МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Оксана ШОВКОПЛЯС
(підпис)
грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»
освітньо-професійної програми «Інформатика»
на тему: Інформаційна технологія створення мобільного застосунку електронної комерції авторських виробів декоративно-ужиткового мистецтва
здобувача групи ІН.м-32 Рудик Олексія Олександровича

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

Олексій РУДИК
(підпис)

Керівник,
кандидат технічних наук,
доцент
Альона МОСКАЛЕНКО
(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Оксана ШОВКОПЛЯС
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 «Комп'ютерні науки», освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-32 Рудик Олексія Олександровича

1. Тема роботи: Інформаційна технологія створення мобільного застосунку електронної комерції авторських виробів декоративно-ужиткового мистецтва
затверджую наказом по СумДУ від *«03» грудня 2024 року № 1257-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 07 грудня 2024 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для розробки інформаційної технології створення мобільного застосунку електронної комерції 3) Розробка інформаційної технології створення мобільного застосунку електронної комерції авторських виробів декоративно-ужиткового мистецтва 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

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

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	15.10 – 17.10	
2	*Огляд технологій, що використовуються для розробки інформаційної технології створення мобільного застосунку електронної комерції*	17.10 – 20.10	
3	*Розробка інформаційної технології створення мобільного застосунку електронної комерції авторських виробів декоративно-ужиткового мистецтва*	21.10 – 30.12	

4	*Аналіз отриманих результатів*	03.12 – 05.12	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	05.12 – 07.12	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 48 стр., 24 рис., 2 додатка, 30 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною, оскільки вона поєднує підтримку місцевих майстрів та унікальної продукції з сучасними можливостями мобільних технологій. Розробка інформаційної технології мобільного додатку для продажу ляльок ручної роботи задовольняє зростаючий попит на авторські вироби, що дозволяє збільшити охоплення ринку, включаючи міжнародний рівень, та сприяє підтримці ремесел.

Об'єкт дослідження — процеси організації та реалізації електронної комерції авторських виробів декоративно-ужиткового мистецтва за допомогою мобільного застосунку.

Предмет дослідження — методи, інструменти та технології, що використовуються для створення мобільного застосунку для продажу авторських виробів ручної роботи.

Мета роботи — розробка інформаційної системи мобільного застосунку для електронної комерції, що сприяє зручності купівлі авторських виробів декоративно-ужиткового мистецтва та підтримці місцевих майстрів.

Методи дослідження — методи аналізу, проєктування, програмування та тестування інформаційних систем.

Результати — розроблено інформаційну систему мобільного застосунку для електронної комерції. Система дозволяє користувачам отримати список доступних ляльок та придбати їх. Реалізовано можливість авторизації та реєстрації користувачів. Додано інтеграцію з платіжною системою для безпечних онлайн-оплат.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ, МОБІЛЬНИЙ ДОДАТОК, POSTGRESQL, REACT NATIVE, EXPRESS.JS, EXPO

ЗМІСТ

ВСТУП

Актуальність. Тема кваліфікаційної роботи є важливою та своєчасною, оскільки вона спрямована на поєднання підтримки локальних майстрів із сучасними можливостями мобільних технологій. Розробка інформаційної системи мобільного застосунку для продажу ляльок ручної роботи відповідає зростаючому попиту на авторські вироби, що дозволяє розширити ринок збуту та підвищити конкурентоспроможність майстрів. Дане рішення сприяє популяризації унікальних ремесел, створює можливості для просування товарів на міжнародному рівні та підтримує розвиток малого бізнесу через використання сучасних інструментів цифрової торгівлі.

Об’єкт дослідження. Процеси організації та реалізації електронної комерції авторських виробів декоративно-ужиткового мистецтва за допомогою мобільного застосунку.

Предмет дослідження. Методи, інструменти та технології, що використовуються для створення мобільного застосунку для продажу авторських виробів ручної роботи.

Новизна. Новизна кваліфікаційної роботи полягає у створенні інформаційної системи, яка дозволяє майстрам продавати унікальні авторські вироби через мобільний застосунок. Цей застосунок поєднує інструменти для інтеграції платіжних систем, створення особистих кабінетів для користувачів та впровадження сучасних технологій обробки замовлень і відстеження доставки товарів. Унікальність рішення полягає у поєднанні функцій для підтримки взаємодії між майстрами та покупцями, що створює нові можливості для просування декоративно-ужиткового мистецтва та розвитку креативної індустрії.

Структура. Дана робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан

Останнім часом ручна робота та унікальні вироби стають дедалі популярнішими серед споживачів. Люди цінують оригінальність, індивідуальний підхід і естетичну привабливість ляльок, створених вручну. Цей тренд веде до зростання попиту на онлайн-магазини, що спеціалізуються на таких товарах. Інтернет-магазини надають майстрам можливість залучити клієнтів з різних країн і розширити свою цільову аудиторію. Завдяки міжнародній доставці ручні ляльки стають доступними для покупців по всьому світу, відкриваючи нові можливості для розвитку бізнесу [23-25].

Згідно з аналітичними дослідженнями, попит на унікальні та ексклюзивні вироби зростає щороку. Споживачі все більше цінують індивідуальний підхід і можливість отримати оригінальні товари, які не мають масових аналогів. Особливу цінність мають авторські ляльки ручної роботи, які створюються в обмеженій кількості, що підвищує їхню цінність в очах покупців. Окрім цього, ручна робота асоціюється з автентичністю, екологічністю та підтримкою малих виробників [23, 25, 26].

Також виникають спеціальні спільноти та форуми, де любителі ручної роботи й колекціонери ляльок обмінюються досвідом, ідеями та навіть замовленнями. Це сприяє підвищенню обізнаності про онлайн-магазини та збільшує їхню популярність. Такі спільноти часто розташовані на платформах соціальних мереж (Facebook, Instagram, TikTok) та спеціалізованих форумах і маркетплейсах (Etsy). Завдяки цим майданчикам майстри можуть просувати свої вироби та отримувати безпосередній зворотний зв’язок від клієнтів.

Технологічний прогрес покращує функціональність та користувацький досвід в інтернет-магазинах [17]. Сучасні можливості, такі як зручні фільтри для пошуку, персоналізовані рекомендації, спрощений процес оформлення замовлення та оплати, а також додаткові функції для відгуків і взаємодії з клієнтами, створюють комфортні умови для покупців. Крім того, інноваційні мобільні додатки для покупок, які дозволяють користувачам переглядати

каталог товарів у зручному форматі зі своїх смартфонів, відіграють важливу роль у підвищенні рівня продажів.

Ще однією важливою складовою є інтеграція інструментів для онлайн-оплати. Сучасні сервіси, такі як **Stripe**, **PayPal** та інші платіжні шлюзи, дозволяють користувачам безпечно та швидко здійснювати платежі без необхідності вводити банківські реквізити щоразу. Інтеграція таких платіжних систем у мобільний додаток забезпечує зручність та безпеку платежів, що позитивно впливає на рішення споживачів щодо покупки [14, 21].

Ще однією помітною тенденцією є застосування **штучного інтелекту (AI)** та **машинного навчання (ML)** для підвищення ефективності інтернет-магазинів. Сервіси персоналізованих рекомендацій дозволяють автоматично пропонувати користувачам ті товари, які можуть їх зацікавити. Сучасні алгоритми дозволяють вивчати уподобання клієнтів на основі їхньої історії покупок і взаємодії з додатком, що допомагає збільшувати продажі та підвищувати рівень задоволеності клієнтів [13, 29].

Крім цього, мобільні застосунки мають значну перевагу над веб-сайтами. Завдяки push-сповіщенням користувачі отримують миттєві повідомлення про акції, знижки та нові надходження товарів, що дозволяє утримувати аудиторію та стимулювати повторні покупки. Застосунки також забезпечують офлайн-доступ до контенту та дозволяють зберігати переглянуті товари, що неможливо у веб-версіях без підключення до мережі [18].

Серед найбільш популярних інструментів для створення інтернет-магазинів ручної роботи виділяються такі платформи, як **Etsy**, **Shopify** та **Amazon Handmade**. Проте створення власного мобільного застосунку відкриває більше можливостей для індивідуалізації бренду та контролю над користувацьким досвідом. Додаток дозволяє майстрам безпосередньо взаємодіяти з клієнтами, пропонувати спеціальні акції та створювати ексклюзивний контент для лояльних клієнтів.

Зважаючи на ці тенденції, створення мобільного застосунку для продажу ляльок ручної роботи є своєчасним та перспективним кроком для

майстрів і підприємців, які прагнуть вийти на нові ринки та підвищити продажі. Сучасні технології дозволяють створити ефективну та зручну платформу для торгівлі унікальними товарами, що задовольняє потреби як виробників, так і покупців [27, 30].

1.2 Аналіз аналогічних проєктів

Мобільний додаток, спеціалізований на продажу ляльок ручної роботи, є інноваційною платформою, яка дозволяє покупцям зі всього світу знаходити та купувати унікальні вироби мистецтва безпосередньо зі своїх смартфонів. Такий додаток не лише створює зручні умови для покупців, надаючи їм доступ до широкого асортименту товарів з різними стилями, матеріалами та дизайнами, але й значно полегшує майстрам презентацію та продаж своїх виробів [25].

У мобільних додатках для продажу товарів ручної роботи зазвичай доступні корисні функції, які роблять процес покупки зручнішим та інтуїтивно зрозумілим. Це можуть бути функції пошуку за категоріями, фільтри, персоналізовані рекомендації, можливість додавання товарів до кошика, оплата онлайн та зручна доставка. Окрім цього, користувачі можуть залишати відгуки та оцінювати товари, що допомагає майстрам будувати репутацію, а новим покупцям — робити вибір на основі відгуків інших користувачів.

Мобільні додатки, що спеціалізуються на виробах ручної роботи, стають все більш популярними через зручність і доступність, яку вони пропонують як майстрам, так і покупцям. Нижче наведено кілька прикладів мобільних додатків, що надають можливість продавати і купувати ляльки ручної роботи та інші авторські вироби.

Декілька прикладів подібних інтернет-магазинів:

1. **Etsy**: Etsy — це глобальний маркетплейс для продажу виробів ручної роботи та вінтажних товарів. Мобільний додаток Etsy дозволяє користувачам переглядати та купувати унікальні ляльки ручної роботи від майстрів з усього світу. Додаток доступний для iOS та Android.

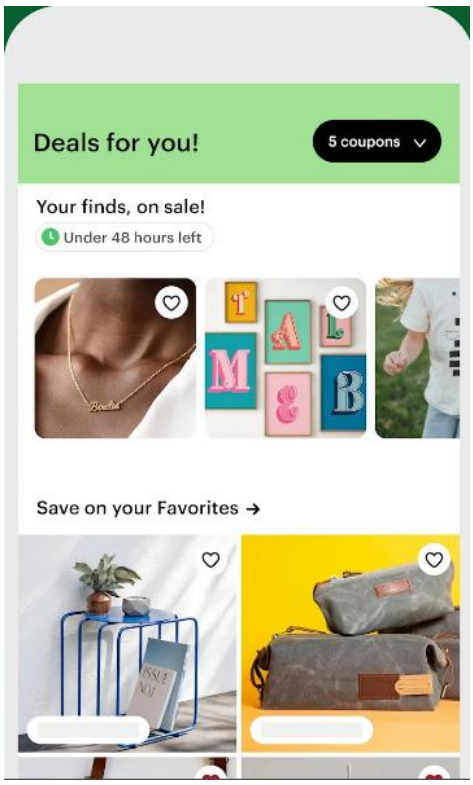


Рисунок 1.1.1 – Приклад мобільного додатку Etsy

2. **Handmade at Amazon**: Amazon пропонує розділ "Handmade", де представлені вироби ручної роботи, включаючи ляльки. Мобільний додаток Amazon надає доступ до цього розділу, дозволяючи користувачам здійснювати покупки безпосередньо зі смартфона.

Рисунок 1.1.2 – Приклад мобільного додатку Amazon

3. Folksy: Folksy — це британський ринок для продажу виробів ручної роботи. Вони пропонують мобільний додаток, де користувачі можуть переглядати та купувати унікальні ляльки та інші вироби.

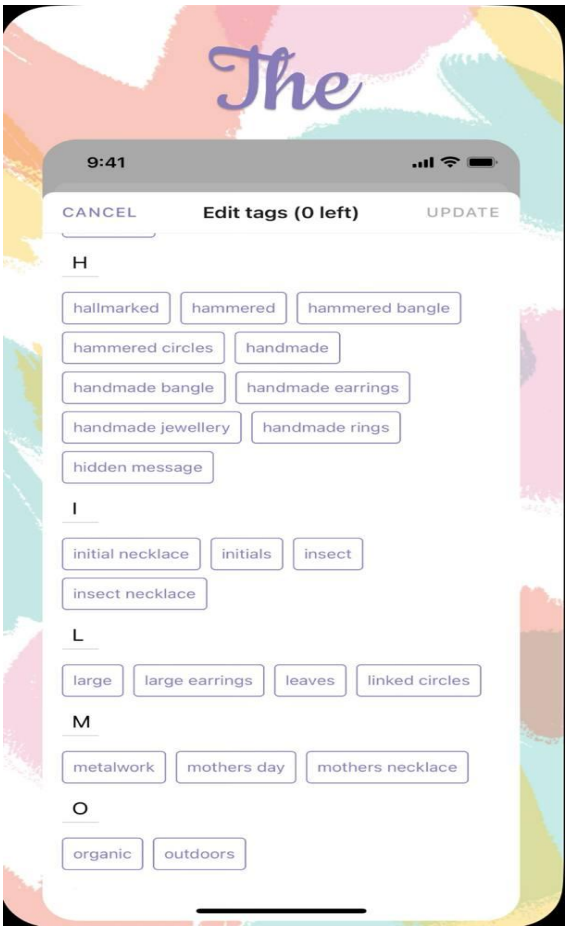


Рисунок 1.1.3 – Приклад мобільного додатку Folksy

Враховуючи сучасний стан електронної комерції та порівнюючи електронну комерцію через веб застосунки та мобільні застосунки, можна дійти висновку, що другі мають низку суттєвих переваг.

По-перше, мобільні застосунки забезпечують кращу продуктивність та швидкодію, оскільки вони працюють безпосередньо на пристрої користувача та можуть зберігати частину даних локально. Це дозволяє прискорити процес завантаження контенту та підвищити стабільність роботи у порівнянні з веб-застосунками, які залежать від швидкості інтернет-з'єднання.

По-друге, мобільні застосунки забезпечують більш персоналізований підхід до користувача. Вони дозволяють надсилати push-повідомлення про акції, знижки або статус замовлення, що підвищує рівень залученості клієнтів.

Завдяки можливості інтеграції з іншими застосунками на смартфоні (камерою, геолокацією тощо), користувачі отримують більш зручний і адаптований досвід.

По-третє, мобільні застосунки працюють навіть в офлайн-режимі, що дозволяє користувачам переглядати інформацію про товари та замовлення без підключення до інтернету. Це підвищує доступність сервісу та забезпечує безперервну взаємодію з клієнтом [17].

Крім того, мобільні застосунки забезпечують більше можливостей для інтеграції з платіжними системами, що спрощує процес оплати товарів та послуг. Користувачі можуть оплачувати замовлення через Apple Pay, Google Pay або інші платіжні сервіси без необхідності вручну вводити платіжні дані, що робить процес швидшим та зручнішим.

З урахуванням вищезазначеного, мобільні застосунки є кращим вибором для електронної комерції, оскільки вони забезпечують більш гнучкий, швидкий та персоналізований досвід користувача, а також дозволяють підвищити ефективність моніторингу товарів та спрощують процес покупки для споживачів.

