МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
« » червня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Веб-орієнтована система інформаційного забезпечення продажу продукції компанії Krispy Kreme»
здобувача групи ІН-01 Циганенко Катерини Андріївни

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

_____ Катерина ЦИГАНЕНКО

(підпис)

Керівник,
кандидат техн. наук,
доцент кафедри комп'ютерних наук
к.т.н., доцент Ігор ШЕЛЕХОВ _____
(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувачки групи ІН-01 Циганенко Катерини Андріївни

1. Тема роботи: «Веб-орієнтована система інформаційного забезпечення продажу продукції компанії Krispy Kreme»

   затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*

2. Термін здачі здобувачем кваліфікаційної роботи

3. Вхідні дані до кваліфікаційної роботи _____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

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 на виконання _____ Керівник _____

(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблематики предметної області, формулювання і постановка завдань дослідження*		
2	*Огляд технологій, що застосовуються для рекомендації комплементарних товарів*		
3	*Розробка веб-орієнтованої системи інформаційного забезпечення продажу продукції компанії Krispy Kreme*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 62 стр., 20 рис., 2 додатки, 14 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена дослідженню важливої теми розвитку електронної комерції та методам створення веб систем для неї.

Об’єкт дослідження — веб системи для електронної комерції.

Мета роботи — розробка веб-орієнтованої системи інформаційного забезпечення продажу продукції компанії Krispy Kreme.

Методи дослідження — методи створення веб систем для електронної комерції.

Результати — розроблено веб-орієнтованої системи інформаційного забезпечення продажу продукції компанії Krispy Kreme, яка має стандартний функціонал для висвітлення товару та забезпечення продажів. Проведено тестування та порівняння з іншими системами.

ІНФОРМАЦІЙНА СИСТЕМА, ЕЛЕКТРОННА КОМЕРЦІЯ, REACT JS, EXPRESS JS, POSTGRESQL, NODE JS.

Зміст

Вступ

Обґрунтування вибору теми роботи:

Актуальність. З підвищенням популярності онлайн-покупок і зміною споживчих звичок людей, дослідження ринку онлайн-продажів продуктів харчування може виявити потенційні можливості для розвитку бізнесу. Інформаційні технології стають ключовими у цьому процесі. Великі компанії, такі як Krispy Kreme, постійно шукають нові способи покращити свої методи продажу та залучити клієнтів.

Об’єкт дослідження. Процес продажу продукції компанії Krispy Kreme через веб-орієнтовану систему інформаційного забезпечення.

Предмет дослідження. Можливості впровадження веб-орієнтованої системи інформаційного забезпечення для оптимізації процесів торгівлі та задоволення потреб клієнтів компанії Krispy Kreme.

Гіпотеза. Впровадження веб-орієнтованої системи інформаційного забезпечення для продажу продукції Krispy Kreme полегшить доступ клієнтів до продукції, покращить обслуговування та підвищить конкурентоспроможність компанії.

Новизна. Дослідження впливу веб-орієнтованої системи на процеси продажу продукції у сфері харчової промисловості є новаторським і може виявити неочікувані можливості для вдосконалення бізнесу Krispy Kreme.

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

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд компанії Krispy Kreme

Krispy Kreme — міжнародна мережа магазинів пончиків, заснована в 1936 році Полом Верноном Рудольфом у Нешвіллі, штат Теннессі. Перший магазин був відкритий у 1937 році під назвою "Krispy Kreme – Doughnut Shop". Найпопулярнішими є пончики з глазур'ю, які продаються гарячими, на відміну від інших асортиментів.

Продукція Krispy Kreme продається в магазинах мережі, продуктових крамницях, міні-маркетах, на автозаправних станціях та в супермаркетах Wal-Mart і Target у США. На міжнародному рівні продукція представлена в мережах супермаркетів Loblaws та на автозаправних станціях Petro-Canada в Канаді, а також у мережах станцій технічного обслуговування у Великій Британії та магазинах Seven Eleven в Австралії. У Великій Британії пончики Krispy Kreme можна знайти в супермаркетах Tesco, Tesco Extra та на більшості станцій технічного обслуговування. До першого випуску акцій прибуток компанії був стабільним, але в останні місяці він зменшився. Незважаючи на це, нові філії відкрилися в центрі Філадельфії та багатьох інших містах.[1]

Заснована в США, Krispy Kreme перетворилася на міжнародну компанію, що працює в понад 30 країнах світу. Компанія має франчайзингові угоди з різними підприємцями та компаніями, що дозволяє їй займати лідируючі позиції на світовому ринку.

Однією з основних стратегій продажу Krispy Kreme є використання концепції франчайзингу, що дозволяє швидко розширюватися і виходити на різні ринки. Крім того, компанія активно використовує маркетингові кампанії та промо-акції для залучення клієнтів.

1.2 SWOT- аналіз компанії Krispy Kreme

SWOT — це ефективний інструмент бізнес-планування, який використовується в бізнесі для формування стратегій. Цей інструмент допомагає проаналізувати внутрішні фактори (сильні та слабкі сторони), які впливають, і зовнішні фактори (можливості та загрози), які можуть мати вплив на організацію.

SWOT може допомогти вам проаналізувати свій бізнес зі стратегічної точки зору. Це допоможе вам визначити, як використати свої можливості, використовуючи свої сильні сторони та як уникнути загроз та усунути слабкі сторони. [2]

Потрібно аналізувати:

- сильні сторони;
- слабкі сторони;
- можливості;
- загрози.

Сильні сторони

- Сильний бренд та відомість.
- Глибоке розуміння ринку глазурованих тістечок.
- Ефективна модель франчайзингу.

Слабкі сторони

- Залежність від попиту на солодощі.
- Конкуренція на ринку fast-food.

Можливості

- Розширення асортименту продукції.
- Розвиток онлайн-продажів та доставки.

Загрози

- Зміни у смакових уподобаннях споживачів.
- Зростання конкуренції в галузі fast-food.

Висновки

Аналіз поточного стану компанії Krispy Kreme показує, що вона має сильну основу для подальшого розвитку та росту. Проте, важливо враховувати поточні тенденції ринку та реагувати на них швидко та ефективно для забезпечення конкурентоспроможності.

1.3 Інтернет комерція

Електронна комерція, або e-commerce - це сфера економіки, коли торгові і фінансові операції проводяться в інтернеті. Якщо говорити простими словами, це будь-яка транзакція, здійснена з електронного пристрою, підключеного до мережі. Аналог торгового центру, але з великим асортиментом і комфортом: його можна відвідати, не виходячи з дому.

Інтернет надає унікальні можливості для комунікації з аудиторією. Спочатку їх гідно оцінили в США, запустивши електронну торгівлю в 1979, потім перспективний напрям поширився по Європі, починаючи з 1981, і вже в кінці 1990-х Китай та країни колишнього Радянського Союзу підхопили естафету.

Цифрова економіка щомісяця набирає обертів: приблизно 8 млрд чоловік населяють нашу планету, до інтернету же підключено 7 млрд пристроїв. Тепер електронній комерції відведена значна частина світового фінансового ринку.

Основний, але далеко не єдиний, плюс інтернет-торгівлі полягає в відсутності географічних обмежень. Співпраця з логістичними компаніями допоможе охопити аудиторію в світовому масштабі. Так діють гіганти інтернет-торгівлі, такі як AliExpress, Alibaba, eBay, Amazon, Zappos, iTunes.

Коли бренд стає популярний, логотип пізнаваний, бізнес отримує нових користувачів по «сарафанному» радіо: люди діляться посиланнями в соціальних мережах на цікаві товари, приємні ціни або УТП (унікальна торгова пропозиція). Таким чином, інтернет допомагає привести умовно безкоштовних клієнтів.

Торговельне обладнання, найм персоналу, оренда приміщень - це все для торгівлі офлайн. В інтернеті про ці витрати можна забути.

Пошук цільової аудиторії став легше - людям подобається збиратися в групи за інтересами в соцмережах, на тематичних форумах і сайтах.

Ціна товару в інтернет-магазині найчастіше нижче, ніж в магазинах офлайн. Це пов'язано зі скороченням ланцюжка посередників між постачальником і замовником, і зменшенням витрат на утримання.

Плюси e-commerce для організацій:

- Зростає якість обслуговування клієнтів.
- Бізнес-процеси стають швидше, ефективніше і простіше.
- Набагато менше паперової роботи.
- Підвищується продуктивність організації: процес запускається, коли надходить запит від клієнта - ні пізніше, ні раніше, без зайвих дій і затримок.

Переваги онлайн-торгівлі для клієнта:

- Підтримка і доступність з будь-якого місця цілодобово: вибрати товар, оформити покупку і доставку можна в будь-який час доби, з будь-якого місця, де є підключення до мережі.
- Можливість вибору і порівняння товарів в різних магазинах.
- Можна залишити свій коментар або побачити відгуки про товар інших покупців перш, ніж прийняти рішення про покупку.
- Завжди доступна докладна інформація про товар або послугу.

