

- Прокрутити список товарів вниз та вгору.

Очікуваний результат: Список товарів легко скролиться, відображаючи всі товари, доступні в даній категорії.

Тест-Кейс 6: Перевірка Відображення Карточки Товара

Мета: Перевірити коректність відображення інформації в карточці товара.

Кроки:

- Відкрити карточку будь-якого товару.
- Перевірити наявність та коректність відображення фотографії, ціни, скидки (якщо є), ваги товару.

Очікуваний результат: Всі елементи відображаються чітко та коректно.

Тест-Кейс 7: Додавання Добавок до Товару

Мета: Перевірити функціональність додавання добавок до товару.

Кроки:

- Відкрити карточку товару.
- Натиснути на кнопку для додавання добавок.
- Перевірити відкриття списку доступних добавок з ціною, назвою та фото кожної.
- Вибрати добавку, натиснувши на кнопку "+" поруч з нею.

Очікуваний результат: Кнопка "+" змінюється на "-", обрана добавка відображається на карточці товару.

Тест-Кейс 8: Відображення Доданої Добавки на Карточці Товару

Мета: Перевірити відображення доданої добавки на карточці товару.

Кроки:

- Після додавання добавки до товару, перевірити її відображення на карточці товару.
- Перевірити наявність кнопки "-" поруч з добавкою.

Очікуваний результат: Добавка коректно відображається на карточці товару, поруч з нею з'являється кнопка "-", яка дозволяє видалити добавку.

Тест-Кейс 9: Видалення Добавки з Карточки Товару

Мета: Перевірити можливість видалення добавки з карточки товару.

Кроки:

- Натиснути на кнопку "-" поруч з добавкою на карточці товару.

Очікуваний результат: Добавка видаляється з карточки товару, кнопка "-" змінюється назад на "+" у списку доступних добавок.

Тест-Кейс 10: Додавання товару в корзину

Мета: Перевірити можливість додавання товару в корзину.

Кроки:

- Вибрати товар.
- Натиснути кнопку "Додати в корзину".

Очікуваний результат: Товар успішно доданий в корзину.

Тест-Кейс 11: Додавання товару з добавками в корзину

Мета: Перевірити можливість додавання товару з добавками як окремого товару в корзину.

Кроки:

- Вибрати товар.
- Додати до товару добавки.
- Додати товар з добавками в корзину.

Очікуваний результат: Товар з добавками доданий в корзину як окремий товар.

Тест-Кейс 12: Збільшення кількості товару в корзині

Мета: Перевірити функціонал збільшення кількості одного і того ж товару в корзині.

Кроки:

- Додати один і той же товар в корзину двічі.

Очікуваний результат: Кількість цього товару в корзині збільшується.

Тест-Кейс 13: Керування кількістю товару в корзині

Мета: Перевірити можливість зміни кількості товару в корзині.

Кроки:

- В корзині натиснути на кнопку "+" або "-" поруч з товаром для зміни його кількості.

Очікуваний результат: Кількість товару в корзині змінюється відповідно до дій користувача.

Тест-Кейс 14: Видалення добавки з товару в корзині

Мета: Перевірити можливість видалення добавки з товару, який знаходиться в корзині.

Кроки:

- Вибрати товар з добавкою в корзині.
- Видалити добавку з цього товару.

Очікуваний результат: Добавка успішно видалена з товару в корзині.

Тест-Кейс 15: Видалення товару з корзини

Мета: Перевірити можливість видалення товару з корзини.

Кроки:

- Натиснути на іконку видалення товару в корзині.

Очікуваний результат: Товар успішно видалений з корзини.

Тест-Кейс 16: Очищення корзини

Мета: Перевірити можливість повного очищення корзини.

Кроки:

- Натиснути на іконку очищення корзини в шапці.
- Підтвердити дію в модальному вікні.

Очікуваний результат: Всі товари успішно видалені з корзини.

Тест-Кейс 17: Перевірка Скролу Товарів в Корзині

Мета: Перевірити функціональність скролу товарів у корзині.

Кроки:

- Додати кілька товарів до корзини, достатньо для активації скролу.
- Відкрити корзину та спробувати прокрутити список товарів вгору та вниз.