1.3 Постановка задачі

Метою роботи є розробити інформаційну технологію створення мобільного додатку для продажу унікальних ляльок ручної роботи, що сприятиме залученню клієнтів і підтримці місцевих майстрів.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. **Аналіз ринку мобільних додатків:**
    - Провести дослідження ринку мобільних додатків для продажу виробів ручної роботи та оцінити конкурентне середовище.
2. **Визначення функціональних вимог до мобільного додатку:**
    - Визначити основні функції мобільного додатку, необхідні для зручного та ефективного процесу купівлі-продажу [7].

- Врахувати вимоги до платіжних систем і обробки замовлень для мобільного середовища.

3. **Проектування інтерфейсу та дизайну:**
   - Розробити інтуїтивний і привабливий дизайн, що відображає стиль і естетику ляльок ручної роботи.
   - Застосувати принципи UX-дизайну для покращення взаємодії користувача з мобільним додатком.
4. **Розробка функціоналу та інтеграція:**
   - Реалізувати необхідний функціонал, такий як каталог продукції, кошик покупок, управління замовленнями, система оцінок, пошук тощо.
   - Інтегрувати платіжні системи для мобільних пристроїв.
5. **Тестування та вдосконалення додатку:**
   - Провести тестування функціональності та забезпечити коректну роботу мобільного додатку.
   - Внести корективи на основі результатів тестування для підвищення стабільності та зручності додатку.
6. **Розгортання та підтримка:**
   - Опублікувати додаток на доступних хостинг платформах.
   - Забезпечити технічну підтримку додатку, щоб гарантувати його безперебійне функціонування.
7. **Оцінка результатів:**
   - Проаналізувати ефективність роботи додатку та визначити напрями для його подальшого вдосконалення.

Ці завдання допоможуть створити функціональний і зручний мобільний додаток для продажу ляльок ручної роботи, що сприятиме залученню клієнтів і підтримці майстрів ручної роботи.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Архітектура інформаційної технології

З розвитком сучасних технологій мобільна розробка отримала значний поштовх завдяки зростаючій популярності кросплатформних фреймворків, таких як **React Native**. Використання мобільних застосунків замість веб-додатків стало пріоритетним через підвищену зручність, швидкість та можливість доступу до нативних функцій пристроїв (камера, push-сповіщення тощо) [17-18].

У процесі вибору архітектури інформаційної системи були розглянуті декілька альтернативних підходів. Кожна з цих архітектур мала як переваги, так і недоліки, що зумовило прийняття рішення на користь клієнт-серверної архітектури. Нижче наведено огляд основних альтернативних підходів [1, 2].

1. Peer-to-Peer (P2P) архітектура

Peer-to-Peer (P2P) архітектура передбачає рівноправний обмін даними між пристроями, де кожен пристрій може виконувати як функцію клієнта, так і функцію сервера. У такій архітектурі відсутній центральний сервер, що дозволяє передавати дані безпосередньо між клієнтськими пристроями.

Переваги P2P архітектури:

- **Відсутність центрального сервера**, що знижує витрати на інфраструктуру.
- **Розподілене навантаження** — обробка даних розподіляється між пристроями, що знижує навантаження на один вузол.

Недоліки P2P архітектури:

- **Низький рівень безпеки**, оскільки дані зберігаються на пристроях клієнтів, що збільшує ризик доступу до чутливої інформації.
- **Складна синхронізація даних** між пристроями, що може призводити до втрати або дублювання даних.
- **Обмежена масштабованість** — зі зростанням кількості користувачів навантаження збільшується на кожен вузол, що призводить до зниження

продуктивності.

- **Складність інтеграції з платіжними системами**, оскільки для проведення безпечних фінансових операцій потрібна центральна система обробки.

Причини відмови від P2P архітектури: P2P архітектура не підходить для інформаційної системи електронної комерції через відсутність централізованого контролю та труднощі із забезпеченням безпеки та цілісності даних. Крім того, інтеграція платіжних систем вимагає централізованого сервера для управління платіжними транзакціями [11].

2. Монолітна архітектура

Монолітна архітектура передбачає створення єдиного додатка, що об'єднує всі компоненти системи, включно з серверною логікою, бізнес-логікою та взаємодією з базою даних. Усі частини додатка працюють як єдине ціле.

Переваги монолітної архітектури:

- **Простота розробки** — початкова розробка простіша, оскільки весь функціонал розміщено в одному додатку.
- **Легке розгортання** — для запуску системи достатньо завантажити та запустити один додаток.
- **Швидкість роботи** на початкових етапах при невеликих обсягах даних.

Недоліки монолітної архітектури:

- **Складність масштабування** — система не дозволяє масштабувати окремі частини, тому збільшення продуктивності потребує розширення всього додатка.
- **Труднощі з оновленнями** — зміна будь-якої частини системи може вимагати зупинки та перезапуску всієї програми.
- **Обмежена модульність** — зміна одного модуля може потребувати перероблення всього додатка, що уповільнює процес оновлення та ускладнює підтримку системи.

Причини відмови від монолітної архітектури: Монолітна архітектура

була відхилена через труднощі з масштабованістю, складність внесення змін та оновлень. У контексті системи електронної комерції потрібно забезпечувати постійний доступ до каталогу товарів та можливість здійснювати платежі без простоїв. Монолітна архітектура створює ризик простою всієї системи під час оновлень. Крім того, відсутність можливості масштабування окремих модулів значно знижує гнучкість системи [12].

3. Мікросервісна архітектура

Мікросервісна архітектура передбачає розділення системи на окремі незалежні модулі (сервіси), кожен з яких виконує одну конкретну функцію, наприклад, обробка платежів, управління каталогом товарів або авторизація користувачів. Кожен мікросервіс працює незалежно від інших і взаємодіє з ними через API.

Переваги мікросервісної архітектури:

- **Висока масштабованість** — кожен мікросервіс може бути масштабований окремо, залежно від навантаження.
- **Чітке розділення відповідальності** — кожен мікросервіс відповідає за одну функцію (каталог товарів, обробка платежів тощо).
- **Легкість оновлень та розширень** — оновлення одного сервісу не впливає на роботу інших частин системи.
- **Зручність командної роботи** — кожна команда може працювати над окремим мікросервісом.

Недоліки мікросервісної архітектури:

- **Складність розробки та тестування** — необхідність координації роботи між мікросервісами ускладнює тестування та підтримку.
- **Затримка при взаємодії між сервісами** — обмін даними між сервісами може сповільнювати процеси через додаткові мережеві запити.
- **Збільшення витрат на інфраструктуру** — кожен мікросервіс потребує власного серверного середовища, що підвищує вартість інфраструктури.

Причини відмови від мікросервісної архітектури: Незважаючи на те, що мікросервісна архітектура підходить для великих і складних систем, для

невеликої інформаційної системи мобільного застосунку електронної комерції використання мікросервісів створює зайву складність. Система буде складною в розробці та підтримці, а також вимагатиме додаткових ресурсів на інфраструктуру. Для поточної системи клієнт-серверна архітектура є більш економічно доцільною та забезпечує достатній рівень гнучкості та масштабованості [12, 30].

Після аналізу можливих архітектур було вирішено використати **клієнт-серверну архітектуру з елементами MVVM (Model-View-ViewModel)** із чітким поділом на фронтенд, бекенд та базу даних. Така структура дозволяє забезпечити масштабованість та легкість обслуговування проєкту. Завдяки цьому кожна частина системи може змінюватися окремо, не порушуючи загальної роботи додатку [13, 29].

Основні переваги вибраної архітектури:

1. **Кросплатформність**

Завдяки використанню React Native можна створювати мобільні застосунки, які працюватимуть як на iOS, так і на Android. Це дозволяє значно зменшити час і вартість розробки, оскільки один код обслуговує дві платформи.

2. **Модульність**

Клієнтська частина, серверна частина та база даних ізольовані одна від одної, що дозволяє оновлювати та змінювати кожну частину незалежно.

Структура каталогів розділена на окремі модулі (auth, orders, dolls, stripe), що полегшує підтримку та розвиток проекту.

3. **Масштабованість**

У майбутньому можна додати нові модулі (наприклад, аналітика, доставка тощо) або інтегрувати нові платіжні системи без необхідності змінювати всю архітектуру.

4. **Захист і безпека**

Авторизація здійснюється за допомогою JWT-токенів, які захищають API-запити від неавторизованого доступу.

Для оплати використовується Stripe, який гарантує безпеку транзакцій та відповідає стандартам PCI DSS [14].

5. Простота розробки

Використання Zustand для управління станом на клієнті спрощує розробку реактивного інтерфейсу [19].

Drizzle ORM дозволяє працювати з базою даних без написання складних SQL-запитів, що знижує ймовірність помилок [22].

Попри численні переваги, ця архітектура має й певні недоліки:

1. Залежність від інтернет-з'єднання

Клієнтська частина (мобільний додаток) не зможе повноцінно працювати без стабільного доступу до інтернету, оскільки для отримання даних про товари, оновлення статусів замовлень і платежів необхідний зв’язок із сервером.

2. Затримка між клієнтом і сервером

Клієнт повинен надсилати запити до сервера та чекати відповіді, що може призводити до затримок, особливо коли мережа повільна або сервер перевантажений.

3. Складність синхронізації стану між клієнтом та сервером

Дані на сервері можуть змінюватися у режимі реального часу (наприклад, при зміні товарів або статусу замовлення), але клієнт не завжди отримує ці оновлення миттєво.

4. Залежність від зовнішніх сервісів

Система залежить від Stripe для обробки платежів. У разі збою Stripe користувачі не зможуть оплатити замовлення [11, 27].

2.2 Аналіз інструментів розробки

Основою розробки мобільного застосунку став **React Native** — фреймворк, створений компанією Facebook. Він дозволяє створювати кросплатформні додатки для iOS та Android за допомогою єдиного коду. Це дозволяє значно скоротити витрати на розробку та тестування застосунку,

оскільки один і той самий код працює для обох платформ. Крім того, **React Native** підтримує механізм "гарячого оновлення" (hot-reload), що дозволяє моментально бачити зміни у застосунку під час розробки [17]. Використання React Native також спрощує інтеграцію з іншими інструментами, такими як **Expo**.

Expo було обрано для пришвидшення процесу розробки та тестування застосунку. Цей інструмент дозволяє запускати додаток безпосередньо на мобільному пристрої через застосунок **Expo Go** без необхідності створювати складні білди для iOS або Android. Expo має вбудовані API для роботи з камерою, датчиками та іншими функціями пристроїв. Завдяки цьому стало можливим швидше створювати та тестувати нові функції, а також економити час і ресурси на етапах розробки [18].

Для управління станом додатка було обрано **Zustand**. На відміну від інших менеджерів стану, таких як Redux, Zustand є легшим, швидшим і менш складним у використанні. Він забезпечує простий синтаксис та дозволяє ефективно управляти станом навіть у великих додатках. Zustand дозволяє уникнути надмірної кількості "шуму" у коді, який зазвичай створюється при використанні Redux. Цей інструмент також забезпечує кращу продуктивність завдяки мінімізації кількості повторних рендерів компонентів [19].

Для обробки запитів до серверного API використовується бібліотека **Axios**, яка значно спрощує надсилання HTTP-запитів та обробку відповідей. Axios дозволяє налаштовувати заголовки запитів, встановлювати базову URL-адресу для всіх запитів і автоматично обробляти помилки. Це робить взаємодію з бекендом більш ефективною та зручною для розробників. Завдяки Axios розробники можуть надсилати запити для отримання даних про замовлення, інформацію про ляльки та здійснювати інші запити до серверної частини застосунку [27].

Для роботи з асинхронними запитами та кешування даних у застосунку використовується **React Query** (також відомий як **TanStack Query**). Цей інструмент автоматизує роботу з кешуванням даних, повторними спробами

запитів після помилок і відновленням даних після втрати інтернет-з'єднання. Використання React Query значно спрощує взаємодію з API, оскільки не потрібно самостійно обробляти кеш та синхронізацію даних. Це робить застосунок більш стійким та надійним при роботі зі змінними мережевими умовами [29].

Для обробки платежів у застосунку використовується **Stripe** — сучасна та безпечна платіжна система, яка дозволяє приймати онлайн-платежі. Stripe підтримує кілька платіжних методів, таких як кредитні картки, Apple Pay та Google Pay. Використання Stripe забезпечує відповідність стандартам безпеки PCI-DSS, що дозволяє захистити конфіденційні платіжні дані клієнтів. Stripe також надає можливість налаштувати вебхуки для сповіщень про стан платежів, що дозволяє своєчасно оновлювати статус замовлення у застосунку. Завдяки Stripe користувачі можуть швидко та безпечно здійснювати платежі у мобільному додатку [14].

Технологічний стек застосунку побудований таким чином, щоб забезпечити максимальну ефективність, зручність розробки та надійність роботи застосунку.

Апаратне забезпечення було наступним:

1. **Серверне обладнання**
    - Сервер на Genezio для запуску серверної частини (Node.js + Express.js) та бази даних (PostgreSQL).
2. **Клієнтські пристрої**
    - Мобільні пристрої користувачів (смартфони та планшети) з операційними системами **iOS** та **Android**.
    - Для тестування програми використовуються емулятори пристроїв та реальні пристрої.

2.3 Аналітичний огляд бази даних

Бази даних є основою для зберігання, обробки та управління великими обсягами даних у сучасних інформаційних системах та мобільних застосунках.

Вибір відповідної бази даних значно впливає на продуктивність, надійність, масштабованість та безпеку застосунку. На рисунку 2.3.1 було показано рейтинг найпопулярніших систем управління базами даних.

423 systems in ranking, December 2024

Rank Dec 2024	Rank Nov 2024	Rank Dec 2023	DBMS	Database Model	Score Dec 2024	Score Nov 2024	Score Dec 2023
1.	1.	1.	Oracle	Relational, Multi-model	1263.79	-53.22	+6.38
2.	2.	2.	MySQL	Relational, Multi-model	1003.76	-14.04	-122.88
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	805.69	+5.88	-98.14
4.	4.	4.	PostgreSQL	Relational, Multi-model	666.37	+12.04	+15.47
5.	5.	5.	MongoDB	Document, Multi-model	400.39	-0.54	-18.76
6.	6.	6.	Redis	Key-value, Multi-model	150.27	+1.63	-8.08
7.	7.	↑ 10.	Snowflake	Relational	147.36	+4.87	+27.48
8.	8.	↓ 7.	Elasticsearch	Multi-model	132.32	+0.68	-5.43
9.	9.	↓ 8.	IBM Db2	Relational, Multi-model	122.78	+1.04	-11.81
10.	10.	↑ 11.	SQLite	Relational	101.72	+2.24	-16.23

Рисунок 2.3.1 – рейтинг найпопулярніших СУБД

Для створення мобільного додатку з продажу ляльок ручної роботи було обрано **PostgreSQL** як основну базу даних, що є сучасним рішенням для надійного зберігання та обробки структурованих даних. PostgreSQL — це об’єктно-реляційна система керування базами даних (СУБД) з відкритим кодом, яка активно розвивається та підтримується спільнотою розробників у всьому світі. Її основна перевага — здатність обробляти як структуровані, так і напівструктуровані дані, що робить її універсальним інструментом для багатьох сучасних додатків [20].

Основні переваги PostgreSQL:

1. **Підтримка ACID-транзакцій**

PostgreSQL повністю підтримує транзакції за принципом ACID (атомарність, узгодженість, ізольованість та надійність), що забезпечує збереження цілісності даних навіть у разі збоїв системи або відмови сервера. Це критично важливо для мобільного застосунку, де транзакції можуть включати створення замовлень, оплати та інші операції [15].

2. **Підтримка складних запитів**

PostgreSQL підтримує складні SQL-запити, включно з підзапитами, об'єднаннями (JOIN), агрегаціями та аналітичними функціями. Це дозволяє

виконувати ефективний пошук і фільтрацію замовлень, товарів та користувачів за різними критеріями.

3. Масштабованість та продуктивність

PostgreSQL може обробляти великі обсяги даних і підтримувати масштабування вгору (вертикальне масштабування) та горизонтальне масштабування за допомогою шардінгу та реплікації. Завдяки цьому база даних може забезпечувати високу продуктивність навіть за умови великого навантаження користувачів.