- Підприємець змушений через високу конкуренцію в електронній комерції робити знижки, проводити акції і конкурси. Багато банків пропонують кешбек при оплаті картою.

Незважаючи на сильні плюси, у цифрових фінансових відносинах є і недоліки.

Недоліки онлайн-комерції:

- Велика конкуренція в популярних напрямках онлайн-торгівлі. Наприклад, продаючи побутову, відео та аудіотехніку, мобільні телефони доведеться буквально боротися за кожного покупця. В цьому випадку необхідні чималі витрати на маркетинг, SMM просування, SEO оптимізацію сайту.
- Мало довіри до нових інтернет-магазинів з боку користувачів. Цьому сприяли шахраї, які давно звернули увагу на електронну комерцію.
- Відсутність знань про роботу в мережі. Тут працюють принципи, відмінні від торгівлі офлайн: аналітика, логістика, обробка замовлень відбуваються інакше. Затримка з відповідями на запити покупців може зіпсувати імідж компанії.
- Успішний бізнес залежить від стабільного швидкого інтернету і щоденної залученості власника в управління справою.
- Для регулювання бізнес-процесів в мережі потрібна добре розроблена законодавча база. Її немає. В тому числі, це стосується захисту персональних даних. Необхідна інтеграція актуального програмного забезпечення, часто дорогого.
- Замовник не може прийти і самостійно забрати покупку. Потрібен ідеально налагоджений механізм доставки.
- Відповідальних віддалених співробітників важко знайти і навчити.

Всі ці проблеми електронної комерції можна вирішити. Звичайно, доведеться потрудитися, але результат того вартий.[3]

1.4 Веб-застосунок для інтернет комерції

Що таке інтернет-застосунок?

Веб-програма — це програмне забезпечення, яке запускається у веб-браузері. Компанії повинні обмінюватися інформацією та надавати послуги віддалено. Вони використовують веб-програми для зручного та безпечного зв'язку з клієнтами. Найбільш поширені функції веб-сайту, такі як кошики, пошук та фільтрація товарів, обмін миттєвими повідомленнями та стрічки новин соціальних мереж, за своєю структурою є веб-застосунками. Вони дозволяють отримати доступ до складних функціональних можливостей без встановлення або налаштування програмного забезпечення.

Які основні переваги веб-застосунків?

Веб-застосунки мають ряд переваг, і майже всі великі компанії використовують їх як частину своїх пропозицій користувача. Ось деякі з найбільш поширених переваг, пов'язаних із веб-застосунками.[4]

- Доступність
- Ефективна розробка
- Простота для користувача
- Масштабованість

1.5 Приклади сайтів для електронної торгівлі

Рисунок 1.5.1 Сайт компанії FRONKS

Для початку поглянемо на сайт молодої та не дуже відомої компанії. FRONKS засновані у 2016 році, і є маленькою компанією, яка виготовляє органічне молоко (з суміші пророслих горіхів) яке не містить молочних продуктів та пророщені горіхи. Вони ведуть свою діяльність лише у декількох штатах США і їм не потрібен великий інтернет-магазин з можливістю додати товар до кошика та сортування, тому мінімалістичний сайт це те що їм потрібно.

З іншої сторони ми можемо розглянути те що кожен з нас розуміє під словом «інтернет-магазин». Наприклад сайт магазину Rozetka.

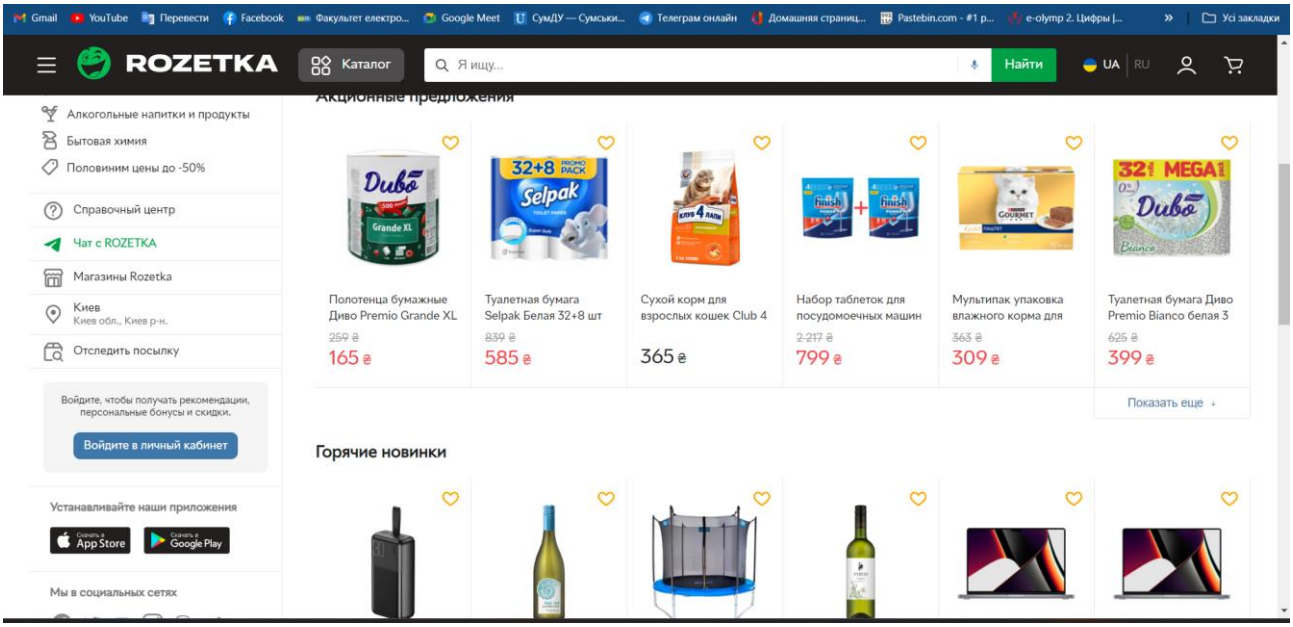


Рисунок 1.5.2 Сайт магазину Rozetka

На цьому сайті розміщені різні товари різних категорій, ви можете обрати товари, додати їх до кошика та оплатити свою покупку. Також можете зареєструватися або авторизуватися, щоб алгоритми підібрали товари, які будуть вам до вподоби.

1.6 Постановка задачі

Метою роботи є розробка програмного забезпечення для веб-магазину електронної комерції компанії Krispy Kreme. Дане дослідження виконується в рамках кваліфікаційної роботи бакалавра за спеціальністю 122 «Комп'ютерні науки».

Розроблений веб-додаток повинен включати наступний функціонал:

- Можливість реєстрації за допомогою логіну та паролю.
- Авторизація в систему за допомогою облікових даних користувача.
- Додавання та видалення товарів з кошика.
- Перегляд різних сторінок сайту.
- Перегляд сторінок товарів.
- Можливість сортувати товари за категоріями.

2 ВИБІР МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель додатка

Архітектура веб-додатку є макетом з усіма програмними компонентами (такими як бази даних, додатки та проміжне програмне забезпечення) та їх взаємодією один з одним. Він визначає, як дані доставляються через HTTP, та гарантує, що клієнтський сервер та внутрішній сервер зможуть їх зрозуміти. Більше того, це також гарантує, що у всіх запитах користувачів є дійсні дані. Він створює записи та керує ними, забезпечуючи при цьому доступ та автентифікацію на основі дозволів. Вибір правильної конструкції визначає зростання, надійність, сумісність та майбутні IT-потреби вашої компанії. Таким чином, важливо розуміти компоненти, що становлять архітектуру веб-додатків.

Компоненти архітектури веб-додатків

1. Веб-браузер. Браузер, клієнтський компонент або інтерфейсний компонент є ключовим компонентом, який взаємодіє з користувачем, отримує вхідні дані та керує логікою подання, одночасно контролюючи взаємодію користувача з програмою. При необхідності дані користувача також перевіряються.

2. Веб-сервер. Веб-сервер, також відомий як внутрішній компонент або компонент на стороні сервера, обробляє бізнес-логіку та обробляє запити користувачів, надсилаючи запити до потрібного компонента та керуючи всіма операціями програми. Він може обробляти і контролювати запити від різних клієнтів.

3. Сервер бази даних. Сервер бази даних надає необхідні дані для програми. Він вирішує завдання, пов'язані із даними. У багаторівневій архітектурі сервери баз даних можуть керувати бізнес-логікою за допомогою процедур, що зберігаються.