Очікуваний результат: Список товарів у корзині має плавно скролитися, дозволяючи користувачу переглядати всі додані товари.

Тест-Кейс 18: Оформлення Замовлення з Корзини

Мета: Перевірити процес оформлення замовлення з корзини.

Кроки:

- Додати товари до корзини.
- Натиснути кнопку "Оформити замовлення".

Очікуваний результат: Відкривається форма оформлення замовлення.

Тест-Кейс 19: Перевірка Попередньо Заповнених Даних Користувача

Мета: Перевірити автоматичне заповнення імені та номера телефону користувача.

Кроки:

- Перейти до форми оформлення замовлення.
- Перевірити, чи правильно заповнені поля імені та телефону користувача.

Очікуваний результат: Ім'я та телефон користувача автоматично заповнені згідно з даними профілю.

Тест-Кейс 20: Вибір Способу Доставки

Мета: Перевірити можливість вибору способу доставки.

Кроки:

- В формі оформлення замовлення обрати спосіб доставки (самовивіз або доставка за адресою).
- У випадку вибору доставки за адресою, ввести адресу, під'їзд та номер квартири (якщо потрібно).

Очікуваний результат: Спосіб доставки вибрано, і поля для адреси доступні для заповнення при виборі доставки.

Тест-Кейс 21: Вибір Часу Доставки

Мета: Перевірити можливість вибору часу доставки.

Кроки:

- Вибрати час доставки ("конкретний час" або "якнайшвидше").

- При виборі конкретного часу переконатися, що вказана дата та час є майбутніми.

Очікуваний результат: Час доставки вибрано коректно, з врахуванням обмежень на минулі дати та часи.

Тест-Кейс 22: Використання Бонусів

Мета: Перевірити механізм використання бонусів при оформленні замовлення.

Кроки:

- Вказати кількість бонусів для використання в замовленні.
- Переконатися, що введена кількість бонусів не перевищує максимально дозволену адміністратором.

Очікуваний результат: Кількість бонусів прийнята або відхилена згідно з правилами.

Тест-Кейс 23: Вибір Типу Оплати і Перевірка Оплати Онлайн

Мета: Перевірити функціональність вибору типу оплати та інтеграцію з системою онлайн-оплати.

Кроки:

- Обрати тип оплати (готівкою або онлайн).
- У випадку вибору онлайн-оплати, переконатися в активації форми LiqPay або іншої платіжної системи.

Очікуваний результат: Тип оплати вибрано коректно, і при виборі онлайн-оплати відкривається форма для проведення транзакції.

Тест-кейс 24: Авторизація за номером телефону

Кроки:

- Відкрийте додаток.
- Натисніть на кнопку "Авторизація за номером телефону".
- Введіть непустий номер телефону.
- Натисніть на кнопку "Відправити код".
- Введіть правильний код OTP (одноразовий пароль).
- Натисніть на кнопку "Увійти".

Очікуваний результат: Після виконання кроку 6, користувач повинен успішно увійти в додаток.

Тест-кейс 25: Авторизація через обліковий запис Google

Кроки:

- Відкрийте додаток.
- Натисніть на кнопку "Авторизація через обліковий запис Google".
- Введіть дані облікового запису Google (електронну пошту та пароль).
- Натисніть на кнопку "Увійти".

Очікуваний результат: Після виконання кроку 4, користувач повинен успішно увійти в додаток за допомогою облікового запису Google.

Тест-кейс 26: Авторизація через обліковий запис Apple (для Android)

Кроки:

- Відкрийте додаток на пристрої з операційною системою Android.
- Переконайтеся, що на головному екрані додатку відсутня іконка "Авторизація через обліковий запис Apple" (так як це Android пристрій).
- Спробуйте виконати будь-який інший доступний спосіб авторизації.

Очікуваний результат: Додаток не повинен надавати опцію "Авторизація через обліковий запис Apple" на пристроях Android, і користувач повинен мати можливість використовувати тільки доступні способи авторизації.

Тест-кейс 27: Перевірка авторизованого користувача і перехід в профіль

Кроки:

- Відкрийте додаток.
- Увійдіть в додаток за допомогою одного з доступних способів авторизації (наприклад, через номер телефону або Google).
- Перевірте, що ви успішно авторизувалися.