4. Розширюваність

PostgreSQL дозволяє створювати власні функції, тригери та розширення. Вона підтримує такі розширення, як PostGIS (для геопросторових запитів) та pg_trgm (для пошуку за подібністю тексту). Це розширює можливості розробки та спрощує інтеграцію нових функцій.

5. JSON та NoSQL-функціональність

PostgreSQL дозволяє зберігати та обробляти JSON-документи, що надає можливість комбінувати підхід реляційних та документно-орієнтованих баз даних. У мобільному застосунку це може бути корисним для зберігання додаткових атрибутів товарів та параметрів, які не мають жорсткої структури.

6. Безпека та контроль доступу

PostgreSQL має потужні механізми контролю доступу. Вона підтримує розмежування прав доступу до таблиць, рядків і навіть окремих стовпців. Це дозволяє створювати багаторівневу безпеку та контролювати доступ користувачів до певних даних.

7. Відкритий код і активна спільнота

PostgreSQL — це СУБД з відкритим вихідним кодом, яка активно підтримується спільнотою розробників. Це означає, що будь-які оновлення та виправлення доступні для розробників безкоштовно. Відкрита ліцензія дозволяє використовувати PostgreSQL без додаткових витрат, що робить її ідеальним вибором для стартапів та невеликих проєктів.

Основні недоліки PostgreSQL:

1. **Складність налаштування та адміністрування**

PostgreSQL має широкий набір інструментів для налаштування та оптимізації, що може ускладнити процес її адміністрування. Підтримка великих баз даних вимагає знань з налаштування та моніторингу продуктивності.

2. **Швидкість запису при великих обсягах даних**

При дуже великих обсягах даних швидкість запису може знижуватися, особливо якщо не використовуються методи шардінгу. Для великих застосунків це може стати проблемою, якщо не застосовуються спеціальні стратегії оптимізації.

3. **Обмеження горизонтального масштабування**

Масштабування PostgreSQL на кількох серверах складніше, ніж у деяких NoSQL-базах, таких як MongoDB. Для цього потрібна додаткова інфраструктура та спеціальні інструменти для реплікації та шардінгу.

4. **Затримки при оновленнях версій**

Перехід з однієї версії PostgreSQL на іншу може вимагати міграції даних та зміни конфігурації, що може викликати затримки в роботі системи. Оновлення версій потребує чітко спланованого підходу, особливо у великих системах [20].

На відміну від MongoDB, яка зберігає дані у вигляді JSON-документів, PostgreSQL дозволяє створювати чітко визначені схеми бази даних з визначеними типами даних та зв’язками між таблицями. Це забезпечує більшу структурованість даних та дозволяє уникнути дублювання. Порівняно з MySQL, PostgreSQL має кращу підтримку транзакцій за стандартом ACID, розширені можливості для складних запитів та підтримує більший набір аналітичних функцій. У порівнянні з NoSQL-базами, такими як Redis або Firebase, PostgreSQL дозволяє працювати зі складними структурованими даними та забезпечує кращу узгодженість та цілісність даних [15].

Основними причинами вибору саме цієї СУБД були:

1. **Структурованість даних**

База даних повинна підтримувати чітку структуру даних для зберігання інформації про користувачів, замовлення, ляльки, відгуки та платежі. PostgreSQL дозволяє створювати зв’язки між таблицями, що спрощує подальший пошук та обробку даних.

2. Транзакції

Процес оформлення замовлень та здійснення платежів потребує надійної обробки транзакцій, щоб уникнути помилок або втрати даних. PostgreSQL забезпечує ACID-транзакції, що гарантує цілісність даних навіть у разі помилок.

3. Гнучкість запитів

Завдяки потужній мові SQL та підтримці складних запитів, PostgreSQL дозволяє створювати персоналізовані рекомендації для користувачів, обробляти фільтри товарів та створювати статистичні звіти.

4. Можливість горизонтального масштабування

У разі зростання обсягів даних можна додати реплікацію для підвищення продуктивності системи або створити резервне копіювання бази для забезпечення надійності.

5. Відкритий код

Відсутність ліцензійних платежів робить PostgreSQL економічно вигідним вибором, що дозволяє створювати доступні та масштабовані рішення навіть для стартапів.

PostgreSQL є оптимальним вибором для розробки сучасної інформаційної системи електронної комерції. Завдяки її можливостям забезпечується збереження цілісності даних, безпека та масштабованість. Вибір PostgreSQL забезпечує стабільність роботи та дає змогу ефективно обробляти великі обсяги даних [20].

2.4 Нефункціональні вимоги

Нефункціональні вимоги є важливим компонентом проєктування інформаційної системи, оскільки вони забезпечують ефективність, безпеку та

задоволеність користувачів. Ці вимоги визначають характеристики, що впливають на продуктивність, масштабованість, доступність, безпеку та інші параметри, необхідні для надійної роботи мобільного застосунку для продажу ляльок ручної роботи. Вони формують основу для створення якісного та зручного застосунку, який здатний задовольнити потреби користувачів та бізнесу [27, 30].

Однією з найважливіших вимог до застосунку є забезпечення безпеки. Для цього система повинна використовувати надійні протоколи аутентифікації та авторизації, що запобігають несанкціонованому доступу. Дані користувачів мають бути захищені за допомогою шифрування, а передача даних між клієнтською та серверною частинами має відбуватися за допомогою безпечних протоколів, таких як HTTPS. Важливо також обмежити доступ до критичних даних лише авторизованим користувачам та забезпечити захист від спроб несанкціонованого втручання [14, 17, 20].

Значна увага приділяється продуктивності застосунку. Він повинен забезпечувати швидку реакцію на дії користувачів, зокрема, час відгуку має бути не більше 2 секунд під час перегляду каталогу ляльок або оформлення замовлення. Система повинна стабільно працювати при одночасному використанні її багатьма користувачами та ефективно обробляти великі обсяги запитів. Для досягнення цієї мети використовуються оптимізовані запити до серверної частини та застосовується кешування даних для зменшення часу відповіді.

Масштабованість є ще однією ключовою вимогою до мобільного застосунку. Система повинна бути спроєктована таким чином, щоб забезпечувати горизонтальне та вертикальне масштабування серверної інфраструктури для обробки зростаючої кількості користувачів та замовлень. Це передбачає можливість динамічного збільшення серверних ресурсів для підтримки зростаючого навантаження, а також можливість обробки великої кількості запитів без погіршення продуктивності [29].

Доступність застосунку також є важливою нефункціональною вимогою.

Застосунок має бути доступним для користувачів 24/7, що дозволяє забезпечити безперебійний доступ до його функціоналу. Середній час безвідмовної роботи (SLA) повинен становити не менше 99,9%, що гарантує безперервну доступність для користувачів навіть за умови технічних проблем чи планових оновлень системи. Система повинна забезпечувати резервування та відновлення роботи у разі збоїв.

Обслуговуваність застосунку полягає у простоті внесення змін та оновлень. Програма повинна мати модульну структуру, що дозволяє легко додавати нові функції або змінювати існуючі. Система повинна підтримувати можливість оновлення без необхідності повного перезапуску, а код має бути добре документований для полегшення його підтримки іншими розробниками. Обслуговуваність також передбачає можливість автоматичних оновлень через магазини застосунків, такі як Google Play та App Store.

Ще однією важливою вимогою є переносимість системи. Оскільки застосунок розробляється для роботи на пристроях з операційними системами Android та iOS, він повинен забезпечувати однакову функціональність та інтерфейс користувача на різних пристроях. Це досягається за допомогою кросплатформного фреймворка (наприклад, React Native), який дозволяє створювати єдиний код для всіх платформ. Такий підхід спрощує підтримку застосунку та знижує витрати на розробку [7].

Надійність системи полягає у забезпеченні стабільної роботи навіть у разі збою мережі або пристрою користувача. Критичні дані, такі як інформація про замовлення, повинні бути синхронізовані із сервером, щоб запобігти втраті інформації. Для захисту від збоїв застосовується система резервного копіювання даних та механізми автоматичного відновлення роботи. Завдяки цьому мінімізуються помилки у роботі застосунку та скорочується час простою.

Юзабіліті, або зручність використання, є важливою вимогою, оскільки забезпечує простоту взаємодії користувачів із системою. Інтерфейс застосунку має бути інтуїтивно зрозумілим та зручним для використання. Користувачі

повинні мати можливість виконувати основні дії за кілька натискань. Основний акцент робиться на створенні логічної структури навігації та відповідності інтерфейсу сучасним стандартам UI/UX-дизайну. Застосунок має підтримувати принципи Human Interface Guidelines та Material Design [6].

Сумісність системи передбачає можливість інтеграції із зовнішніми сервісами та інструментами. Зокрема, застосунок повинен підтримувати інтеграцію із платіжними системами (наприклад, Stripe) для обробки платежів. Крім того, система повинна бути сумісною із зовнішніми API для обміну даними та надавати можливість роботи із сторонніми інструментами, такими як аналітичні сервіси Firebase Analytics або Google Analytics.

Нарешті, забезпечення приватності та конфіденційності даних користувачів є критично важливим. Усі особисті дані, такі як паролі та дані платежів, мають бути зашифровані для запобігання несанкціонованому доступу. Доступ до конфіденційної інформації має бути обмеженим та контролюватися за допомогою політики доступу. Уся обробка персональних даних повинна здійснюватися відповідно до Загального регламенту про захист даних (GDPR) та законодавства України про захист персональних даних.

Таким чином, нефункціональні вимоги забезпечують фундаментальну основу для ефективного функціонування, безпеки та масштабованості мобільного застосунку для продажу ляльок ручної роботи. Вони охоплюють безпеку, продуктивність, масштабованість, доступність, обслуговуваність, переносимість, надійність, зручність використання, сумісність та конфіденційність даних. Усі ці вимоги сприяють створенню високоякісного та зручного застосунку, який здатний задовольнити потреби як користувачів, так і бізнесу.

2.5 Функціональні вимоги

Функціональні вимоги визначають основний набір можливостей та дій, які повинен виконувати мобільний застосунок для продажу ляльок ручної роботи. Ці вимоги встановлюють конкретні функції та завдання, які

забезпечують основний користувацький досвід, включаючи перегляд каталогу товарів, оформлення замовлення, обробку платежів та інші важливі операції. Виконання цих вимог гарантує ефективну взаємодію користувача із системою та забезпечує відповідність бізнес-цілям проєкту [9].

Основною функціональною вимогою є можливість перегляду каталогу товарів. Користувач повинен мати доступ до повного списку ляльок із зображеннями, назвами, цінами та описами. Для зручності користувачів застосунок має забезпечувати фільтрацію та сортування товарів за різними критеріями (ціна, популярність, новинки тощо). Це дозволяє спростити процес пошуку необхідного товару та підвищує зручність використання застосунку. Застосунок також повинен підтримувати функцію пошуку, що дає змогу швидко знаходити конкретний товар за його назвою або ключовими словами.

Важливою функцією є можливість перегляду детальної інформації про товар. Користувачі повинні мати можливість натиснути на будь-який товар у каталозі та побачити детальний опис, додаткові фотографії, матеріали виготовлення, розміри та інші характеристики. Такий підхід дозволяє потенційним покупцям зробити більш усвідомлений вибір перед покупкою. Крім того, користувачі повинні мати можливість переглядати відгуки та рейтинги інших клієнтів, щоб отримати більш повне уявлення про якість товару.

Функціональні вимоги також передбачають можливість додавання товарів до кошика. Користувач повинен мати можливість вибрати потрібну кількість товарів та додати їх до кошика для подальшого оформлення замовлення. Система повинна дозволяти користувачам переглядати вміст кошика, редагувати кількість товарів або видаляти непотрібні позиції. Це забезпечує гнучкість і контроль користувачів над своїм замовленням [6].

Процес оформлення замовлення є ключовою функцією мобільного застосунку. Користувачі повинні мати можливість перейти до оформлення замовлення з кошика, заповнити необхідну інформацію (ім'я, адресу доставки та контактні дані) та вибрати спосіб оплати. Застосунок повинен забезпечити

підтримку безпечних методів оплати, таких як оплата банківською карткою через інтеграцію з платіжними системами (наприклад, Stripe). Після успішної оплати користувач повинен отримати повідомлення про підтвердження замовлення, включаючи деталі замовлення та передбачувану дату доставки.

Інтеграція з платіжною системою є ще одним важливим функціональним елементом. Застосунок має забезпечувати безпечну обробку платежів за допомогою платіжного шлюзу (наприклад, Stripe) [16]. Процес оплати повинен бути швидким та безпечним. Користувачі повинні мати можливість оплачувати товари за допомогою різних способів, таких як банківські картки та інші платіжні сервіси. Платіжна інформація повинна бути захищена за допомогою шифрування та інших сучасних методів безпеки [14].

Ще одним важливим аспектом є можливість відстеження статусу замовлення. Після оформлення замовлення користувач повинен мати доступ до інформації про його статус (наприклад, "в обробці", "відправлено", "доставлено"). Це дозволяє користувачам бути в курсі стану свого замовлення та забезпечує прозорість процесу купівлі. В особистому кабінеті користувач повинен мати можливість переглядати історію своїх замовлень та їх статуси.

Персоналізований користувацький досвід є ще однією важливою вимогою. Застосунок повинен підтримувати створення користувацьких облікових записів, де користувачі можуть зберігати свої особисті дані, історію замовлень, списки бажань (wishlist) та персоналізовані рекомендації товарів. Система повинна забезпечувати можливість редагування профілю користувача та зміну пароля.

Доступ до персоналізованих повідомлень та сповіщень є ще однією функціональною вимогою. Користувачі повинні отримувати повідомлення про нові акції, знижки або статус своїх замовлень. Сповіщення мають бути ненав'язливими та інформативними. Застосунок повинен мати можливість надсилати push-сповіщення для інформування користувачів про нові товари або зміни в статусі замовлення [17].

Система має забезпечувати багатомовну підтримку. Для залучення

міжнародної аудиторії застосунок повинен підтримувати кілька мовних інтерфейсів. Це забезпечує можливість користувачам із різних країн взаємодіяти із застосунком рідною мовою, що покращує користувацький досвід та підвищує лояльність клієнтів.

Ще однією важливою функцією є надання користувачам можливості залишати відгуки та оцінки товарів. Це дозволяє іншим користувачам робити більш усвідомлений вибір під час купівлі. Відгуки мають містити можливість додавати текстові коментарі та оцінки у вигляді зірок. Відгуки користувачів дозволяють майстрам отримувати зворотний зв'язок та покращувати свою продукцію.

Система повинна також підтримувати функціональність зв’язку з підтримкою клієнтів. Користувачі повинні мати можливість зв'язатися зі службою підтримки для вирішення питань або отримання допомоги. Це може бути реалізовано через чат-підтримку або систему звернень, де користувачі можуть залишити свої запитання та отримати відповідь.

Загалом функціональні вимоги до інформаційної системи електронної комерції включають можливості перегляду каталогу товарів, оформлення замовлення, обробки платежів, перегляду статусу замовлень, персоналізованих сповіщень та багатомовної підтримки. Вони спрямовані на створення простого, інтуїтивно зрозумілого інтерфейсу, який дозволить користувачам легко знаходити потрібні товари та здійснювати покупки з максимальним комфортом. Забезпечення цих функціональних можливостей дозволить підвищити рівень задоволеності користувачів, сприятиме збільшенню конверсії та сприятиме розвитку ремісничої спільноти, що створює унікальні товари ручної роботи [30].

3 ТЕХНІЧНА РЕАЛІЗАЦІЯ

3.1 Архітектура проєкту

Процес роботи починається з вибору середовища розробки, планування майбутньої архітектури файлів та необхідних пакетів. Visual Studio Code (**VS Code**) — це безкоштовне інтегроване середовище розробки (IDE) від Microsoft, доступне для Windows, macOS та Linux. Воно підтримує безліч мов програмування та має широкий вибір розширень для автодоповнення, відладки, інтеграції з Git та підсвічування синтаксису. Завдяки інтуїтивному інтерфейсу та можливості налаштування під потреби користувачів, VS Code є одним із найпопулярніших інструментів для розробників.