Існує три рівні трирівневої архітектури:

1. Рівень подання/клієнтський рівень

2. Прикладний рівень/Бізнес-рівень

3. Рівень даних

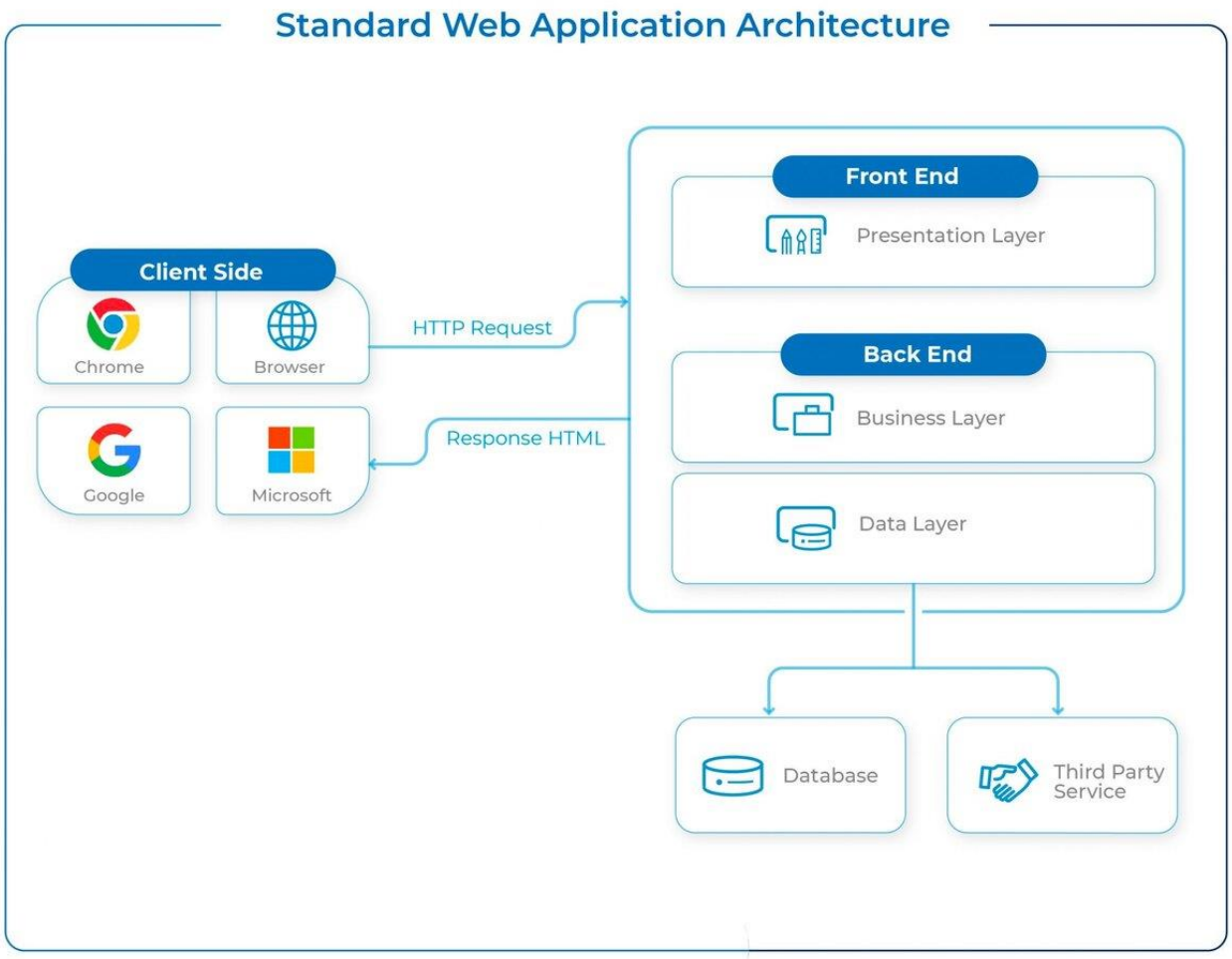


Рисунок 2.1.1 Інформаційна модель трирівневої архітектури веб-додатка

У цій моделі проміжні сервери одержують запити клієнтів та обробляють їх, координуючи свої дії з підлеглими серверами, застосовуючи бізнес-логіку. Зв'язок між клієнтом та базою даних управляється проміжним прикладним рівнем, що дозволяє клієнтам отримувати доступ до даних різних рішень СУБД.

Трирівнева архітектура безпечніша, оскільки клієнт не має прямого доступу до даних. Можливість розгортання серверів додатків на кількох машинах забезпечує більш високу масштабованість, кращу продуктивність та краще повторне використання. Ви можете масштабувати його по горизонталі, масштабуючи кожен елемент незалежно. Можна абстрагувати основну діяльність від сервера бази даних, щоб ефективно виконувати балансування навантаження. Цілісність даних покращується, оскільки всі дані проходять через сервер додатків, який вирішує, як і кому слід отримувати доступ до даних. З цієї причини зміна керівництва є простим та економічно ефективним процесом. Клієнтський рівень може бути тонким клієнтом, що означає зниження витрат за обладнання. Ця модульна модель дозволяє модифікувати один рівень, не торкаючись інших компонентів.[9]

Отже, трирівнева архітектура була обрана для написання веб-додатку через надійність, можливість масштабованості, незалежність кожного аспекту та продуктивність.

2.2 Структура бази даних

Перед початком розробки слід визначити які дані ми будемо зберігати. Так як на нашому веб додатку користувачі можуть переглядати товари певних категорій, то в нашій БД ми повинні зберігати дані про користувачів, категорії товару, сам товар, а також кошики користувачів, куди вони заносять обрані товари. Для цього створимо базу даних з таблицями для користувачів, товарів, категорій, та

кошику.

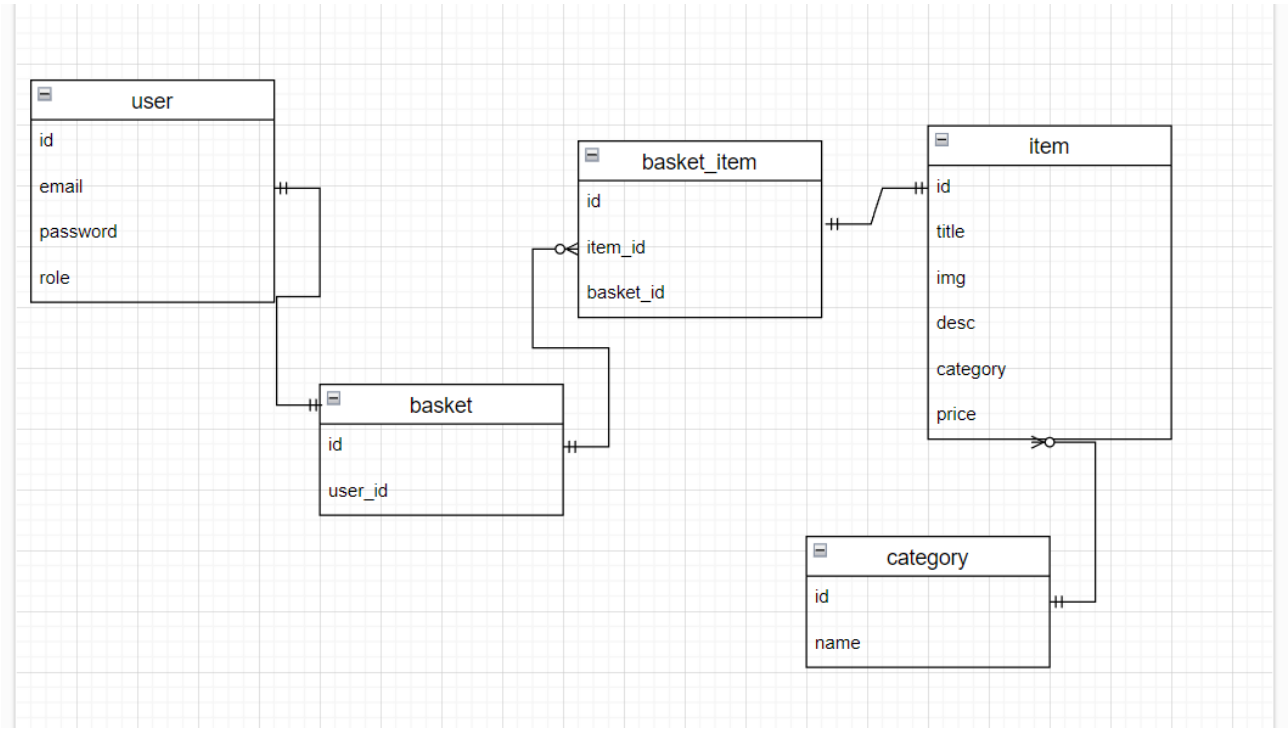


Рисунок 2.2.1 ER-діаграма таблиць та їх зав’язків бази даних

2.3 Структура серверної частини проекту

Структура серверної частини проекту складається з 6 директорії та файлу ініціалізації сервера (index.js)(див. рис. 2.3.1):

- Controllers - Файли в цій директорії містять логіку контролерів для обробки запитів до API.
- Error - Файли в цій директорії містять логіку для обробки помилок.
- Middleware - Файли в цій директорії містять проміжне ПЗ для обробки запитів.
- Models - Файли в цій директорії містять визначення моделей для бази даних.
- Routes - Файли в цій директорії містять визначення маршрутів для API.
- Static - Директорія для зберігання статичних файлів, таких як зображення товарів.