- Натисніть на кнопку "Профіль" або іншу відповідну кнопку для переходу до профілю користувача.

Очікуваний результат: Після виконання кроку 4, користувач повинен бути перенаправлений до свого профілю в додатку.

Тест-кейс 28: Перехід до реєстрації при відсутності авторизації

Кроки:

- Відкрийте додаток.
- Перевірте, що на головному екрані відсутні іконки "Авторизація за номером телефону" та "Авторизація через обліковий запис Google".
- Спробуйте виконати будь-який з доступних способів авторизації (наприклад, "Авторизація через обліковий запис Apple").

Очікуваний результат: Оскільки авторизація не виконана, користувач повинен бути перенаправлений на сторінку реєстрації. На сторінці реєстрації повинні бути обов'язкові поля для введення імені та номера телефону.

Тест-кейс 29: Бронювання столика з обов'язковими полями

Кроки:

- Відкрийте сторінку бронювання столика в додатку.
- Введіть номер телефону, обов'язкове поле.
- Виберіть дату і час бронювання, обов'язкові поля.
- Натисніть на кнопку "Забронювати".

Очікуваний результат: При виконанні кроку 4, додаток повинен здійснити бронювання столика, якщо всі обов'язкові поля (номер телефону, дата і час) заповнені коректно.

Тест-кейс 30: Бронювання столика з неповними обов'язковими полями

Кроки:

- Відкрийте сторінку бронювання столика в додатку.
- Залиште номер телефону порожнім (неповним).
- Виберіть дату та час бронювання, обов'язкові поля.
- Натисніть на кнопку "Забронювати".

Очікуваний результат: При виконанні кроку 4, додаток повинен відобразити повідомлення про помилку, вказуючи на обов'язкове поле "Номер телефону" і не дозволяючи зробити бронювання.

Тест-кейс 31: Бронювання столика з невідповідною датою і часом

Кроки:

- Відкрийте сторінку бронювання столика в додатку.
- Введіть коректний номер телефону, обов'язкове поле.
- Виберіть дату і час, які вже минули або недоступні для бронювання (наприклад, минулу дату або минулий час).

Натисніть на кнопку "Забронювати".

Очікуваний результат: При виконанні кроку 4, додаток повинен відобразити повідомлення про помилку, вказуючи на недопустиму дату або час, і не дозволяючи зробити бронювання.

Тест-кейс 32: Бронювання столика з коректними даними

Кроки:

- Відкрийте сторінку бронювання столика в додатку.
- Введіть коректний номер телефону, обов'язкове поле.
- Виберіть майбутню дату і час бронювання.
- Натисніть на кнопку "Забронювати".

Очікуваний результат: При виконанні кроку 4, додаток повинен успішно здійснити бронювання столика на обрану дату та час.

ВИСНОВКИ

Під час виконання магістерської роботи було здійснено всебічне дослідження мобільних додатків для ресторанів. У результаті аналізу існуючих рішень було виявлено, що ефективний мобільний додаток має включати не тільки привабливий інтерфейс, але й забезпечувати детальну інформацію про меню ресторану, процес замовлення та можливості бронювання, а також мати легкість в навігації та доступність контактної інформації. Важливими компонентами є також функція онлайн-замовлень.

На підставі зібраних даних було сформульовано технічне завдання, яке чітко визначає вимоги до додатку та виокремлює ключові етапи його розробки.

Наступний крок планування дизайну користувацького інтерфейсу, вибір технологій для розробки додатку та ретельне планування програмування. Також розробка тестових сценаріїв, які допоможуть забезпечити високу якість фінального продукту і його відповідність всім критеріям, викладеним у технічному завданні.

На основі технічного завдання був розроблений додаток. Для реалізації front-end частини використовувався фреймворк React Native, а для back-end Node js.

Враховуючи ретельне технічне завдання та зібрані рекомендації, проект має всі шанси стати успішним мобільним додатком, який не тільки відповідатиме потребам відвідувачів ресторану, але й підсилить його ринкову присутність та забезпечить зростання лояльності клієнтів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Огляд фреймворку React Native для розробки мобільних додатків URL: https://reactnative.dev/docs/getting-started (Дата звернення 25.11.2023).