Архітектура проєкту складається з бекенд- (рис. 3.1.1) та фронтенд-частини (рис. 3.1.2), які взаємодіють для забезпечення функціональності мобільного застосунку. Основні модулі виконують різні ролі, включаючи серверну обробку, збереження даних, створення інтерфейсу користувача і взаємодію з API [4].

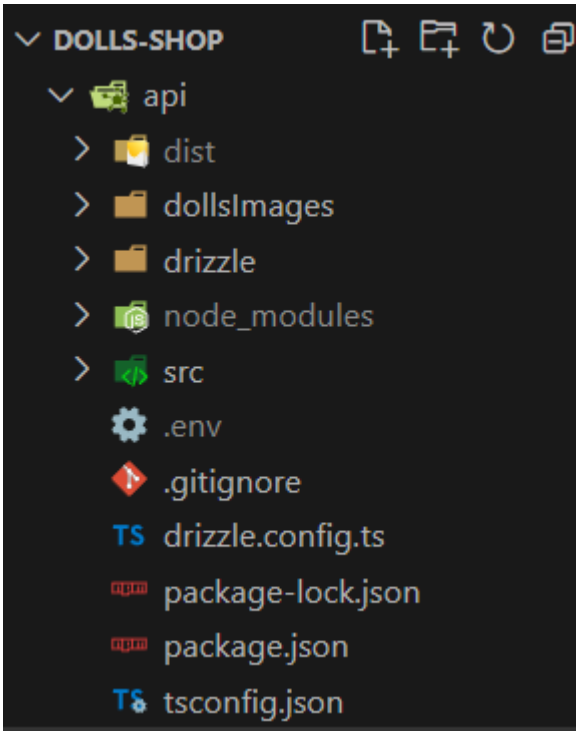


Рисунок 3.1.1 – серверна частина проєкту

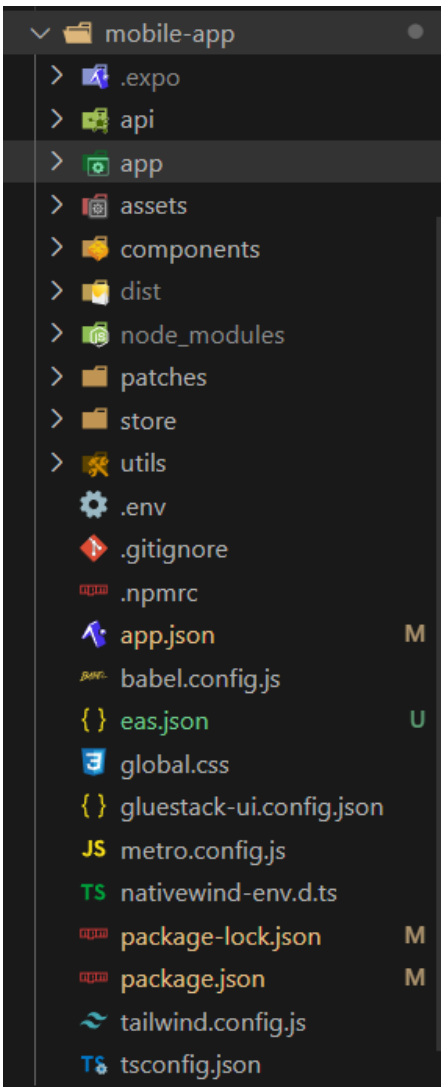


Рисунок 3.1.2 – Клієнтська частина проєкту

Для серверної частини додатку точкою входу є файл **src/index.ts** [6]

Саме в ньому:

1. Налаштовується середовище (наприклад, завантаження змінних із файлу **.env**).
2. Ініціалізуються основні залежності, такі як підключення до бази даних (PostgreSQL через Drizzle ORM).
3. Запускається сервер за допомогою **Express**:
4. Сервер починає прослуховувати вказаний порт, і стає доступним для запитів з клієнтської частини.

Для клієнтської частини точкою входу є файл **app/_layout.js**, який:

1. Завантажує кореневі компоненти програми через **Expo Router**, що відповідає за маршрутизацію сторінок (наприклад, головна сторінка, кошик, історія замовлень).
2. Ініціалізує глобальні стилі, такі як **Tailwind CSS**, та підключає бібліотеки для управління станом (**Zustand**).
3. Забезпечує рендеринг початкового інтерфейсу (головна сторінка магазину).

Сторона	Бібліотека/Фреймворк	Опис
1	2	3
Бекенд	Express	Фреймворк для побудови API, що забезпечує обробку HTTP-запитів і маршрутизацію.
	Drizzle ORM	Бібліотека для взаємодії з PostgreSQL, яка використовує TypeScript для типізації.
	Stripe SDK	Інтеграція з платіжною системою Stripe для обробки оплат.
	bcrypt	Використовується для хешування паролів, забезпечуючи безпеку даних користувачів.
	JWT	Використовується для аутентифікації через токени JSON Web Token.
Фронтенд	React Native	Основний фреймворк для розробки мобільного застосунку.
	Expo Router	Інструмент для маршрутизації сторінок у мобільному застосунку.
	Gluestack UI	Набір компонентів для швидкої розробки інтерфейсу.
	NativeWind	Інтеграція Tailwind CSS для React Native, що спрощує стилізацію компонентів.
	Zustand	Бібліотека для управління станом, яка забезпечує легкість і простоту роботи з даними.
	Axios	Використовується для роботи з HTTP-запитами до бекенду.

Таблиця 3.1.1 – Опис використаних бібліотек

3.2 Реалізація серверної частини проєкту

Розробка серверної частини проєкту включає два основні етапи: проєктування структури бази даних та створення моделей для її взаємодії. У проєкті станом на зараз було створено три основні сутності: **ляльки, користувачі та замовлення**. Ці сутності стали основою для створення моделей бази даних [3, 5]:

- **dollsSchema** – модель, яка містить дані, пов’язані з ляльками, включаючи: ідентифікатор ляльки (*id*), назву (*dollName*), URL зображення (*image*), опис (*description*), ціну (*price*), лічильник переглядів (*viewsCount*), дату створення (*createdAt*) та оновлення (*updatedAt*).
- **usersSchema** – модель для зберігання даних про користувачів: ідентифікатор (*id*), ім’я (*name*), URL аватару (*avatarUrl*), електронну пошту (*email*), пароль (*password*), адресу (*address*), роль (*role*), дату створення (*createdAt*) та оновлення (*updatedAt*).
- **ordersSchema** – модель для замовлень, що містить: ідентифікатор замовлення (*id*), статус замовлення (*status*), ідентифікатор користувача (*userId*), ідентифікатор платежу в Stripe (*stripePaymentIntentId*), дату створення (*createdAt*) та оновлення (*updatedAt*).

Для того, щоб була можливість отримати список товарів замовлення, була створена суміжна таблиця **orderItemsTable**

Для того, щоб була можливість отримати список товарів замовлення, була створена суміжна таблиця **orderItemsTable**

- **orderItemsTable –** модель для списку товарів у замовленні, що містить: ідентифікатор списку (*id*), ідентифікатор замовлення (*orderId*), ідентифікатор ляльки (*dollId*), кількість штук обраного товару (*quantity*), ціна за товар, помножена на кількість (*price*).

Знаходяться ці схеми в директорії ***src/db/*** (рис. 3.2.1)

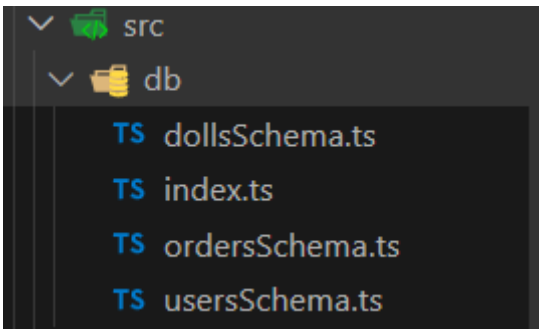


Рисунок 3.2.1 – структура моделей

В корені бекенд частини проєкту знаходиться директорія, що є тимчасовим сховищем для зображень ляльок(рис. 3.2.2)

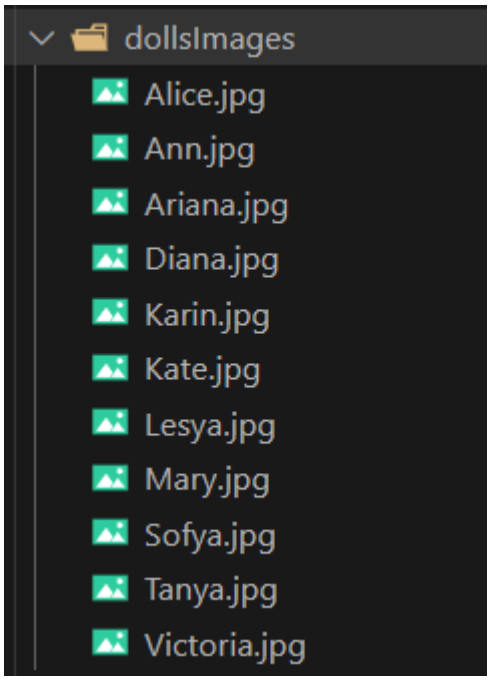


Рисунок 3.2.2 – сховище для зображень ляльок

- Директорія **middlewares**:

Дана директорія містить файли, що відповідають за обробку проміжної логіки запитів до сервера.

authMiddleware.ts — Middleware для перевірки авторизації користувачів. Перевіряє наявність токена в запитах і визначає, чи має користувач доступ до запитуваних ресурсів.

validationMiddleware.ts — Middleware для перевірки валідації даних. Перевіряє правильність вхідних даних, таких як форма запиту, перед виконанням логіки контролера. Це допомагає запобігти помилкам через некоректні дані.

- Директорія **routes**:

Дана директорія містить маршрути, які відповідають за обробку запитів і визначають, який контролер викликати для конкретного маршруту.

auth/index.ts — Обробляє маршрути, пов'язані з автентифікацією, такі як реєстрація користувача, вхід у систему.

dolls/dollsController.ts — Контролер для роботи з ляльками, обробляє запити, пов'язані з переглядом, додаванням, оновленням та видаленням ляльок.

orders/ordersController.ts — Контролер для роботи з замовленнями. Використовується для обробки запитів щодо створення, перегляду або управління замовленнями.

stripe/stripeController.ts — Контролер, який інтегрує Stripe API для обробки платежів. Обробляє маршрути, пов'язані з оплатами.

Файли **index.ts** в кожній піддиректорії — Описують маршрути для відповідних функціональностей і об'єднують їх у єдиний набір, щоб забезпечити простоту інтеграції.

- Директорія **types**:

Ця директорія містить типи і розширення для Express та інших компонентів проєкту.

express/index.d.ts — Файл оголошення типів, використовується для додавання кастомних типів або властивостей у стандартні типи Express. Наприклад, додає властивість rawBody до об'єкта запиту (Request) для підтримки обробки Stripe Webhooks. Це дозволяє проєкту бути більш типізованим та уникати помилок типів у TypeScript.

3.3 Реалізація клієнтської частини проєкту

Робота з клієнтською частиною мобільного додатку розпочинається зі створення інтерфейсу користувача для взаємодії з додатком та реалізації логіки отримання даних із серверної частини. Для цього було використано React Native з Expo, що дозволяє швидко та ефективно створювати мобільні

додатки з можливістю кросплатформеного використання [8].

У проєкті було використано низку додаткових бібліотек для полегшення розробки, зокрема Tailwind CSS для стилізації компонентів, Zustand для управління глобальним станом, а також Gluestack UI для побудови адаптивних інтерфейсів [10].

Архітектура клієнтської частини додатку була організована в кілька основних директорій для спрощення навігації, модульності та логічної організації компонентів.

Далі буде наведено представлення роботи мобільного додатку у вигляді діаграми (рис.3.3.1):

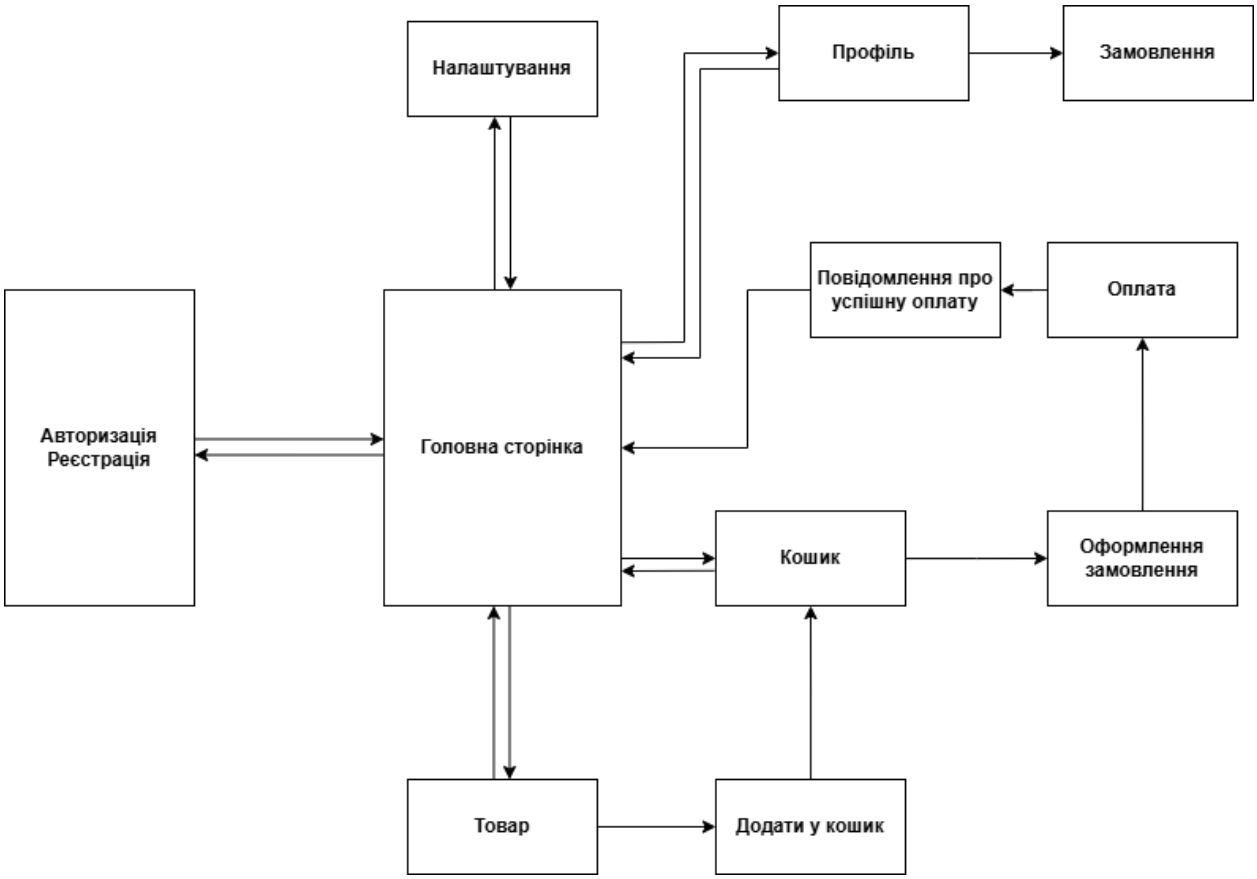


Рисунок 3.3.1 – Візуалізація роботи мобільного застосунку

Основні модулі в кореневій директорії:

- **app.json** – файл конфігурації додатку для Expo, в якому описано метадані проєкту, налаштування платформи та шляхи до ресурсів.
- **babel.config.js** – файл конфігурації Babel, який використовується для

трансформації коду.