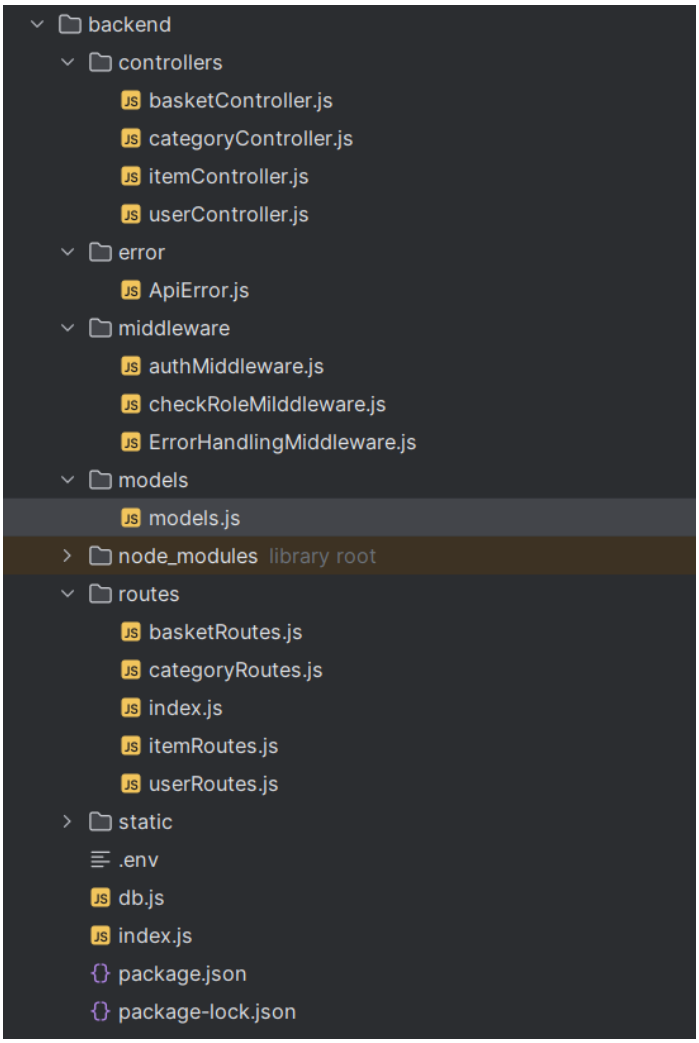


Рисунок 2.3.1 Структура серверної частини проекту

2.4 Структура клієнтської частини проекту

Структура клієнтської частини проекту складається з 5 директорії та файлів App.js, index.css, index.js, riutes.js (див. рис. 2.4.1):

- Components - Файли в цій директорії містять компоненти, які використовуються на різних сторінках.
- Http - Файли в цій директорії містять логіку для виконання HTTP-запитів.
- Pages - Файли в цій директорії містять логіку сторінок, які відображаються користувачу.
- Store - Файли в цій директорії містять логіку для управління станом.
- Utils - Файли в цій директорії містять утиліти та допоміжні функції такі як константи шляхів до сторінок.

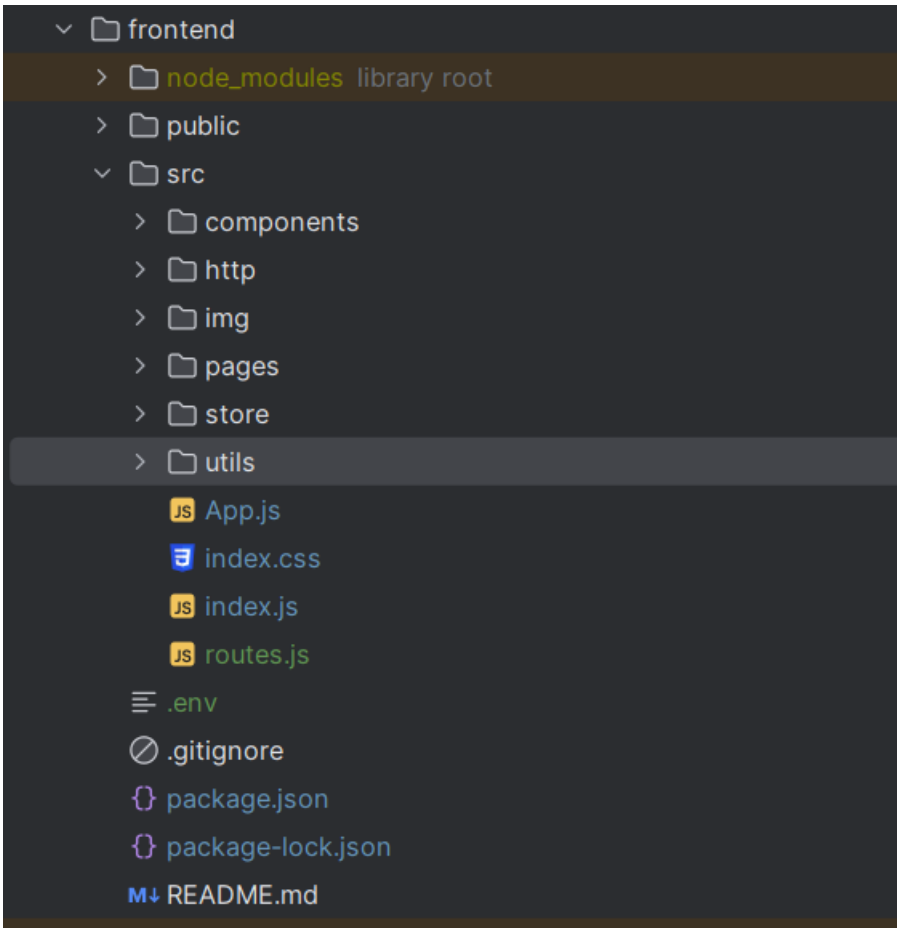


Рисунок 2.4.1 Структура клієнтської частини проекту

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Використані технології

Для написання серверної частини веб-додатку було використано фреймворк Node.js[10] Express.js[11]. Node.js — це відкрите програмне забезпечення, яке використовується для створення сучасних веб-додатків. Його основними перевагами є продуктивність, масштабованість, велика кількість розширень, а також підтримка мови програмування JavaScript. Його аналогами є такі фреймворки, як Ruby on Rails, написаний на Ruby, Django, написаний на Python, та ASP.NET Core, написаний на C#. Всі ці фреймворки використовуються для написання веб-додатків, але в нашому випадку було обрано саме Node.js тому, що при написанні програм використовується мова програмування JavaScript, яка має дуже гарні бібліотеки та фреймворки, що допомагають при роботі з даними та базами даних, такі як Sequelize[12] і Mongoose, і по-друге, JavaScript має за замовчуванням функціонал асинхронності, що допомагає виконувати операції паралельно з іншими операціями, без блокування виконання основного коду програми. Саме ці властивості допоможуть нам написати веб-додаток, який буде виконувати весь функціонал бізнес логіки, а також роботу з базою даних дуже швидко, при цьому не витрачаючи час на написання зайвого коду для підтримки функціоналу, якого немає в інших фреймворках, або ж які реалізують його гірше, ніж Node.js.