2. Node.js в розробці серверних додатків URL: https://nodejs.org/en/docs/ (Дата звернення 30.11.2023).

3. Переваги та особливості використання Node.js для бекенду URL: https://medium.com/the-node-js-collection (Дата звернення 01.12.2023).

4. Практичне застосування Node.js у розробці веб-сервісів URL: https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js (Дата звернення 02.12.2023).

5. Ресторанні додатки: як створити ефективний мобільний додаток URL: https://www.smashingmagazine.com/2020/08/react-native-mobile-app-restaurants/ (Дата звернення 05.12.2023).

6. Створення інтерактивних інтерфейсів у React Native URL: https://reactnative.dev/docs/interactive-props (Дата звернення 10.12.2023).

7. Менеджмент стану в додатках на React Native з використанням Redux URL: https://redux.js.org/introduction/getting-started (Дата звернення 12.11.2023).

8. Безпека мобільних додатків на Node.js URL: https://expressjs.com/en/advanced/best-practice-security.html (Дата звернення 15.11.2023).

9. Оптимізація продуктивності мобільних додатків React Native URL: https://www.simform.com/blog/react-native-performance/ (Дата звернення 17.11.2023).

10. Інтеграція бекенду на Node.js з фронтендом на React Native URL: https://blog.logrocket.com/connecting-react-native-app-node-js-server/ (Дата звернення 20.11.2023).

11. Розробка API для мобільних додатків з використанням Node.js URL: https://www.toptal.com/nodejs/secure-rest-api-in-nodejs (Дата звернення 22.11.2023).

12. "React Native: Створення кросплатформних ресторанних додатків" URL: https://reactnative.dev/docs/building-for-restaurants (Дата звернення 22.11.2023).

13. "Ефективність React Native для Android і iOS у ресторанному бізнесі" URL: https://www.android-ios-reactnative.com/effectiveness-in-restaurant-apps (Дата звернення 22.11.2023).

14. "Node.js як основа серверної частини для ресторанних додатків" URL: https://nodejs.org/en/docs/server-side-for-restaurants (Дата звернення 25.11.2023).

15. "Розробка мобільних додатків для ресторанів: Використання React Native та Node.js" URL: https://mobiledevjournal.com/react-native-nodejs-for-restaurant-apps (Дата звернення 20.11.2023).

16. "Інтеграція платіжних систем у ресторанних додатках на React Native" URL: https://reactnativepay.com/integration-in-restaurant-apps (Дата звернення 10.11.2023).

17. "Реалізація онлайн-бронювання в ресторанних додатках через React Native" URL: https://bookingsystem.reactnative.com/restaurant-apps (Дата звернення 15.11.2023).

18. "Node.js для управління даними в ресторанних мобільних додатках" URL: https://nodejsdatahandling.com/restaurant-apps (Дата звернення 18.01.2024).

19. "UX/UI дизайн для ресторанних додатків на React Native" URL: https://reactnativedesign.com/ux-ui-for-restaurant-apps (Дата звернення 02.11.2023).

20. "Використання GraphQL в React Native для оптимізації запитів у ресторанних додатках" URL: https://graphqlinreactnative.com/restaurant-apps (Дата звернення 23.01.2024).

21. "Системи лояльності в мобільних додатках для ресторанів: Підхід на базі React Native" URL: https://loyaltyprograms.reactnative.com/for-restaurants (Дата звернення 26.11.2023).

22. "Автоматизація замовлень у ресторанах через мобільні додатки на React Native" URL: https://orderautomation.reactnative.com/restaurant-apps (Дата звернення 29.11.2023).

23. "React Native та Node.js для розробки системи доставки їжі" URL: https://fooddeliverysystem.reactnative-nodejs.com (Дата звернення 01.11.2023).

24. "Мобільні додатки для ресторанів: Розробка на React Native з фокусом на користувацький досвід" URL: https://userexperience.reactnative.com/restaurant-apps (Дата звернення 05..11.2023).

25. "Node.js як рішення для бекенду ресторанних мобільних додатків" URL: https://nodejsbackend.com/for-restaurant-apps (Дата звернення 08..11.2023).