- **eas.json** – файл налаштувань Expo Application Services (EAS) для автоматизації побудови та деплойменту.
- **metro.config.js** – файл налаштувань Metro Bundler, що використовується для оптимізації збірки коду.

Основні директорії:

1. **api** – зберігає файли для здійснення запитів до серверної частини:
    - **auth.ts** – обробка запитів, пов’язаних із авторизацією.
    - **dolls.ts** – обробка запитів, пов’язаних з отриманням даних про ляльки.
    - **orders.ts** – обробка запитів, пов’язаних із замовленнями.
    - **stripe.ts** – інтеграція з платіжною системою Stripe.
2. **app** – містить сторінки додатку, розділені за функціональними модулями:
    - **auth** – зберігає сторінки авторизації та реєстрації:
        - **login.tsx** – сторінка авторизації.
        - **register.tsx** – сторінка реєстрації.
    - **doll** – модуль, пов’язаний із відображенням деталей ляльки:
        - **[id].tsx** – сторінка детальної інформації про конкретну ляльку.
    - **order** – модуль, пов’язаний із замовленнями:
        - **[id].tsx** – сторінка детальної інформації про конкретне замовлення.
    - **cart.tsx** – сторінка кошика.
    - **profile.tsx** – сторінка профілю користувача.
    - **settings.tsx** – сторінка налаштувань.

3. **components** – зберігає основні UI-компоненти:
    - **CustomStripeProvider.tsx** – інтеграція Stripe для обробки платежів.
    - **EmptyCart.tsx** – компонент порожнього кошика.
    - **ProductListItem.tsx** – компонент для відображення інформації про ляльку в списку.
    - **Layout.tsx** – компонент загального макету сторінок.
4. **store** – для управління глобальним станом за допомогою Zustand:
    - **authStore.ts** – зберігає стани, пов’язані з авторизацією.
    - **cartStore.ts** – зберігає стани, пов’язані з кошиком.
5. **utils** – зберігає допоміжні файли:
    - **darkmode.context.ts** – файл для реалізації темної теми.
6. **assets** – містить статичні файли, такі як зображення, логотипи тощо.

Ця структура проєкту забезпечує зручність у розробці, підтримці та розширенні функціональності додатку. Кожен модуль чітко відокремлений і виконує свою функцію, що сприяє підвищенню продуктивності у розробці.

3.4 Інструкція з використання інформаційної технології

З ціллю подальшого вдосконалення проєкту, було прийнято рішення не делоїти мобільний застосунок у App Store та Google Play Store, тому для можливості протестувати функціонал, додаток було опублікувано через Expo Go. Кожен може його протестувати, встановивши на свій пристрій Expo Go та перейти по посиланню, або відсканувавши QR-код (рис. 3.4.1) через камеру свого пристрою, або у додатку Expo Go.

Рисунок 3.4.1 – QR-код застосунку

Коли користувач відкриває додаток, він бачить перед собою домашню сторінку (рис. 3.4.2).

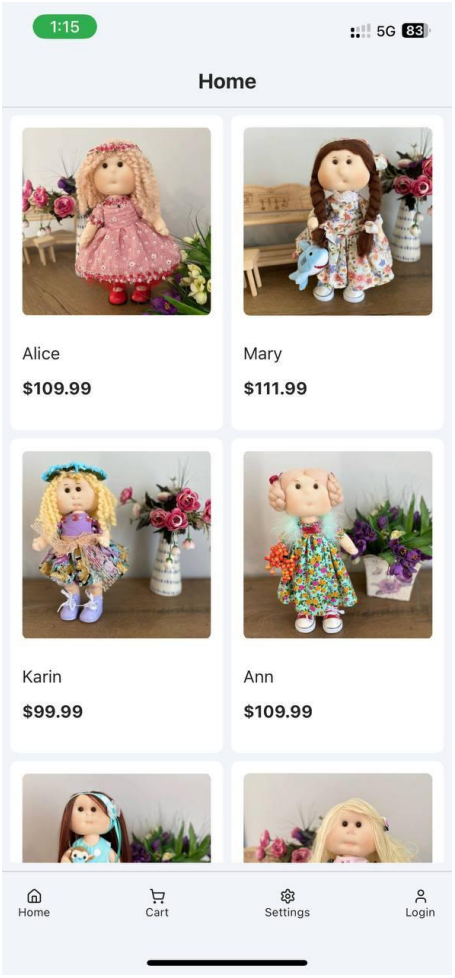


Рисунок 3.4.2 – Домашня сторінка

На цій сторінці є можливість передивитись список ляльок, доступних для придбання. Користувач має можливість переходити по різним сторінкам через навігаційне меню внизу єкрану (рис. 3.4.3).

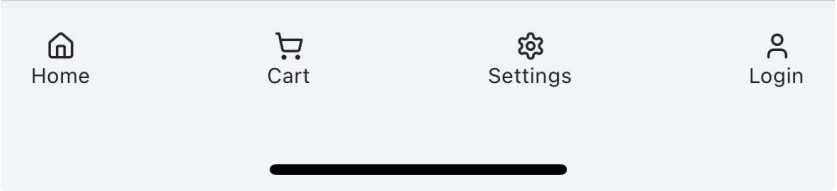


Рисунок 3.4.3 – Навігаційне меню

Натиснувши на кнопку **Login**, користувач переходить на сторінку авторизації (рис. 3.4.4):

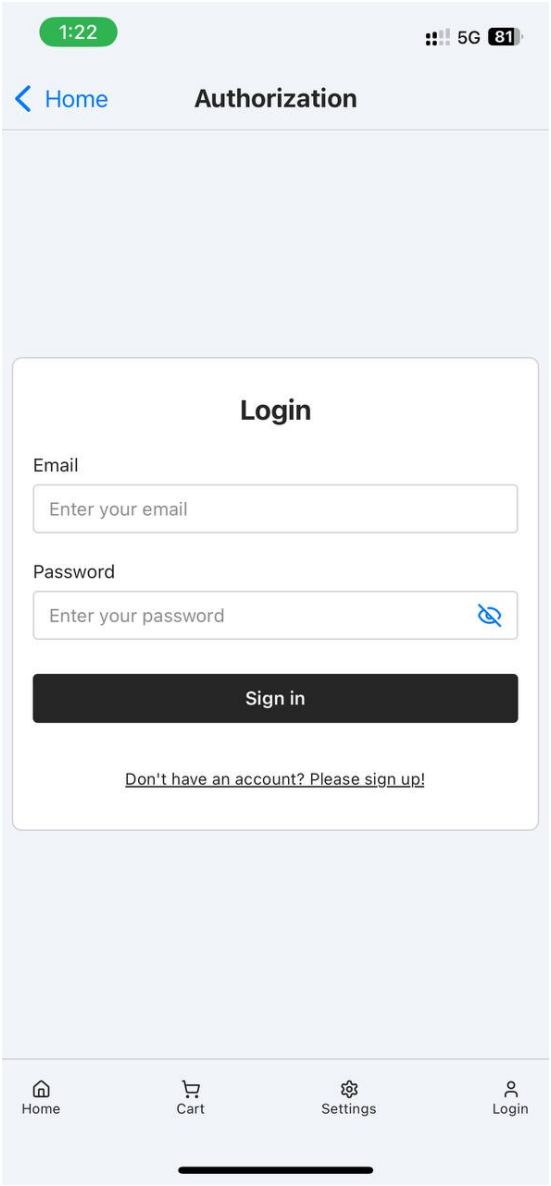


Рисунок 3.4.4 – Єкран авторизації

Якщо ж у користувача ще немає облікового запису, він може зареєструватись, натиснувши на посилання “Don’t hav an account? Please sign up!”, що б перейти на сторінку реєстрації нового користувача (рис. 3.4.5):

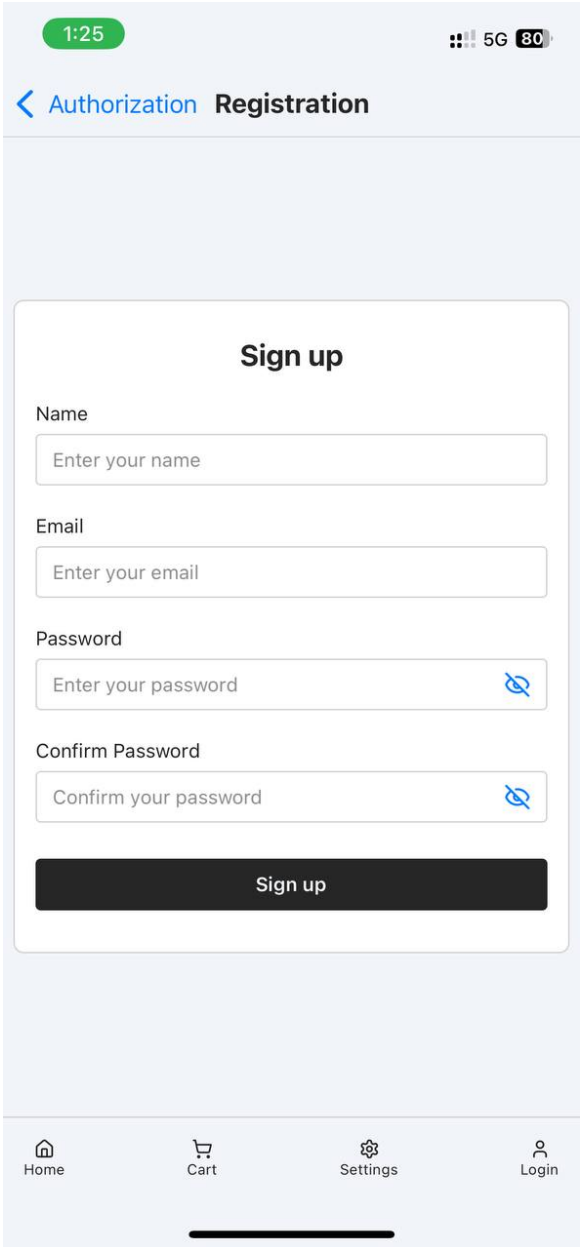


Рисунок 3.4.5 – Єкран реєстрації

Після успішної авторизації чи реєстрації, користувач повертається на головну сторінку, де в меню змінюється підпис кнопки авторизації на “Profile” (рис. 3.4.6):

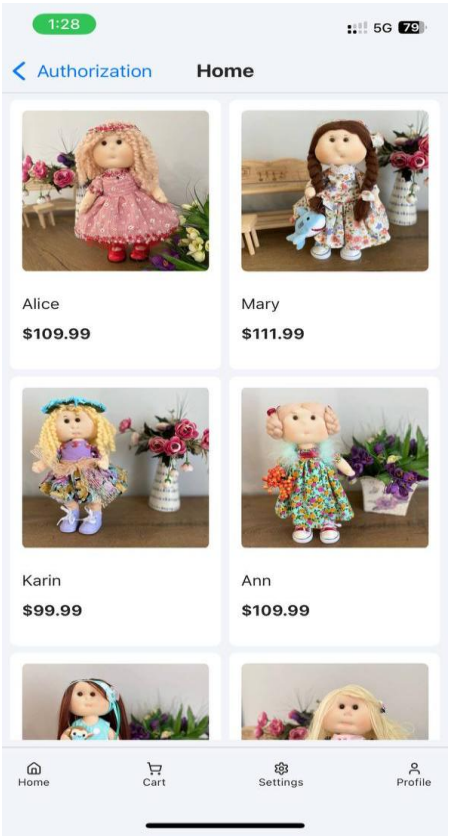


Рисунок 3.4.6 – Успішна реєстрація

Користувач може перейти до єкрану налаштувань, щоб змінити тему на темну, це значно збільшує користувацький досвід(рис. 3.4.7):

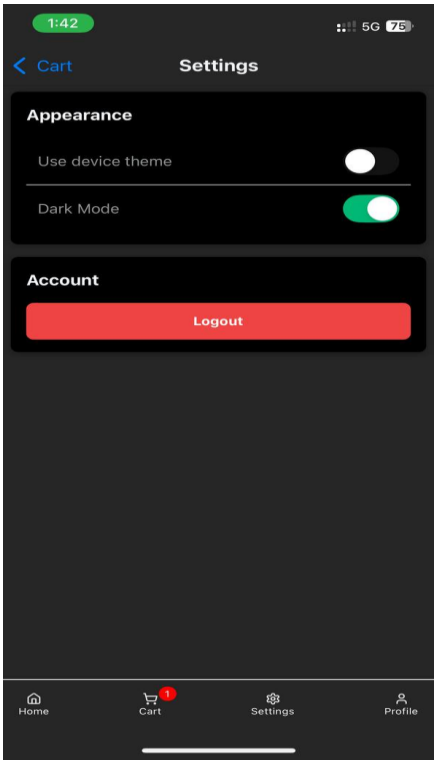


Рисунок 3.4.7 – Єкран налаштувань та зміна теми

Також є можливість перейти на сторінку обраної ляльки (рис. 3.4.8):

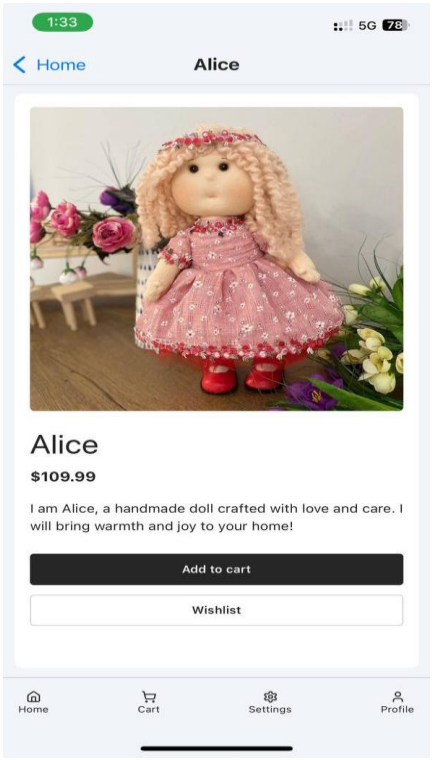


Рисунок 3.4.8 – Сторінка обраної ляльки

На цьому єкрані користувач може додати обрану ляльку до кошика, а далі натиснути на меню кошика, щоб побачити його вміст (рис.3.4.9):

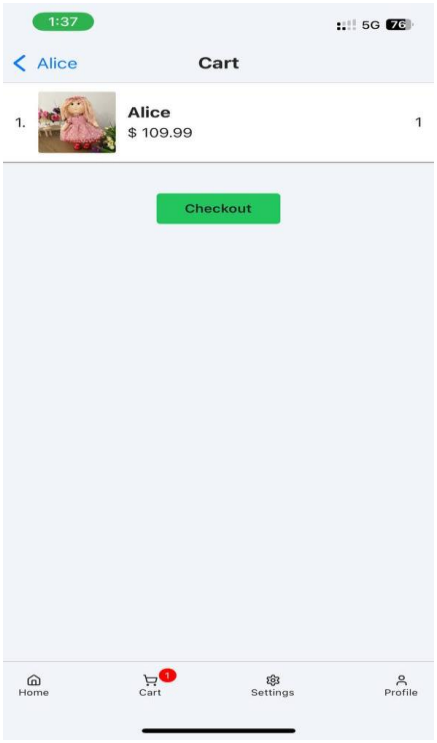


Рисунок 3.4.9 – Єкран кошику

Далі користувач натискає кнопку Checkout і перед ним з’являється форма оплати замовлення, яку користувач заповнює (рис. 3.4.10) та натискає на кнопку оплати (рис. 3.4.11):