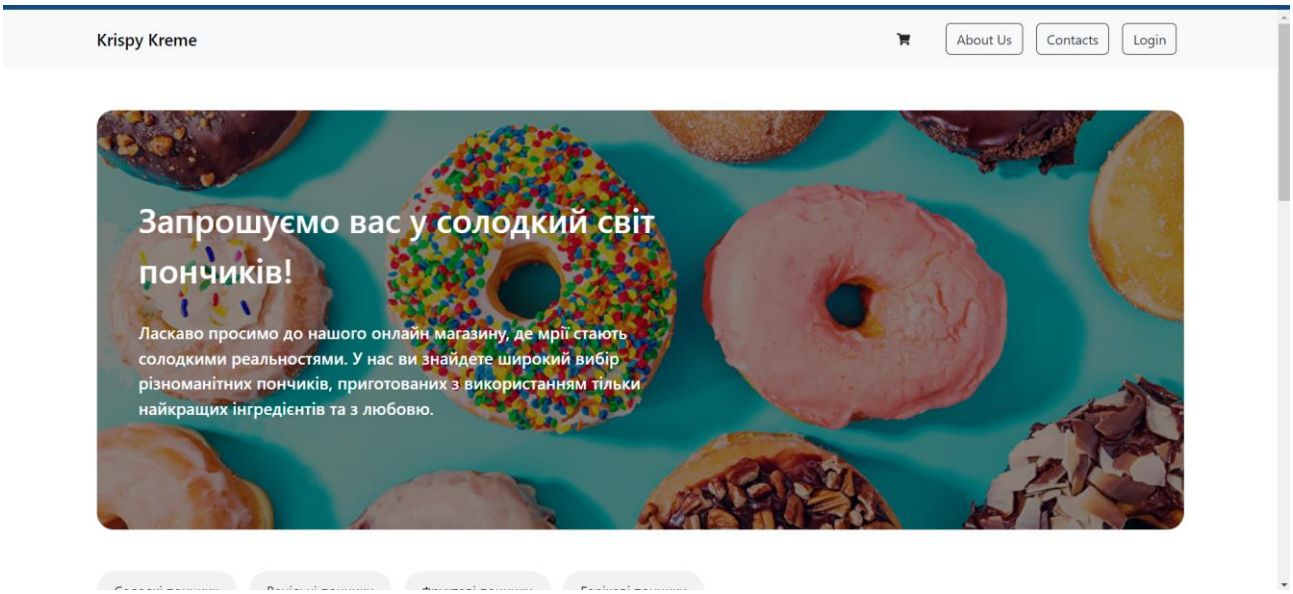
Як СУБД для нашого проекту було обрано PostgreSQL[13]. PostgreSQL — це потужна реляційна база даних, яка використовується для збереження, опрацювання та управління даними. Через її підтримку складних запитів, транзакцій та розширень, PostgreSQL підходить для обробки великих обсягів інформації та запитів, що є важливим для нашого проекту. Хоча і існують інші гарні альтернативи, такі як MySQL та SQLite, PostgreSQL надає більше

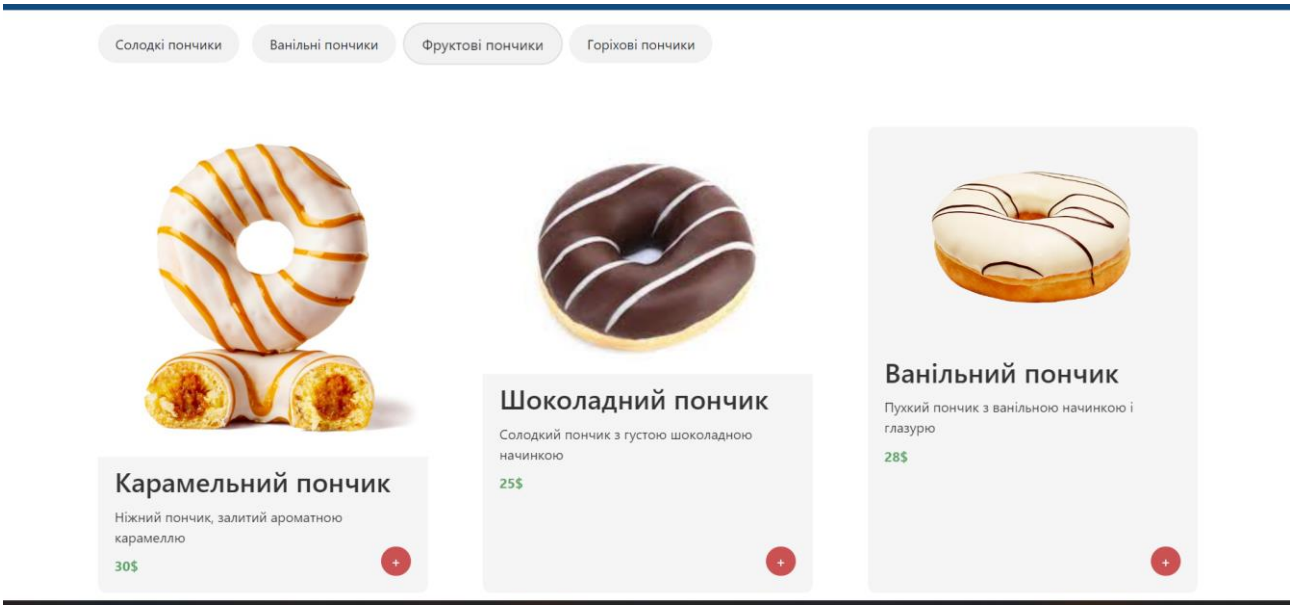
можливостей для масштабування і складних обчислень, що є критичним для нашого веб-додатку.

Для написання клієнтської частини веб-додатку було використано бібліотеку React[5] з використанням Bootstrap[14]. React — це бібліотека для розробки веб-додатків, яка дозволяє створювати динамічні та інтерактивні користувацькі інтерфейси. Аналогами до React є такі популярні фреймворки, як Angular та Vue.js. Всі ці засоби виконують одну й ту саму функцію, але в кожного з них є свої переваги та недоліки. У порівнянні з Angular, React надає більше свободи в організації проекту, але це також може означати більше роботи для розробників у сенсі вибору правильних інструментів та архітектури. Vue.js, зі свого боку, славиться своєю швидкістю розробки та легкістю вивчення, чого не можна сказати про Angular. Незважаючи на це, React є гарним варіантом для нашого проекту через його гнучкість та підтримку великою кількістю розширень і інструментів, що робить його ідеальним для створення складних та масштабованих інтерфейсів.

3.2 Огляд розробленого додатка

Переглянемо головну сторінку додатку (див. рис. 3.2.1-3.2.3) та її можливості :





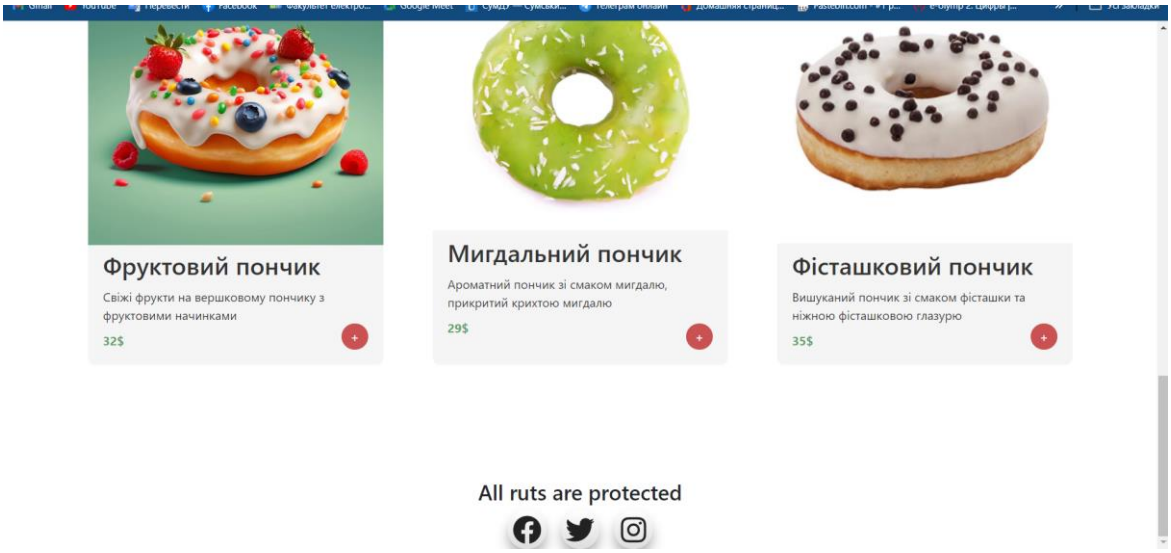


Рисунок 2.1-2.3 Головна сторінка додатку

- Можливість переглядати кошик.

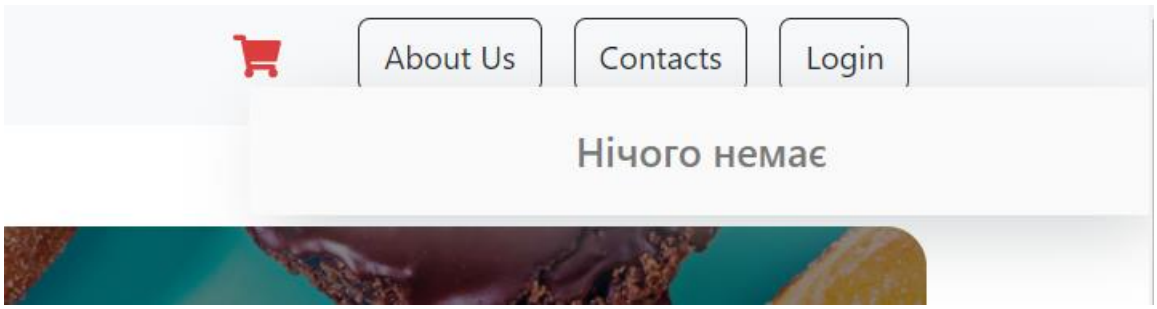


Рисунок 3.2.4 Пустий кошик

- Додавання та видалення товарів з кошика та обчислення загальної вартості покупок.