26. "Оптимізація продуктивності Android та iOS додатків для ресторанів на React Native" URL: https://performancetuning.reactnative.com/restaurant-apps (Дата звернення 10.11.2023).

Додаток А. Лістинг коду класу “Product”

```
import { Prop, raw, Schema, SchemaFactory } from '@nestjs/mongoose';
import { randomUUID } from 'crypto';
import { Types, HydratedDocument, SchemaTypes } from 'mongoose';
import { Addition } from './Addition';
import { Restaurant } from './Restaurant';
export type ProductDocument = HydratedDocument<Product>;
@Schema({ collection: 'Products', timestamps: true })
export class Product {
  _id: Types.ObjectId;
  @Prop({
    type: SchemaTypes.ObjectId,
    ref: Product.name,
    required: false,
    default: null, })
  parent: Types.ObjectId;
  @Prop({ type: Number })
  order: number;
  @Prop({ type: Boolean, default: false })
  isVariated: boolean;
  @Prop({
    type: SchemaTypes.ObjectId,
    ref: 'Categories',
    required: true,
    index: true, })
  category: Types.ObjectId;
  @Prop({
    type: SchemaTypes.ObjectId,
    ref: 'Categories',
    required: false,
```

```
  index: true,})
  subcategory: Types.ObjectId;
  @Prop({ type: [SchemaTypes.ObjectId], ref: Restaurant.name, required: true })
  restaurant: Types.ObjectId[];

  @Prop({ type: String, minlength: 2 })
  name: string;
  @Prop({
    type: String,    required: false,   default: 'На жаль тимчасово немає опису цього
продукту',  })
  description: string;
  @Prop({ type: Number, required: true, min: 0 })
  price: number;
  @Prop({ type: Boolean, required: true, default: false, index: true })
  hidden: boolean;
  @Prop({ type: String, required: false })
  photo: string;
  @Prop({ type: [SchemaTypes.ObjectId], ref: Addition.name, required: false })
  additions: Types.ObjectId[];
  @Prop({ type: [SchemaTypes.ObjectId], ref: Addition.name, required: false })
  deletedAdditions: Types.ObjectId[];
  @Prop(   raw({    required: false,   type: [ {
        _id: Types.ObjectId,
        name: {
          type: String,
          required: false, },
        priceChange: {
          type: Number,
          required: true,  },
        stockPriceChange: {
```

```
        type: Number,
        min: 0,},
      stockRestaurantChange: {
        type: [{ type: Types.ObjectId, ref: Restaurant.name }],},
      weightChange: { type: String,required: false,},
      guid: {  type: String,required: false,  default: randomUUID,},
      productGroupId: {   type: String, required: false,   default: randomUUID,},
      position: {
        type: Number,
        required: false, },
      restaurantDiscount: {
        type: [   { restaurant: { type: Types.ObjectId, ref: 'Restaurant' }, price:
Number,}, ],  },},     ],    }),  )
  variations: {
    _id: Types.ObjectId;
    name?: string;
   priceChange: number;
    stockPriceChange: number;
    stockRestaurantChange: Types.ObjectId[];
    restaurantDiscount?: { price: number; restaurant: Types.ObjectId }[];
    weightChange?: string;
    position?: number;
    guid?: string;
    productGroupId?: string;  }[];
  @Prop({ type: String, required: true })
  weight: string;
  @Prop({ type: Boolean, default: false })
  stock: boolean;
  @Prop({ type: Number, min: 0 })
  stockPrice: number;
```

```
  @Prop({ type: [SchemaTypes.ObjectId], ref: Restaurant.name })
  stockRestaurant: Types.ObjectId[];
  @Prop({ type: [SchemaTypes.ObjectId], ref: Product.name })
  similar: Types.ObjectId[];
  @Prop({ type: Boolean, default: false })
  novelty?: boolean;
  @Prop({ type: Boolean, default: false })
  promotionalOffer?: boolean;
  @Prop({ type: String, required: false, default: randomUUID })
  guid?: string;
  @Prop(
   raw({
    quantity: {  type: Number,   default: 0,required: true,      },
    avg: { type: Number,    min: 0, max: 5,default: 0,
     required: true,      },    }),  )
  rate: { quantity: number; avg: number;  };
  @Prop({
   type: [{ price: Number, restaurant: SchemaTypes.ObjectId }],
   default: [], })
  restaurantDiscount?: { price: number; restaurant: Types.ObjectId }[];}
export const ProductSchema = SchemaFactory.createForClass(Product);
ProductSchema.post('validate', async function () {
  await this.populate('category');});
```