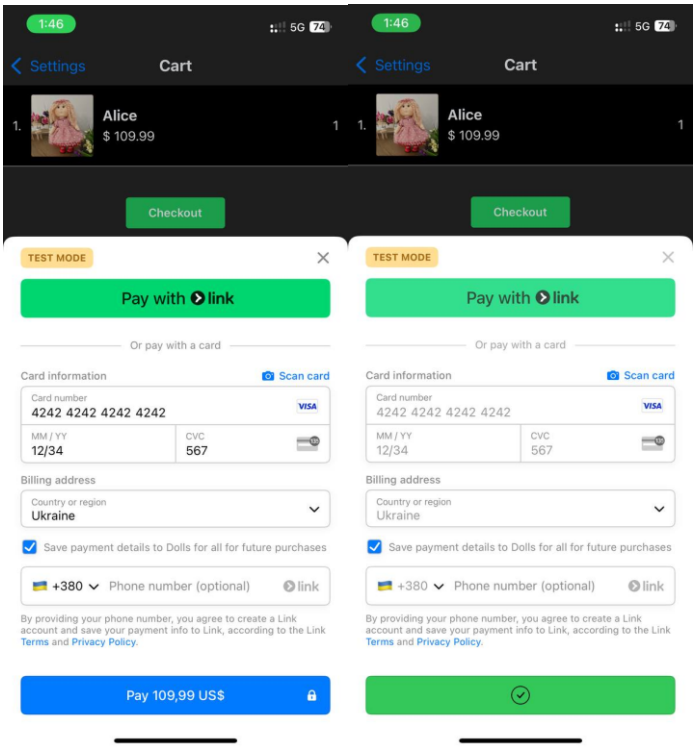


Рисунок 3.4.10, 3.4.11 – Форма оплати

Далі користувач отримує повідомлення про те, що транзакція була виконана успішно і повертається на головний єкран (рис. 3.4.12):

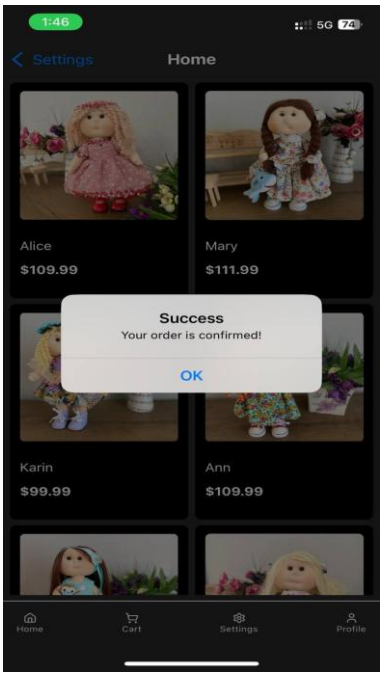
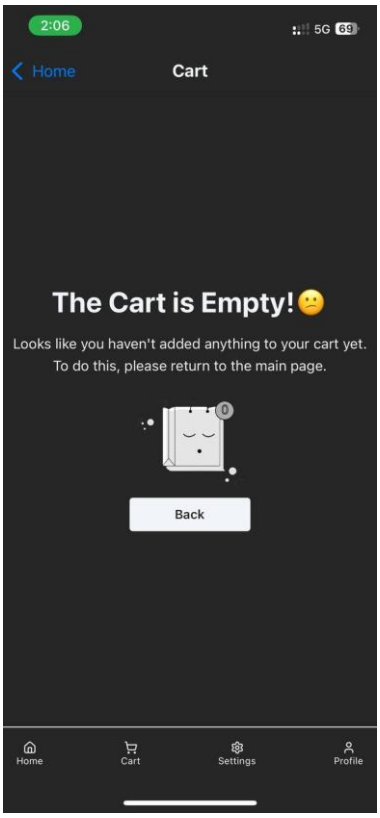


Рисунок 3.4.12 - Успішна транзакція

Кошик тепер порожній (рис. 3.4.13):



Для перевірки замовлення, користувач може перейти на єкран профілю, де побачить інформацію про себе та історію замовлень (рис. 3.4.14):

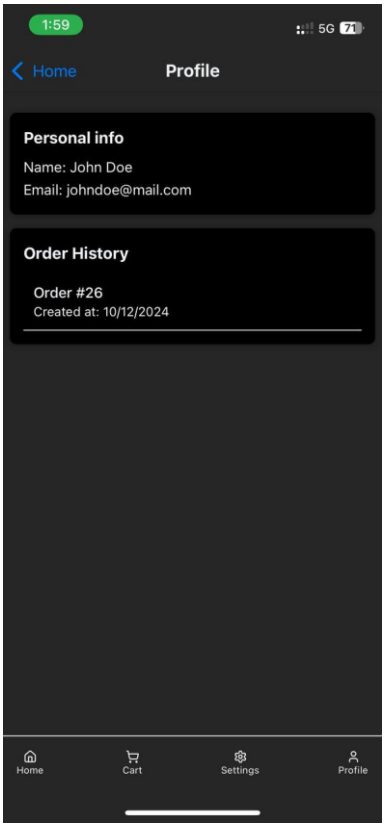


Рисунок 3.4.14 – Єкран профілю користувача

І далі натиснути на своє замовлення, що б переглянути його деталі (рис. 3.4.15):

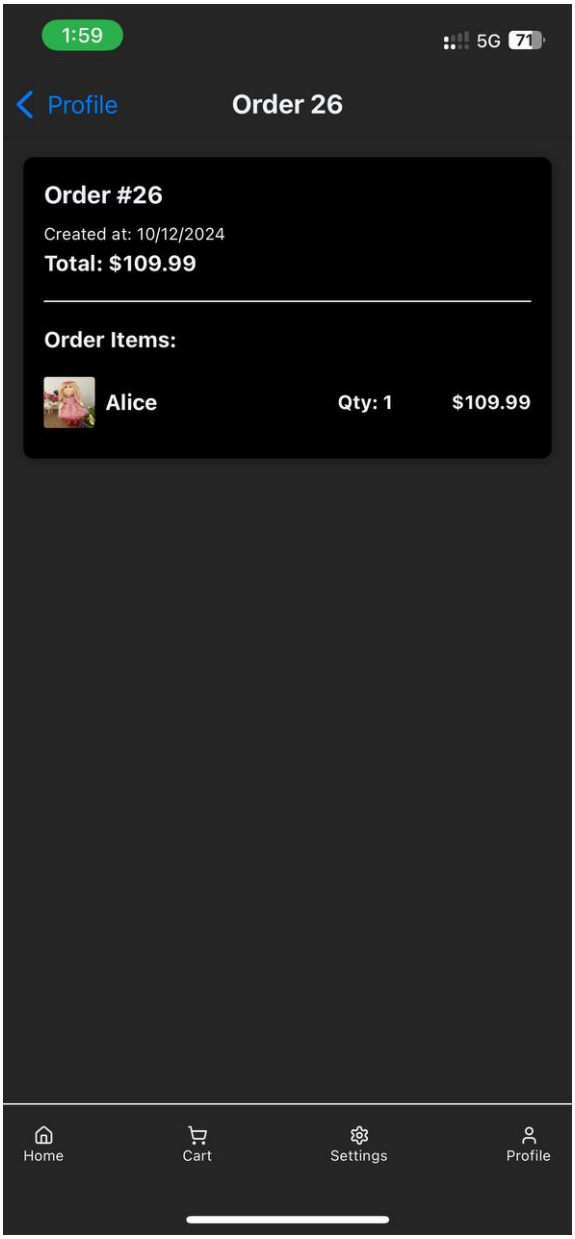


Рисунок 3.4.15 – Інформація про замовлення

Таким чином було проведено тестування додатку, де було перевірено, що весь наведений функціонал працює чітко та швидко.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено інформаційну технологію створення мобільного додатку для продажу ляльок ручної роботи, який побудований з використанням сучасних технологій, таких як React Native, Expo та PostgreSQL. Обрані технології забезпечили зручність розробки, високу продуктивність та адаптивність для мобільних платформ, що дозволяє додатку працювати на різних пристроях із системами iOS та Android.

React Native в поєднанні з Expo надали можливість створити зручний та інтерактивний інтерфейс користувача, що забезпечує легку взаємодію з додатком. База даних PostgreSQL стала основою для зберігання структурованих даних, що забезпечує надійність і масштабованість системи. Інтеграція з платіжними сервісами, такими як Stripe, дозволила реалізувати зручний і безпечний механізм оплати.

Система демонструє високу продуктивність завдяки оптимізації запитів до бази даних і використанню ефективних підходів до управління станом мобільного додатку. Особливу увагу було приділено питанням безпеки: реалізовано надійну аутентифікацію та авторизацію користувачів, захищено дані за допомогою шифрування, а також передбачено заходи протидії мережевим загрозам.

Результатом роботи став повнофункціональний мобільний додаток, який відповідає сучасним стандартам розробки програмного забезпечення для електронної комерції. Розроблений додаток є ефективним інструментом для майстрів ручної роботи, сприяючи розширенню їхньої аудиторії, автоматизації процесів продажу та покращенню взаємодії з клієнтами.

У перспективі додаток може бути вдосконалений шляхом впровадження функцій зміни мови, персоніфікація рекомендацій, система знижок та доставка по всьому світу, потенціальне створення адмін дошки для керування замовленнями та редагування списку товарів, а також покращення дизайну для підвищення користувацького досвіду.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Буч Г. Об'єктно-орієнтований аналіз та проектування з прикладами застосування. – М.: Вільямс, 2007. – 560 с.
2. Фаулер М. Архітектура корпоративних програмних застосунків. – К.: Діалектика, 2006. – 352 с.
3. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Патерни проектування: повторно використовувані рішення об'єктно-орієнтованих проблем. – СПб.: Пітер, 2010. – 368 с.
4. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002. – 560 p..
5. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1994. – 395 p.
6. Freeman E., Freeman E. Head First Design Patterns: Building Extensible, Maintainable Object-Oriented Software. – O'Reilly Media, 2004. – 694 p.
7. Ічанська Н.В., Улько С.І. Основні аспекти створення мобільних додатків та вибір інструментів їх розробки // Системи управління, навігації та зв'язку. – 2020. – Вип. 1(59). – С. 74–78.
8. Дєдух Т.А. Методи розробки мобільних додатків // Матеріали XXIII Всеукраїнської науково-технічної конференції. – 2023.
9. Люлька Р.О. Методології розробки мобільних додатків для медичних інформаційних систем // Вісник НТУУ 'КПІ'. – 2022.
10. Zhang X., Lu Y., Wang M. Development of Mobile Application for E-Commerce using React Native and Firebase // Journal of Mobile Development Research. – 2022. – Vol. 10(2). – P. 15–28.
11. Chavan V., Mahajan A. Comparative Analysis of Client-Server vs. Peer-to-Peer Architecture in Modern Applications // International Journal of Computer Applications. – 2021. – Vol. 176(1). – P. 32–37.
12. Agarwal S., Jain N. Microservices vs. Monolith: A Comparative Study on Software Architecture for Scalable Applications // IEEE Access. – 2022. – Vol. 10. – P. 1234–1245.
13. Brown K., White J. Cloud-Native Applications: Patterns for Distributed Systems and Scalable Architectures // Journal of Cloud Engineering. – 2021. – Vol. 8(3). – P. 45–67.
14. Smith R., Johnson K. Security Challenges in Stripe Payment Gateway Integration // Journal of Secure Payments. – 2023. – Vol. 6(2). – P. 78–89.
15. Patel P., Desai D. PostgreSQL vs. MongoDB: A Comparative Performance Analysis for E-Commerce Databases // International Journal of Database Management. – 2022. – Vol. 18(4). – P. 66–79.
16. Singh R., Kumar V. Implementation of Payment Gateways in Mobile Applications: A Review of Best Practices // Journal of Mobile Application Development. – 2023. – Vol. 11(1). – P. 58–72.
17. React Native Documentation. [Електронний ресурс]. – URL: https://reactnative.dev/docs/getting-started. – Дата звернення: 10.12.2024.

18. Expo Documentation. [Електронний ресурс]. – URL: https://docs.expo.dev/. – Дата звернення: 10.12.2024.
19. Zustand Documentation. [Електронний ресурс]. – URL: https://docs.pmnd.rs/zustand/getting-started/introduction. – Дата звернення: 10.12.2024.
20. PostgreSQL Documentation. [Електронний ресурс]. – URL: https://www.postgresql.org/docs/. – Дата звернення: 10.12.2024.
21. Stripe API Reference. [Електронний ресурс]. – URL: https://stripe.com/docs/api. – Дата звернення: 10.12.2024.
22. Drizzle ORM Documentation. [Електронний ресурс]. – URL: https://orm.drizzle.team/. – Дата звернення: 10.12.2024.
23. Як електронна комерція змінює ринок: тенденції 2024. [Електронний ресурс]. – URL: https://cpashka.biz/blog/iak-elektronna-komertsiia-zminiuie-rynok-tendentsii-2024/. – Дата звернення: 10.12.2024.
24. Тренди та виклики українського ринку eCommerce у 2024 році. [Електронний ресурс]. – URL: https://uaateam.agency/blog/trendy-ta-vyklyky-ukrainskogo-rynku-ecommerce/. – Дата звернення: 10.12.2024.
25. Глобальна електронна комерція: ключові цифри та тренди e-commerce 2024. [Електронний ресурс]. – URL: https://rau.ua/novyni/trendi-e-com-2024/. – Дата звернення: 10.12.2024.
26. Що чекає на український e-commerce у 2024 році: розбираємо ключові тренди. [Електронний ресурс]. – URL: https://ua-retail.com/2023/11/shho-chekaye-na-ukrainskij-e-commerce-u-2024-roci-rozbirayemo-klyuchovi-trendi/. – Дата звернення: 10.12.2024.
27. Martin R.C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall, 2017. – 432 p.
28. Stevens W., Myers G., Constantine L. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design. – Yourdon Press, 1979. – 400 p.
29. Tanenbaum A.S., Van Steen M. Distributed Systems: Principles and Paradigms. – Pearson Education, 2007. – 704 p.
30. Bass L., Clements P., Kazman R. Software Architecture in Practice. – Addison-Wesley, 2012. – 640 p.