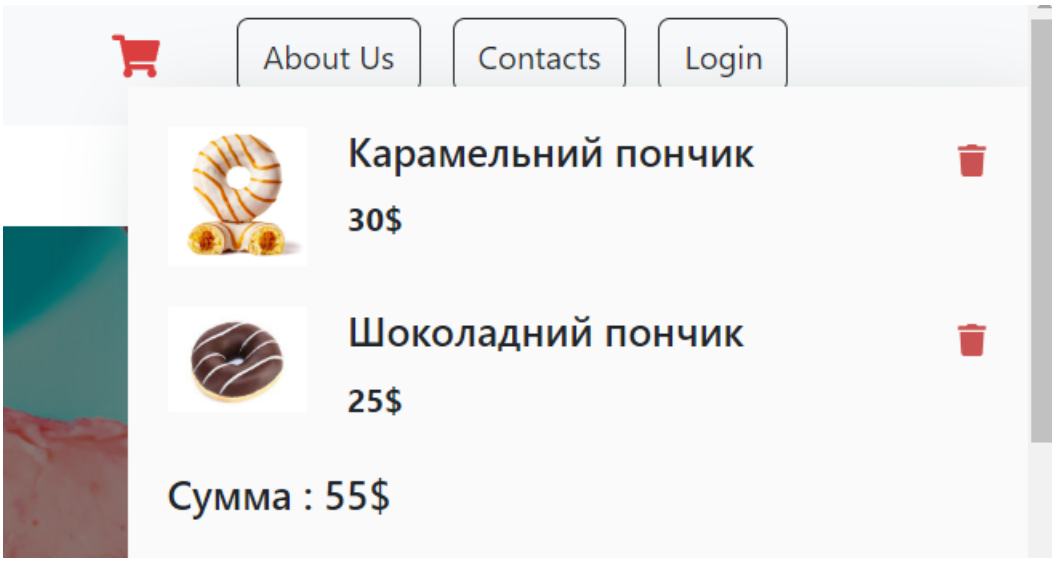


Рисунок 3.2.5 Додавання товарів до кошика

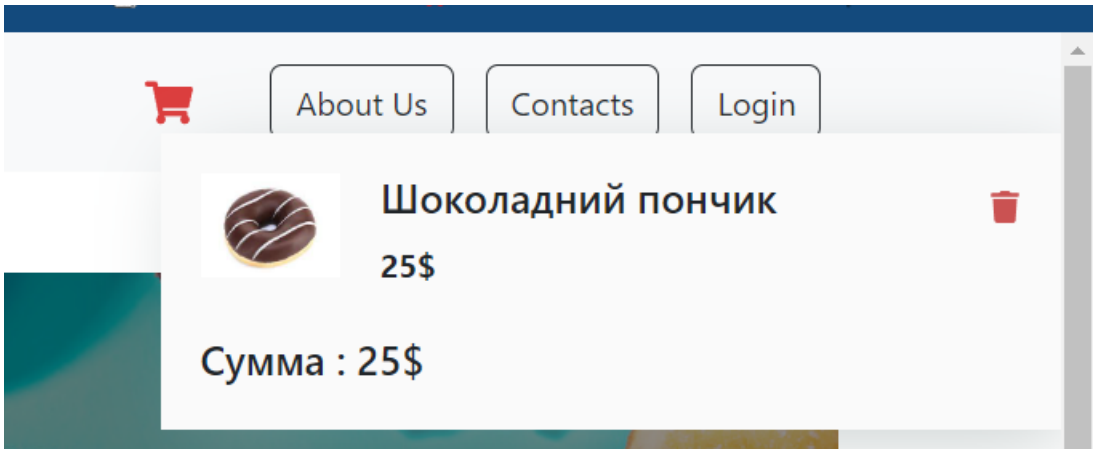


Рисунок 3.2.6 Видалення товарів з кошика

- Можливість переходити на окрему сторінку товару. Натиснувши на зображення товару ви перейдете на сторінку товару

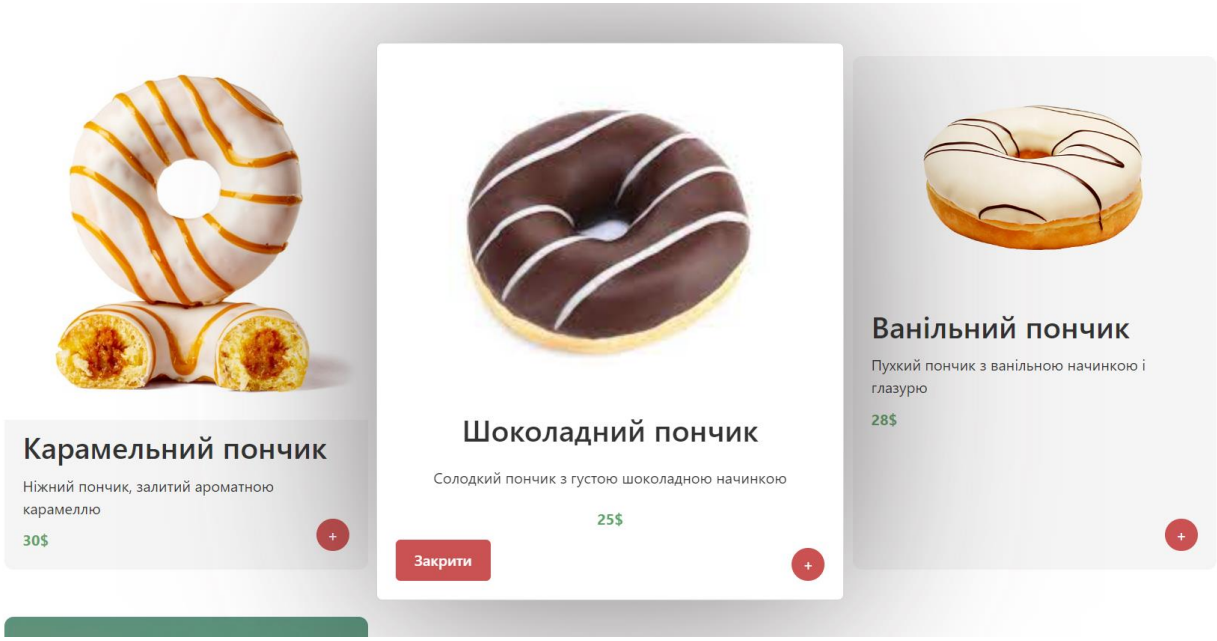


Рисунок 3.2.7 Сторінка товару

- Можливість сортувати товари за категоріями. Натиснувши на кнопку з категорією виділяються лише товари даної категорії