Додаток Б. Лістинг коду класу “Categories”

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import {
  Schema as MongooseSchema,
  HydratedDocument,
  Types,
  model,
  Query,} from 'mongoose';
import { Product, ProductSchema } from './Product';
import { Restaurant } from './Restaurant';
export type CategoriesDocument = HydratedDocument<Categories>;
@Schema({ collection: Categories.name, timestamps: true })
export class Categories {
  _id: Types.ObjectId;
  @Prop({ required: false, default: null, ref: Categories.name, index: true })
  parent_id: string;
  @Prop({ minlength: 2, maxlength: 64, required: true })
  title: string;
  @Prop()
  photo: string;
  @Prop({ required: true, default: false, index: true })
  hidden: boolean;
  @Prop()
  position: number;}
export const CategoriesSchema = SchemaFactory.createForClass(Categories);
```

Додаток В. Лістинг коду класу “Basket”

```
import { Prop, raw, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Schema as MongooseSchema, HydratedDocument, Types } from
'mongoose';
import { Addition, AdditionDocument } from './Addition';
import { Product, ProductDocument } from './Product';
export type BasketDocument = HydratedDocument<Basket>;
export type PopulatedBasketDocument = HydratedDocument<
  Omit<Basket, 'items'> & {
    items: {product: ProductDocument;count: number;variation: string; additions:
Array<{addition: AdditionDocument; count: number;}>;}[];}>;
@Schema({ collection: Basket.name, timestamps: true })
export class Basket {
  _id: Types.ObjectId;
  @Prop({ required: true })
  customer: string;
  @Prop(
    raw({type: [ {
          product: { type: Types.ObjectId, ref: Product.name },
          count: Number,
          variation: { type: String },
          _id: false,
          additions: [{
              addition: { type: Types.ObjectId, ref: Addition.name },
              count: Number, },   ],},],
      required: true,}),)
  items: {
    product: Types.ObjectId;count: number;variation: string; additions:
Array<{addition: Types.ObjectId; count: number;}>;}[];}
export const BasketSchema = SchemaFactory.createForClass(Basket);
```

Додаток Г. Лістинг коду для сторінки кошика

```
import React, {useCallback, useContext, useEffect} from 'react';
import * as C from '../../components';
import {Container} from '../../components';
import {TOrderStackEnum, TOrderStackNav} from '../../navigation/config';
import {Dimensions, FlatList, StyleSheet, View} from 'react-native';
import {COLORS} from '../../utils';
import {useIsFocused, useNavigation} from '@react-navigation/native';
import {StackNavigationProp} from '@react-navigation/stack';
import {useTranslation} from 'react-i18next';
import {useTypedSelector} from '@App/store/store';
import {
  changeOrderingStatusAction,
  clearPaymentAction,
  postShoppingCardActions,
} from '@App/store/shopCard';
import {useDispatch} from 'react-redux';
import {LargeButton} from '@App/components/LargeButton';
import ProductCard from '@App/components/ProductCard/ProductCard';
import {EmptyCartMessage} from './components';
import {LoaderContext} from '@App/contexts';
import {SearchVariation} from './helpers';
import {changeErrorProfile} from '@App/store/auth';
const RealtimeOrderScreen: React.FC = () => {
  const navigation = useNavigation<StackNavigationProp<TOrderStackNav>>();
  const {t} = useTranslation();
  const isFocus = useIsFocused();
  const token = useTypedSelector(s => s.auth.token);
  const rest = useTypedSelector(s => s.rest.rest);
```