ДОДАТОК А

(A1) // back-end *src/index.ts*

```
import express, { json, urlencoded, Request } from 'express';
import dollsRoutes from './routes/dolls/index.js';
import authRoutes from './routes/auth/index.js';
import ordersRoutes from './routes/orders/index.js';
import stripeRoutes from './routes/stripe/index.js';

import serverless from 'serverless-http';  18k (gzipped: 5k)

const port = 3001;
const app = express();

app.use(urlencoded({ extended: false }));
app.use(
  json({
    Tabnine | Edit | Test | Explain | Document | Ask
    verify: (req: Request, res, buf) => {
      req.rawBody = buf;
    },
  })
);      You, 2 weeks ago • expo router setup
Tabnine | Edit | Test | Explain | Document | Ask
app.get('/', (req, res) => {
  res.send('Hello World!');
});
app.use('/dolls', dollsRoutes);
app.use('/auth', authRoutes);
app.use('/orders', ordersRoutes);
app.use('/stripe', stripeRoutes);
if (process.env.NODE_ENV === 'dev') {
  Tabnine | Edit | Test | Explain | Document | Ask
  app.listen(port, () => {
    console.log(`Example app listening on port ${port}`);
  });
}

export const handler = serverless(app);
```

(A2) *src/db/dollsSchema.ts*

```ts
export const dollsTable = pgTable('dolls', {
  id: integer().primaryKey().generatedAlwaysAsIdentity(),
  dollName: varchar({ length: 255 }).notNull(),
  description: text().notNull(),
  viewsCount: integer().default(0),
  image: varchar({ length: 255 }).notNull(),
  price: doublePrecision().notNull(),
  createdAt: timestamp().defaultNow().notNull(), // Автоматично додається при створенні
  updatedAt: timestamp().defaultNow().notNull(), // Автоматично оновлюється
});       You, last week • backend deployed to genezio successfully

export const createDollSchema = createInsertSchema(dollsTable).omit({
  id: true,
  viewsCount: true,
  createdAt: true,
  updatedAt: true,
});

export const updateDollSchema = createInsertSchema(dollsTable)
  .omit({
    id: true,
    viewsCount: true,
    createdAt: true,
    updatedAt: true,
  })
  .partial();
```

(А3) ***src/db/usersSchema.ts***

```ts
export const usersTable = pgTable('users', {
  id: integer().primaryKey().generatedAlwaysAsIdentity(),

  email: varchar({ length: 255 }).notNull().unique(),
  password: varchar({ length: 255 }).notNull(),
  role: varchar({ length: 255 }).notNull().default('user'),

  name: varchar({ length: 255 }).notNull(),
  address: text(),
  avatarUrl: varchar({ length: 255 }),       You, last week • backend deployed
  createdAt: timestamp().defaultNow().notNull(), // Автоматично додається при створенні
  updatedAt: timestamp().defaultNow().notNull(), // Автоматично оновлюється
});

export const createUserSchema = createInsertSchema(usersTable).omit({
  id: true,
  role: true,
});

export const loginSchema = createInsertSchema(usersTable).pick({
  email: true,
  password: true,
});
```

(A4) ***src/db/ordersSchema.ts***

```ts
export const ordersTable = pgTable('orders', {
  id: integer().primaryKey().generatedAlwaysAsIdentity(),
  createdAt: timestamp().notNull().defaultNow(),
  status: varchar({ length: 50 }).notNull().default('New'),
  userId: integer()
    .references(() => usersTable.id)
    .notNull(),
  stripePaymentIntentId: varchar({ length: 255 }),
});

export const orderItemsTable = pgTable('order_items', {
  id: integer().primaryKey().generatedAlwaysAsIdentity(),
  orderId: integer()
    .references(() => ordersTable.id)
    .notNull(),
  dollId: integer()
    .references(() => dollsTable.id)
    .notNull(),
  quantity: integer().notNull(),
  price: doublePrecision().notNull(),
});
```

(A5) ***src/middlewares/authMiddleware.ts***

```ts
export function verifyToken(req: Request, res: Response, next: NextFunction) {
  const token = req.header('Authorization');

  if (!token) {
    res.status(401).json({ error: 'Access denied' });
    return;
  }

  try {
    // decode jwt toke data
    const decoded = jwt.verify(token, process.env.JWT_SECRET as string);
    if (typeof decoded !== 'object' || !decoded?.userId) {
      res.status(401).json({ error: 'Access denied' });
      return;
    }
    req.userId = decoded.userId;
    req.role = decoded.role;
    next();
  } catch (e) {
    res.status(401).json({ error: 'Access denied' });
  }
}

Tabnine | Edit | Test | Explain | Document | Ask
export function verifySeller(req: Request, res: Response, next: NextFunction) {
  const role = req.role;
  if (role !== 'seller') {
    res.status(401).json({ error: 'Access denied' });
    return;
  }
  next();
}
```

(A6) ***src/middlewares/validationMiddleware.ts***

```
export function validateData(schema: z.ZodObject<any, any>) {
  return (req: Request, res: Response, next: NextFunction) => {
    try {
      schema.parse(req.body);
      req.cleanBody = _.pick(req.body, Object.keys(schema.shape));
      next();
    } catch (error) {
      if (error instanceof ZodError) {
        const errorMessages = error.errors.map((issue: any) => ({
          message: `${issue.path.join('.')} is ${issue.message}`,
        }));
        res.status(400).json({ error: 'Invalid data', details: errorMessages });
      } else {      You, 2 weeks ago • expo router setup
        res.status(500).json({ error: 'Internal Server Error' });
      }
    }
  };
}
```

(A7) ***src/routes/auth/index.ts***

```
const generateUserToken = (user: any) => {
  return jwt.sign(
    { userId: user.id, role: user.role },
    process.env.JWT_SECRET as string,
    {
      expiresIn: '30d',
    }
  );
};

Tabnine | Edit | Test | Explain | Document | Ask
router.post('/register', validateData(createUserSchema), async (req, res) => {
  try {
    const data = req.cleanBody;
    data.password = await bcrypt.hash(data.password, 10);

    const [user] = await db.insert(usersTable).values(data).returning();

    // @ts-ignore
    delete user.password;
    const token = generateUserToken(user);

    res.status(201).json({ user, token });
  } catch (e) {
    console.log(e);      You, 2 weeks ago • expo router setup
    res.status(500).send('Something went wrong');
  }
});
```

```
router.post('/login', validateData(loginSchema), async (req, res) => {
  try {
    const { email, password } = req.cleanBody;

    const [user] = await db
      .select()
      .from(usersTable)
      .where(eq(usersTable.email, email));
    if (!user) {
      res.status(401).json({ error: 'Authentication failed' });
      return;
    }

    const matched = await bcrypt.compare(password, user.password);
    if (!matched) {
      res.status(401).json({ error: 'Authentication failed' });
      return;
    }

    // create a jwt token
    const token = generateUserToken(user);
    // @ts-ignore
    delete user.password;
    res.status(200).json({ token, user });
  } catch (e) {
    res.status(500).send('Something went wrong');
  }
});
```

(A8) ***src/routes/dolls/index.ts***

```
router.get('/', listDolls);       You, last week • backend depl
router.get('/:id', getDollById);
router.post(
  '/',
  verifyToken,
  verifySeller,
  validateData(createDollSchema),
  createDoll
);
router.put(
  '/:id',
  verifyToken,
  verifySeller,
  validateData(updateDollSchema),
  updateDoll
);
router.delete('/:id', verifyToken, verifySeller, deleteDoll);

export default router;
```

(A9) ***src/routes/dolls/dollsController.ts***

```
export async function listDolls(req: Request, res: Response) {
  try {
    const dolls = await db.select().from(dollsTable);
    res.json(dolls);
  } catch (e) {
    console.log(e);
    res.status(500).send(e);       You, last week • backend deploy
  }
}

Tabnine | Edit | Test | Explain | Document | Ask
export async function getDollById(req: Request, res: Response) {
  try {
    const { id } = req.params;
    const [doll] = await db
      .select()
      .from(dollsTable)
      .where(eq(dollsTable.id, Number(id)));

    if (!doll) {
      res.status(404).send({ message: 'Doll not found' });
    } else {
      res.json(doll);
    }
  } catch (e) {
    res.status(500).send(e);
  }
}
```

```
export async function createDoll(req: Request, res: Response) {
  try {
    const [doll] = await db
      .insert(dollsTable)
      .values(req.cleanBody)
      .returning();
    res.status(201).json(doll);
  } catch (e) {
    res.status(500).send(e);
  }
}

Tabnine | Edit | Test | Explain | Document | Ask
export async function updateDoll(req: Request, res: Response) {
  try {
    const id = Number(req.params.id);
    const updatedFields = req.cleanBody;

    const [doll] = await db
      .update(dollsTable)
      .set(updatedFields)
      .where(eq(dollsTable.id, id))
      .returning();

    if (doll) {
      res.json(doll);
    } else {
      res.status(404).send({ message: 'Doll was not found' });
    }
  } catch (e) {
    res.status(500).send(e);
  }
}
```

```ts
export async function deleteDoll(req: Request, res: Response) {
  try {
    const id = Number(req.params.id);
    const [deletedDoll] = await db
      .delete(dollsTable)
      .where(eq(dollsTable.id, id))
      .returning();
    if (deletedDoll) {
      res.status(204).send();
    } else {
      res.status(404).send({ message: 'Doll was not found' });
    }
  } catch (e) {
    res.status(500).send(e);
  }
}
```

(A10) ***src/routes/orders/index.ts***

```ts
router.post(
  '/',
  verifyToken,
  validateData(insertOrderWithItemsSchema),
  createOrder
);

router.get('/', verifyToken, listOrders);
router.get('/:id', verifyToken, getOrder);
router.put('/:id', verifyToken, validateData(updateOrderSchema), updateOrder);
```

(A11) ***src/routes/orders/ordersController.ts***

```ts
export async function createOrder(req: Request, res: Response) {
  try {
    const { order, items } = insertOrderWithItemsSchema.parse(req.cleanBody);

    const userId = req.userId;

    if (!userId) {      You, last week • Orders CRUD done
      res.status(400).json({ message: 'Invalid order data' });
    }

    const [newOrder] = await db
      .insert(ordersTable)
      // @ts-ignore
      .values({ userId: userId })
      .returning();

    const orderItems = items.map((item: any) => ({
      ...item,
      orderId: newOrder.id,
    }));
    const newOrderItems = await db
      .insert(orderItemsTable)
      .values(orderItems)
      .returning();

    res.status(201).json({ ...newOrder, items: newOrderItems });
  } catch (e) {
    console.log(e);
    res.status(400).json({ message: 'Invalid order data' });
  }
}
```

```ts
export async function listOrders(req: Request, res: Response) {
  try {
    const role = req.role;
    const userId = req.userId;

    let query = db.select().from(ordersTable);

    if (role === 'user') {
      //@ts-ignore
      query = query.where(eq(ordersTable.userId, userId));
    } else if (role === 'seller') {
      //@ts-ignore
      query = query
        .leftJoin(orderItemsTable, eq(ordersTable.id, orderItemsTable.orderId))
        //@ts-ignore
        .where(eq(orderItemsTable.sellerId, userId));
    }

    const orders = await query;
    res.json(orders);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Failed to fetch orders', error });
  }
}
```

```ts
export async function getOrder(req: Request, res: Response) {
  try {
    const id = parseInt(req.params.id);
      You, last week • Orders CRUD done
    const orderWithItems = await db
      .select()
      .from(ordersTable)
      .where(eq(ordersTable.id, id))
      .leftJoin(orderItemsTable, eq(ordersTable.id, orderItemsTable.orderId));

    if (orderWithItems.length === 0) {
      res.status(404).send('Order not found');
    }

    const mergedOrder = {
      ...orderWithItems[0].orders,
      items: orderWithItems.map((oi) => oi.order_items),
    };

    res.status(200).json(mergedOrder);
  } catch (error) {
    console.log(error);
    res.status(500).send(error);
  }
}
```

```
export async function updateOrder(req: Request, res: Response) {
  try {
    const id = parseInt(req.params.id);

    const [updatedOrder] = await db
      .update(ordersTable)
      .set(req.body)
      .where(eq(ordersTable.id, id))
      .returning();

    if (!updatedOrder) {
      res.status(404).send('Order not found');
    } else {
      res.status(200).json(updatedOrder);
    }
  } catch (error) {
    res.status(500).send(error);
  }
}
```

(A12) ***src/routes/stripe/index.ts***

```
const router = Router();

router.get('/keys', getKeys);

router.post('/payment-intent', verifyToken, createPaymentIntent);

router.post('/webhook', webhook);       You, yesterday • Payment f

export default router;
```

(A13) ***src/routes/stripe/stripeController.ts***

```
const stripe = new Stripe(process.env.STRIPE_SECRET_KEY!);
const endpointSecret = process.env.STRIPE_ENDPOINT_SECRET!;       You, yes

Tabnine | Edit | Test | Explain | Document | Ask
export async function getKeys(req: Request, res: Response) {
  res.json({ publishableKey: process.env.STRIPE_PUBLISHABLE_KEY });
}

Tabnine | Edit | Test | Explain | Document | Ask
export async function createPaymentIntent(req: Request, res: Response) {
  try {
    const { orderId } = req.body;

    const [order] = await db
      .select()
      .from(ordersTable)
      .where(eq(ordersTable.id, orderId));

    const orderItems = await db
      .select()
      .from(orderItemsTable)
      .where(eq(orderItemsTable.orderId, orderId));

    const total = orderItems.reduce(
      (sum, item) => sum + item.price * item.quantity,
      0
    );
```

```
    const amount = Math.floor(total * 100);
    if (amount === 0) {
      res.status(400).json({ message: 'Order total is 0' });
      return;
    }
    const [user] = await db
      .select()
      .from(usersTable)
      .where(eq(usersTable.id, order.userId));

    // TODO: Add info about the user
    const customer = await stripe.customers.create({
      email: user.email,
      name: user.name,
      metadata: {
        userId: user?.id,
        orderId: order.id,
      },
    });

    const ephemeralKey = await stripe.ephemeralKeys.create(
      { customer: customer.id },
      { apiVersion: '2024-11-20.acacia' }
    );

    // TODO: calculate the amount dynamically
    const paymentIntent = await stripe.paymentIntents.create({
      amount,
      currency: 'usd',
      customer: customer.id,
      description: `Order ID: ${order.id}`,
    });
```

```
    await db
      .update(ordersTable)
      .set({ stripePaymentIntentId: paymentIntent.id })
      .where(eq(ordersTable.id, orderId));

    res.json({
      paymentIntent: paymentIntent.client_secret,
      ephemeralKey: ephemeralKey.secret,
      customer: customer.id,
      publishableKey: process.env.STRIPE_PUBLISHABLE_KEY,
    });
  } catch (error) {
    console.error('Error creating Payment Intent:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
}
```

```
export async function webhook(req: Request, res: Response) {
  const sig = req.headers['stripe-signature'];
  console.log(sig);

  let event;

  try {
    event = stripe.webhooks.constructEvent(req.rawBody!, sig!, endpointSecret);
    console.log(event);
  } catch (err) {
    res.status(400).send(`Webhook Error: ${(err as Error).message}`);
    return;
  }

  // Handle the event
  switch (event.type) {
    case 'payment_intent.succeeded':
      const paymentIntent = event.data.object;
      await db
        .update(ordersTable)
        .set({ status: 'payed' })
        .where(eq(ordersTable.stripePaymentIntentId, paymentIntent.id));
      break;
    case 'payment_intent.payment_failed':
      const paymentIntentFailed = event.data.object;
      await db
        .update(ordersTable)
        .set({ status: 'payment_failed' })
        .where(eq(ordersTable.stripePaymentIntentId, paymentIntentFailed.id));
      break;
```

```
    case 'payment_method.attached':
      const paymentMethod = event.data.object;
      // Then define and call a method to handle the successful attachment of a
      PaymentMethod.
      // handlePaymentMethodAttached(paymentMethod);      You, yesterday • Paym
      break;
    // ... handle other event types
    default:
      console.log(`Unhandled event type ${event.type}`);
  }

  res.json({ received: true });
}
```

ДОДАТОК Б

(Б1) //frontend *src/api/auth.ts*

```ts
export const login = async (email: string, password: string): Promise<any> => {
  try {
    const response = await axios.post(`${API_URL}/auth/login`, {
      email,
      password,
    });

    return response.data;
  } catch (error: any) {
    console.error('Login error:', error);
    throw new Error(error.response?.data?.message || 'Login failed');
  }
};
export const signup = async (
  name: string,
  email: string,
  password: string
): Promise<any> => {
  try {
    const response = await axios.post(`${API_URL}/auth/register`, {
      name,
      email,
      password,
    });

    return response.data;
  } catch (error: any) {
    console.error('Registration error:', error);
    throw new Error(error.response?.data?.message || 'Registration failed');
  }
};
      You, 22 hours ago • Project done
export const logout = async (logoutFn: () => void) => {
  logoutFn();
  await AsyncStorage.clear();
};
```

(Б2) *src/api/auth.ts*

```
export const listDolls = async () => {
  try {
    const { data } = await axios.get(`${API_URL}/dolls`);
    return data;
  } catch (error) {
    if (axios.isAxiosError(error)) {
      console.error(`Axios error: ${error.response?.status}`);
      throw new Error(
        error.response?.data?.message || 'Unknown error occurred'
      );
    } else {
      console.error('Unexpected error', error);
      throw new Error('Unexpected error');
    }
  }
};

export const fetchDoll = async (id: number) => {
  try {
    const { data } = await axios.get(`${API_URL}/dolls/${id}`);
    return data;
  } catch (error) {
    if (axios.isAxiosError(error)) {
      console.error(`Axios error: ${error.response?.status}`);
      throw new Error(
        error.response?.data?.message || 'Unknown error occurred'
      );
    } else {
      console.error('Unexpected error', error);
      throw new Error('Unexpected error');
    }
  }
};
```