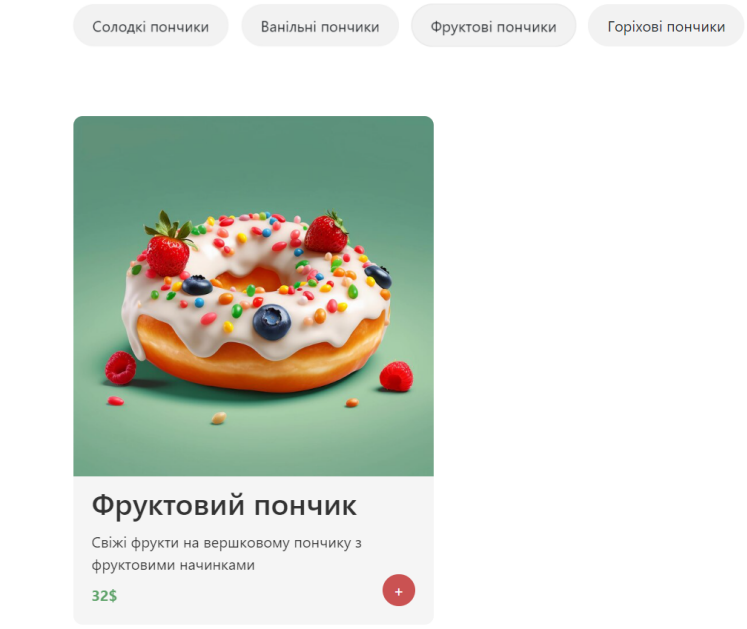


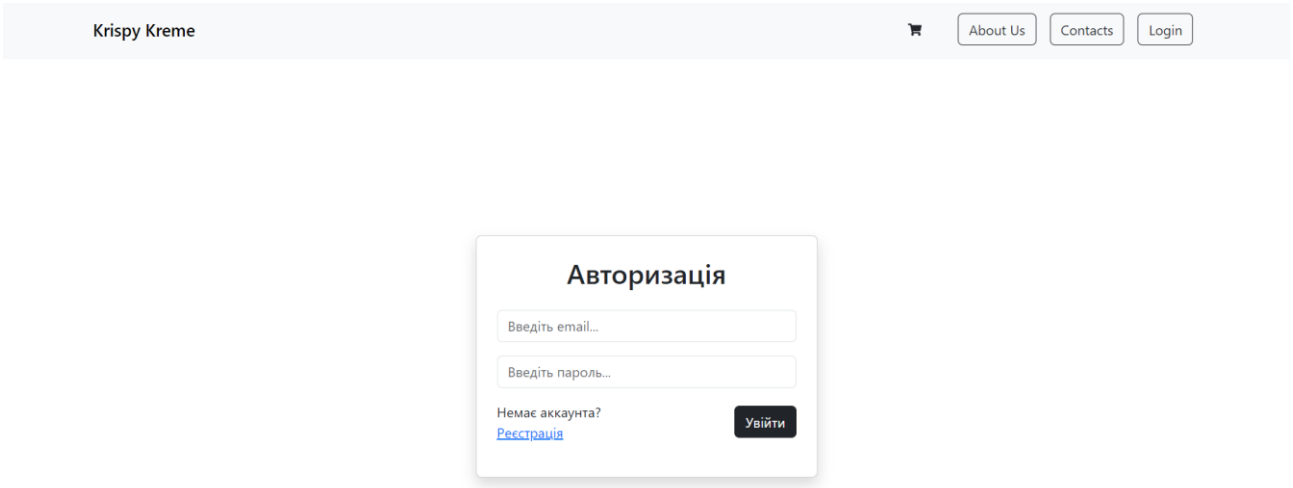
Рисунок 3.2.8 Сортування товару за категоріями

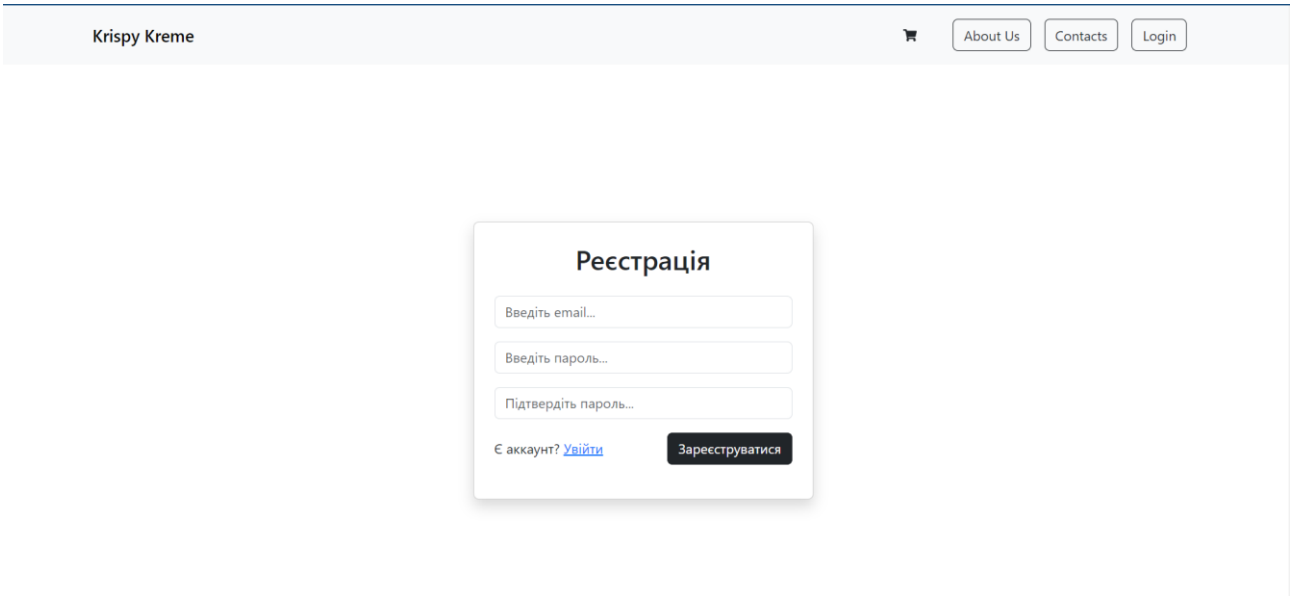
- Можливість перейти на сторінки соціальних мереж компанії