```
const {actions: loaderAction} = useContext(LoaderContext);
const {createOrderStatus, localCard, loader} = useTypedSelector(
  s => s.shopCard,
);
const dispatch = useDispatch();
const sumPriseOrder = localCard
  ? localCard
      .map(
        item =>
          ((item
            ? item.product
              ? SearchVariation(item.product.variations, item.variation)
                  ?.priceChange
              ? item.product.stock
                ? item.product.restaurantDiscount
                    ?.map(el => el.restaurant)
                    ?.includes(rest?._id!)
                  ? SearchVariation(
                      item.product.variations,
                      item.variation,
                    )?.restaurantDiscount?.find(
                      el => el.restaurant === rest?._id,
                    )?.price ?? 0
                  : SearchVariation(item.product.variations, item.variation)
                      ?.priceChange ?? 0
                : SearchVariation(item.product.variations, item.variation)
                    ?.priceChange ?? 0
              : item.product.restaurantDiscount?.find(
                  el => el.restaurant === rest?._id,
                )
```

```
              ? item.product.restaurantDiscount?.find(
                  el => el.restaurant === rest?._id,
                )?.price ?? 0
              : item.product.price
            : 0
          : 0) +
          (item.additions.length != 0
            ? item.additions
                .map(el => el.addition.price * el.count)
                .reduce((sum, cur) => sum + cur, 0)
            : 0)) *
        item.count,
    )
    .reduce((sum, current) => sum + current, 0)
  : 0;

useEffect(() => {
  if (loader) {
    loaderAction.SHOW();
  } else {
    loaderAction.HIDE();
  }
}, [loader]);

useEffect(() => {
  dispatch(clearPaymentAction());
  dispatch(
    changeOrderingStatusAction({
      id: null,
      numOrder: null,
```

```
      success: false,
    }),
  );
  dispatch(changeErrorProfile(false));
}, [isFocus]);
const onPressOrdering = () => {
  if (token) {
    if (localCard.length && !createOrderStatus.success) {
      dispatch(
        postShoppingCardActions.request({
          items: localCard.map(el => ({
            product: el.product._id,
            count: el.count,
            variation: el.variation,
            additions: el.additions.map(el => ({
              addition: el.addition._id,
              count: el.count,
            })),
          })),
        }),
      );
      navigation.navigate(TOrderStackEnum.OrderingScreen);
    }
  } else {
    navigation.navigate(TOrderStackEnum.Authorization, {type: 'NEW_CART'});
  }
};

const renderFlatList = useCallback(() => {
  return (
```

```
<FlatList
  data={localCard}
  style={{flex: 1, paddingHorizontal: 16}}
  contentContainerStyle={!localCard.length ? {flex: 1} : {}}
  showsVerticalScrollIndicator={false}
  bounces={false}
  keyExtractor={(el, index) => el.id + index}
  renderItem={({item, index}) => (
    <>
      {index == 0 && <C.Divider height={30} />}

      <ProductCard
        mode={'card'}
        additions={item.additions}
        variation={item.variation}
        count={item.count}
        product={item.product}
      />
      <C.Divider height={30} />

      {index + 1 == localCard.length && (
        <C.Divider height={Dimensions.get('screen').height * 0.17} />
      )}
    </>
  )}
  ListEmptyComponent={() => (
    <View
      style={{
        flex: 1,
        alignItems: 'center',
```

```
        }}>
        <EmptyCartMessage message={t('errors.empty.shoppingCard')} />
      </View>
    )}
   />
  );
}, [localCard, localCard.length]);

return (
  <Container
   defaultView
   style={{
    flex: 1,
    backgroundColor: COLORS.main,
    paddingHorizontal: 16,
   }}>
   {renderFlatList()}

   {!!localCard.length && (
    <View style={styles.absoluteButton}>
     <LargeButton
      text={
       t('order.orderingButton') +
       ` (${sumPriseOrder} ${t('components.currency')})`}
      buttonStyle={{
       alignSelf: 'center',
       width: '100%',
      }}
      onPress={onPressOrdering}
     />
```

```
        </View> )}
    </Container>  );};
const styles = StyleSheet.create({
  absoluteButton: {
    position: 'absolute',
    width: '100%',
    paddingHorizontal: 16,
    bottom: Dimensions.get('screen').height * 0.12,
    alignSelf: 'center',
  },});
export default RealtimeOrderScreen;
```