(Б3) *src/api/auth.ts*

```
export async function createOrder(items: any[]) {
  try {
    // @ts-ignore
    const token = useAuth.getState().token;

    const response = await axios.post(
      `${API_URL}/orders`,
      { order: {}, items },
      {
        headers: {
          'Content-Type': 'application/json',
          Authorization: `${token}`,
        },
      }
    );

    return response.data;
  } catch (error: any) {
    console.error(
      'Error creating order:',
      error.response?.data || error.message
    );
    throw new Error(error.response?.data?.message || 'Error creating order');
  }
}
```

```ts
export const getOrders = async () => {
  // @ts-ignore
  const token = useAuth.getState().token;

  const res = await axios.get(`${API_URL}/orders`, {
    headers: {
      'Content-Type': 'application/json',
      Authorization: `${token}`,
    },
  });

  if (!res.status.toString().startsWith('2')) {
    throw new Error('Failed to fetch orders');
  }

  return res.data;
};
```

```ts
export async function getOrder(orderId: number) {
  try {
    // @ts-ignore
    const token = useAuth.getState().token;

    const { data: order } = await axios.get(`${API_URL}/orders/${orderId}`, {
      headers: {
        'Content-Type': 'application/json',
        Authorization: `${token}`,
      },
    });
    const itemsWithDollDetails = await Promise.all(
      order.items.map(async (item: any) => {
        const doll = await fetchDoll(item.dollId);

        return {
          ...item,
          dollName: doll.dollName,
          dollImage: doll.image,
        };
      })
    );
    const totalAmount = itemsWithDollDetails.reduce(
      (total: number, item: any) => total + item.price,
      0
    );
    return {
      ...order,
      items: itemsWithDollDetails,
      totalAmount,
    };
  } catch (error) {
    console.error('Failed to fetch order', error);
    throw new Error('Failed to fetch order');
  }
}
```

(Б4) ***src/api/stripe.ts***

```
export const fetchStripeKeys = async () => {
  try {
    const { data } = await axios.get(`${API_URL}/stripe/keys`);

    return data;
  } catch (error) {
    if (axios.isAxiosError(error)) {
      console.error(`Axios error: ${error.response?.status}`);
      throw new Error(
        error.response?.data?.message || 'Unknown error occurred'
      );
    } else {
      console.error('Unexpected error', error);
      throw new Error('Unexpected error');
    }
  }
};
```

```
export const createPaymentIntent = async ({ orderId }: { orderId: string }) => {
  // @ts-ignore
  const token = useAuth.getState().token;
  try {
    console.log('Token:', token);

    const { data } = await axios.post(
      `${API_URL}/stripe/payment-intent`,

      { orderId },
      {
        headers: {
          'Content-Type': 'application/json',
          Authorization: token,
        },
      }
    );

    return data;
  } catch (error) {
    if (axios.isAxiosError(error)) {
      console.error(`Axios error: ${error.response?.status}`);
      throw new Error(
        error.response?.data?.message || 'Unknown error occurred'
      );
    } else {
      console.error('Unexpected error', error);
      throw new Error('Unexpected error');
    }
  }
};
```

(Б5) *RootLayout*

```tsx
export default function RootLayout() {
  const [isDarkMode, setIsDarkMode] = useState(false);
  const [useDeviceSettings, setUseDeviceSettings] = useState(false);
  const scheme = useColorScheme();

  useEffect(() => {
    let subscription: any;

    if (useDeviceSettings) {
      subscription = Appearance.addChangeListener((scheme) => {
        setIsDarkMode(scheme.colorScheme === 'dark');
      });
    }

    return () => {
      if (subscription) {
        subscription.remove();
        subscription = null;
      }
    };
  }, [scheme, isDarkMode, useDeviceSettings]);

  return (
    <QueryClientProvider client={queryClient}>
      <CustomStripeProvider>
        <DarkMode.Provider
          value={{
            isDarkMode,
            setIsDarkMode,
            useDeviceSettings,
            setUseDeviceSettings,
          }}
        >
          <GluestackUIProvider>
            <StatusBar style={isDarkMode ? 'light' : 'dark'} />

            <SafeAreaView
              className={`flex-1 ${isDarkMode ? 'bg-[#262626]' : 'bg-[#f1f5f9]
              '}`}
            >
              <Layout />
            </SafeAreaView>
          </GluestackUIProvider>
        </DarkMode.Provider>
      </CustomStripeProvider>
    </QueryClientProvider>
  );
}
```

(Б6) *HomeScreen*

```
export default function HomeScreen() {
  const { data, isLoading, error } = useQuery({
    queryKey: ['dolls'],
    queryFn: listDolls,
  });
  const { isDarkMode } = useContext(DarkMode);
  const numColumns = useBreakpointValue({ default: 2, sm: 3, xl: 4 });
  if (isLoading) {···
  }
  if (error) {···
  }
  return (
    <Box className={`p-2 ${isDarkMode ? '□bg-[#262626]' : '■bg-[#f1f5f9]'}`}>
      <Stack.Screen options={{ title: 'Home' }} />
      <FlatList
        data={data}
        numColumns={numColumns}
        keyExtractor={(item) => item.id.toString()}
        className='gap-2 mx-auto w-full max-w-[960px]'
        contentContainerClassName='gap-2'
        columnWrapperClassName='gap-2'
        renderItem={({ item }) => <ProductListItem doll={item} />}
      />
    </Box>
  );      You, last week • Tanstack query and api integration
}
```

(Б7) *LoginScreen (Fragment)*

```
export default function LoginScreen() {
  return (
    <SafeAreaView
      className={`flex-1 justify-center ${isDarkMode ? '□bg-[#262626]' : '■bg-
      [#f1f5f9]'}`}
    >
      <Stack.Screen options={{ title: 'Authorization' }} />
      <FormControl
        className={`px-4 py-8 border rounded-lg max-w-[500px]
        border-outline-300 m-2 ${isDarkMode ? '□bg-[#000]' : '■bg-[#fff]'}`}
      >
        <VStack space='xl'>
          <Heading
            className={`text-center text-typography-900 text-2xl leading-3 pt-3
            ${isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]'}`}
          >
            Login
          </Heading>

          <VStack space='xs'>
            <Text
              className={`text-typography-500 leading-1 ${isDarkMode ? '■text-
              [#f1f5f9]' : '□text-[#262626]'}`}
            >
              Email
            </Text>
            <Input>
              <InputField
                className={` ${isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]
                '}`}
                type='text'
                placeholder='Enter your email'
                value={email}
                onChangeText={(text) => setEmail(text)}
              />
```

```
            {errors.email && (
              <Text className='text-red-500 text-sm'>{errors.email}</Text>
            )}
          </VStack>

          <VStack space='xs'>
            <Text
              className={`text-typography-500 leading-1 ${isDarkMode ? 'text-
              [#f1f5f9]' : 'text-[#262626]'}`}
            >
              Password
            </Text>
            <Input>
              <InputField
                className={` ${isDarkMode ? 'text-[#f1f5f9]' : 'text-[#262626]
                '}`}
                type={showPassword ? 'text' : 'password'}
                placeholder='Enter your password'
                value={password}
                onChangeText={(text) => setPassword(text)}
              />
              <InputSlot className='pr-3' onPress={togglePasswordVisibility}>
                <InputIcon
                  as={showPassword ? EyeIcon : EyeOffIcon}
                  className='text-darkBlue-500'
                />
              </InputSlot>
            </Input>
            {errors.password && (
              <Text className='text-red-500 text-sm'>{errors.password}</Text>
            )}
          </VStack>

          {errors.form && (
            <Text className='text-red-500 text-center text-sm'>
```

(Б8) *RegisterScreeen(Fragment)*

```
export default function RegisterScreen() {
  const { isDarkMode } = useContext(DarkMode);

  const setUser = useAuth((s: any) => s.setUser);
  const setToken = useAuth((s: any) => s.setToken);
  const isLoggedIn = useAuth((s: any) => !!s.token);

  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [showPassword, setShowPassword] = useState(false);

  const [errors, setErrors] = useState({ ···
  });

  const [loading, setLoading] = useState(false); // Track loading state

  const signupMutation = useMutation({ ···
  });

  const validate = () => { ···
  };

  const handleRegister = () => { ···
  };

  const togglePasswordVisibility = () => setShowPassword((prev) => !prev);

  if (isLoggedIn) { ···
  }

  return (
    <SafeAreaView
      className={`flex-1 justify-center ${isDarkMode ? 'bg-[#262626]' : 'bg-
      [#f1f5f9]'}`}
```

```
    >
      <Stack.Screen options={{ title: 'Registration' }} />
      <FormControl
        className={`px-4 py-8 border rounded-lg max-w-[500px]
        border-outline-300 m-2 ${isDarkMode ? '□bg-[#000]' : '■bg-[#fff]'}`}
      >
        <VStack space='xl'>
          <Heading
            className={`text-center text-typography-900 text-2xl leading-3 pt-3
            ${isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]'}`}
          >
            Sign up
          </Heading>

          <VStack space='xs'>      You, 23 hours ago • Project done
            <Text
              className={`text-typography-500 leading-1 ${isDarkMode ? '■text-
              [#f1f5f9]' : '□text-[#262626]'}`}
            >
              Name
            </Text>
            <Input>
              <InputField
                className={` ${isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]
                '}`}
                type='text'
                placeholder='Enter your name'
                value={name}
                onChangeText={setName}
              />
            </Input>
            {errors.name && (
              <Text className='■text-red-500 text-sm'>{errors.name}</Text>
            )}
          </VStack>
```

(Б9) ProductDetailsScreen(Fragment)

```
const ProductDetailsScreen = () => {
  return (
    <Box
      className={`flex-1 items-center p-3 ${isDarkMode ? '□bg-[#262626]' : '■bg-
      [#f1f5f9]'}`}
    >
      <Stack.Screen options={{ title: doll.dollName }} />

      <Card
        className={`p-4 rounded-lg max-w-[960px] w-full flex-1 ${isDarkMode ?
        '□bg-[#000]' : '■bg-[#fff]'}`}
      >
        <Image
          source={{
            uri: doll.image,
          }}
          className='mb-6 max-w-[450px] aspect-square w-full rounded-md'
          alt={`${doll.dollName} image`}
          resizeMode='contain'
        />
        <Text
          className={`text-4xl font-normal mb-2 text-typography-700 $
          {isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]'}`}
        >
          {doll.dollName}
        </Text>
        <VStack className='mb-6'>
          <Heading
            size='md'
            className={`mb-4 ${isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]
            '}`}
          >      You, 23 hours ago • Project done
            ${doll.price}
          </Heading>
          <Text
            size='md'
            className={`${isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]'}`}
```

(Б10) OrderDetailsScreen(Fragment)

```
export default function OrderDetailsScreen() {
  return (
    <ScrollView
      className={`flex-1 ${isDarkMode ? '□bg-[#262626]' : '■bg-[#f1f5f9]'}`}
    >
      <Stack.Screen options={{ title: `Order ${order.id}` }} />
      <VStack
        className={`p-4 ■bg-white shadow rounded-lg m-4 ${isDarkMode ?
        '□bg-black' : '■bg-white'}`}
      >
        <Text
          className={`text-xl font-bold mb-2 ${isDarkMode ? '■text-[#f1f5f9]' :
          '□text-[#262626]'}`}
        >
          Order #{order.id}
        </Text>
        <Text
          className={`text-sm □text-black ${isDarkMode ? '■text-[#f1f5f9]' :
          '□text-[#262626]'}`}
        >
          Created at: {new Date(order.createdAt).toLocaleDateString()}
        </Text>
        <Text
          bold
          className={`text-lg □text-black mb-4 ${isDarkMode ? '■text-[#f1f5f9]'
          : '□text-[#262626]'}`}
        >
          Total: ${order.totalAmount.toFixed(2) || '0.00'}
        </Text>

        <VStack
          className={`border-t pt-4 ${isDarkMode ? '■border-[#f1f5f9]' :
          '□border-[#262626]'}`}
        >
          <Text
            className={`text-lg font-bold mb-2 ${isDarkMode ? '■text-[#f1f5f9]'
            : '□text-[#262626]'}`}
```

(Б11) CartScreen(Fragment)

```
export default function CartScreen() {
  return (
    <View
      className={`w-full h-full ${isDarkMode ? '□bg-[#262626]' : '■bg-[#f1f5f9]
      '}`}
    >
      <Stack.Screen options={{ title: `Cart` }} />
      {items.length > 0 ? (
        <Box className='w-full flex'>
          <FlatList
            data={items}
            contentContainerClassName={`gap-4 max-w-[960px] w-full mx-auto $
            {isDarkMode ? 'bg-black' : 'bg-white'}`}
            renderItem={({ item, index }) => (
              <Box>
                <HStack
                  className={`■bg-white p-3 items-center justify-between $
                  {isDarkMode ? '□bg-black' : '■bg-white'}`}
                >
                  <HStack className='gap-3 items-center'>
                    <Text
                      className={`text-lg ${isDarkMode ? '■text-[#f1f5f9]' :
                      '□text-[#262626]'} `}
                    >
                      {index + 1}.
                    </Text>
                    <Image
                      className='w-20 h-20 rounded-sm'
                      source={{ uri: item.doll.image }}
                      alt={item.doll.dollName}
                    />
                    <VStack className='justify-center'>
                      <Text
                        className={`text-xl ${isDarkMode ? '■text-[#f1f5f9]' :
                        '□text-[#262626]'}`}
                        bold
                      >
```

(Б12) ProfileScreen(Fragment)

```
export default function ProfileScreen() {
  return (
    <ScrollView
      className={`flex-1 p-2 ${isDarkMode ? '□bg-[#262626]' : '■bg-[#f1f5f9]'}`}
    >
      <Stack.Screen options={{ title: 'Profile' }} />
      <VStack
        className={`p-4 ■bg-white rounded-lg shadow my-4 ${isDarkMode ?
        '□bg-black' : '■bg-white'}`}
      >
        <Text
          className={`text-xl font-bold mb-2 ${isDarkMode ? '■text-[#f1f5f9]' :
          '□text-[#262626]'}`}
        >
          Personal info
        </Text>
        <Text
          className={`text-lg  ${isDarkMode ? '■text-[#f1f5f9]' : '□text-
          [#262626]'}`}
        >
          Name: {user.name}
        </Text>
        <Text
          className={`text-lg  ${isDarkMode ? '■text-[#f1f5f9]' : '□text-
          [#262626]'}`}
        >
          Email: {user.email}
        </Text>
      </VStack>

      <VStack
        className={`p-4 rounded-lg shadow ${isDarkMode ? '□bg-black' :
        '■bg-white'}`}
      >
        <Text
          className={`text-xl font-bold mb-2 ${
            isDarkMode ? '■text-[#f1f5f9]' : '□text-[#262626]'
```

(Б13) SettingsScreen(Fragment)

```
export default function SettingsScreen() {
  const isLoggedIn = useAuth((state: any) => !!state.token);
  const logoutFn = useAuth((state: any) => state.logout);

  const { isDarkMode, setIsDarkMode, useDeviceSettings, setUseDeviceSettings } =
    useContext(DarkMode);

  const scheme = useColorScheme();
  const currentActivatedTheme: ColorSchemeName = isDarkMode ? 'dark' : 'light';
  function handleUseDeviceTheme() {···
  }

  const toggleDarkMode = useCallback(() => {      You, 23 hours ago • Project done ···
  }, [isDarkMode, scheme, useDeviceSettings]);

  useEffect(() => {···
  }, [isDarkMode, useDeviceSettings]);

  return (
    <VStack
      className={`w-full h-full ${isDarkMode ? '□bg-[#262626]' : '■bg-[#f1f5f9]
      '} gap-4 p-2`}
    >
      <Stack.Screen options={{ title: `Settings` }} />

      <Box
        className={`■bg-white rounded-lg shadow p-4 ${isDarkMode ? '□bg-black' :
        '■bg-white'}`}
      >
        <Heading
          className={`font-bold mb-4 ${isDarkMode ? '■text-[#f1f5f9]' : '□text-
          [#262626]'}`}
        >
          Appearance
        </Heading>
        <HStack className='justify-between items-center p-3'>
          <Text
```