Рисунок 3.2.9 посилання на соціальні мережі

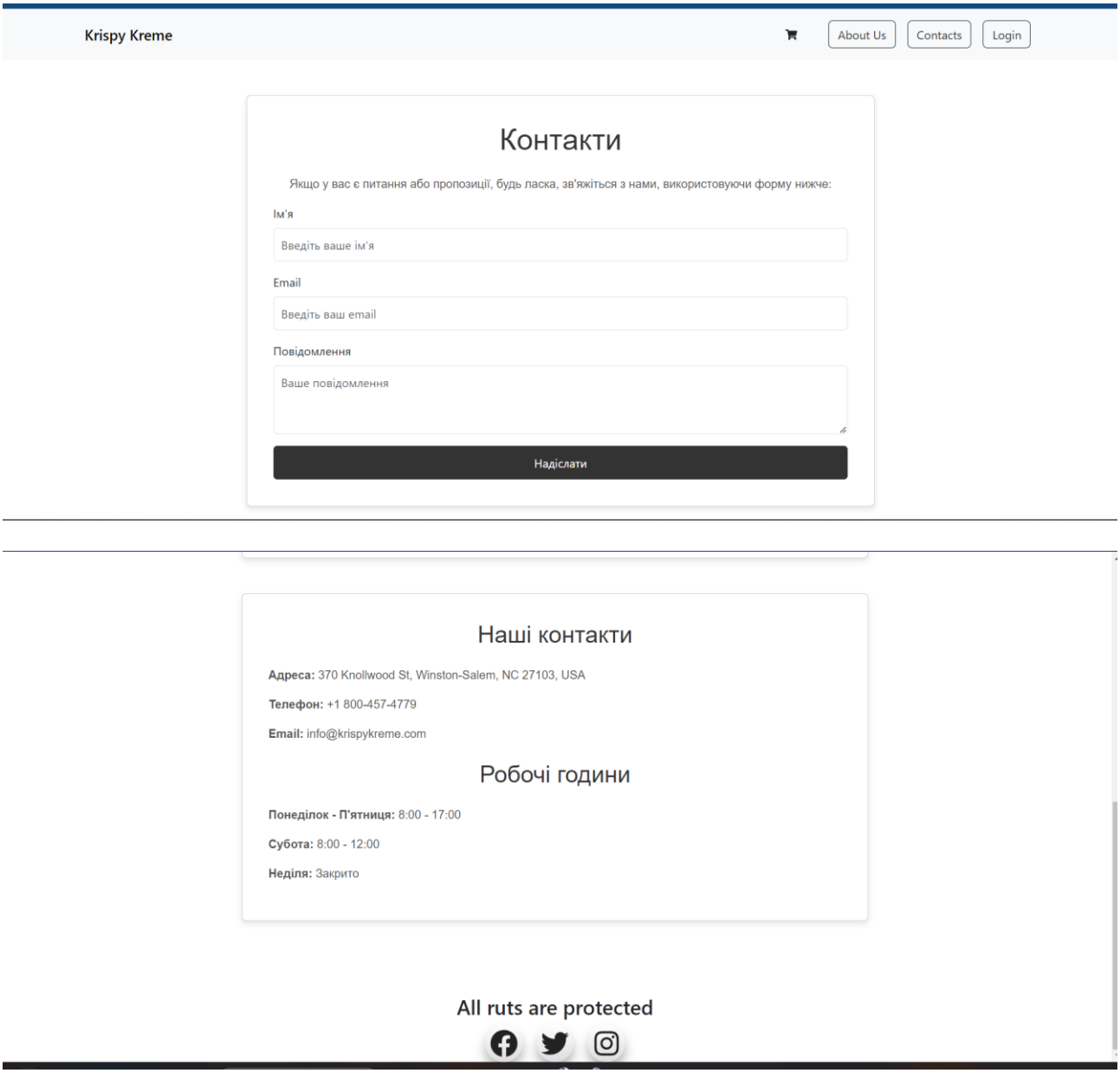
Тепер переглянемо інші сторінки додатку

- Сторінка авторизації/реєстрації

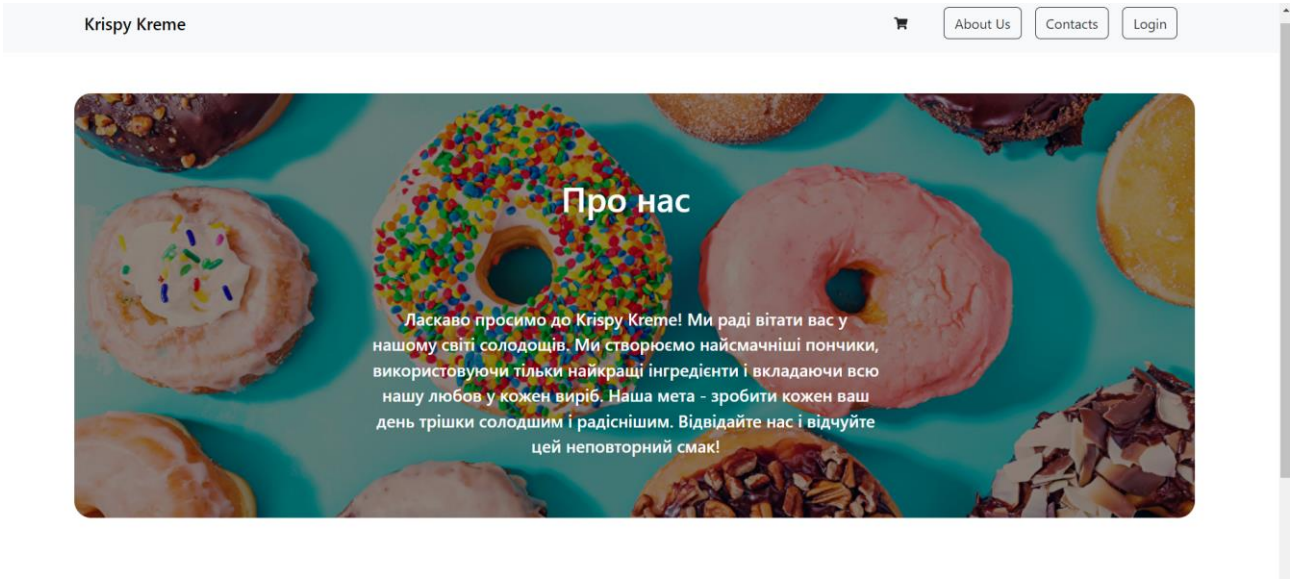




- Сторінка контактів



- Сторінка з інформацією



4 Висновок

В результаті виконання даної роботи було створено веб-додаток з використанням фреймворків Node.js та Express.js для серверної частини і React з використанням Bootstrap для клієнтської частини. Протягом виконання роботи були досягнуті наступні завдання:

1. **Проведено огляд літератури та сучасних технологій веб-розробки**:
    - Досліджено переваги та недоліки різних фреймворків для розробки серверної та клієнтської частин веб-додатків.
    - Node.js було обрано за його продуктивність, масштабованість, асинхронність та використання мови програмування JavaScript. Інші розглянуті фреймворки включали Ruby on Rails, Django та ASP.NET Core.
    - PostgreSQL було обрано як СУБД завдяки її потужності у обробці складних запитів та масштабованості.
2. **Розроблено архітектуру додатку та визначено інструменти для реалізації**:
    - Серверна частина побудована на Node.js та Express.js, що забезпечує високу продуктивність та можливість асинхронної обробки запитів.
    - Клієнтська частина створена за допомогою React, що забезпечує гнучкість та можливість створення динамічних користувацьких інтерфейсів.
    - В якості СУБД обрано PostgreSQL, яка підходить для обробки великих обсягів даних.
3. **Реалізовано основний функціонал веб-додатку**:
    - Створено функції реєстрації та авторизації користувачів.
    - Додано можливість додавання та видалення товарів з кошика, фільтрації за категоріями, перегляд інформації про конкретний товар та перехід на сторінки соціальних мереж.

- Реалізовано перегляд основних сторінок сайту, включаючи головну сторінку, сторінку авторизації/реєстрації, контакти та інформаційну сторінку.

4. **Проведено тестування додатку**:
   - Перевірено функціонування всіх основних компонентів додатку.
   - Встановлено, що додаток відповідає поставленим вимогам та працює згідно з очікуваннями.

Розроблена веб-система має всі необхідні функціональні можливості для забезпечення управління товарами в кошику користувачів. Впровадження цієї системи дозволить зменшити витрати часу на обробку замовлень, покращити якість обслуговування клієнтів та підвищити загальну продуктивність бізнесу. Завдяки використанню сучасних технологій та підходів, веб-додаток забезпечує високу швидкість роботи та масштабованість, що є критичним для успіху в умовах швидкозмінного ринку.

5 Список використаних джерел

1. About Krispy Kreme. Happiness through doughnuts. KrispyKreme.com. URL: https://www.krispykreme.com/about (дата звернення: 15.04.2024).
2. Що таке SWOT аналіз? BUSINESS.DIIA.GOV.UA. URL: https://business.diia.gov.ua/handbook/marketing/so-take-swot-analiz (дата звернення: 15.04.2024).
3. Що таке електронна комерція? E-commerce для початківців. INTERKASSA.COM. URL: https://interkassa.com/blog/shho-take-elektronna-komerciya-e-commerce-dlya-pochatkivciv (дата звернення: 15.04.2024).
4. What is a Web Application? Amazon.com. URL: https://aws.amazon.com/what-is/web-application/?nc1=h_ls (дата звернення: 16.04.2024).
5. React: Making faster, smoother UIs for data-driven Web apps. Infoworld.com. URL: https://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uis-for-data-driven-web-apps.html (дата звернення: 20.04.2024).
6. React Icons. React-icons.github.io. URL: https://react-icons.github.io/react-icons/icons/fa/ (дата звернення: 20.04.2024).
7. Fresh Fonks URL: https://freshfronks.com/ (дата звернення: 10.05.2024).
8. Інтернет-магазин Rozetka URL: https://rozetka.com.ua/ (дата звернення: 10.05.2024).
9. Архітектура веб-додатків: докладний посібник. URL: https://dzen.ru/a/ZVnaW_FP3F9f_xsc (дата звернення: 14.05.2024).
10. Introduction to Node.js. Nodejs.org. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення: 14.05.2024).
11. Express/Node.js Introduction. MDN Web Docs. URL: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/Introduction (дата звернення: 14.05.2024).
12. Sequelize. URL: https://sequelize.org/#:~:text=Sequelize%20is%20a%20modern%20TypeScript,lo

ading%2C%20read%20replication%20and%20more. (дата звернення: 20.05.2024).

13. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення: 20.05.2024).
14. Bootstrap. URL: https://getbootstrap.com/ (дата звернення: 20.05.2024).

Додаток А

Програмні файли серверної частини

BasketController.js

```
const { Basket, BasketItem, Items } = require('../models/models');

class BasketController {
    async addItem(req, res) {
        const { itemId } = req.body;
        const userId = req.user.id;

        let basket = await Basket.findOne({ where: { userId } });
        if (!basket) {
            basket = await Basket.create({ userId });
        }

        const basketItem = await BasketItem.create({ basketId: basket.id,
itemId });
        const item = await Items.findByPk(itemId);
        return res.json({ ...basketItem.dataValues, item });
    }

    async removeItem(req, res) {
        const { itemId } = req.body;
        const userId = req.user.id;

        const basket = await Basket.findOne({ where: { userId } });
        if (!basket) {
            return res.status(404).json({ message: 'Basket not found' });
        }

        await BasketItem.destroy({ where: { basketId: basket.id, itemId }
});
        return res.json({ message: 'Item removed' });
    }

    async getBasket(req, res) {
        try {
            const { userId } = req.query;
            const basket = await Basket.findOne({
                where: { userId },
                include: [{ model: BasketItem, include: [Items] }]
            });

            if (!basket) {
                return res.status(404).json({ message: "Basket not found"
});
            }

            const basketItems = basket.basket_items.map(basketItem => ({
                id: basketItem.id,
                title: basketItem.item.title,
                img: basketItem.item.img,
                desc: basketItem.item.desc,
                price: basketItem.item.price,
```

```
            }));

            return res.json(basketItems);
        } catch (error) {
            console.error('Error fetching basket:', error);
            return res.status(500).json({ message: "Internal server
error" });
        }
    }
}

module.exports = new BasketController();
```

CategoryController.js

```
const {Category} = require('../models/models')
const ApiError = require('../error/ApiError')
class CategoryController{
    async create (req,res){
        const {name} = req.body
        const category = await Category.create({name})
        return res.json({category})
    }

    async get (req,res){
        const categories = await Category.findAll()
        return res.json(categories)
    }

}

module.exports = new CategoryController()
```

ItemController.js

```
const uuid = require('uuid')
const path = require('path');
const {Items} = require('../models/models')
const ApiError = require('../error/ApiError')
class ItemController{
    async create (req,res, next){

        try {
            const {title, desc, price, categoryId} = req.body
            const {img} = req.files
            let fileName = uuid.v4() + ".jpg"
            img.mv(path.resolve(__dirname,'..', 'static', fileName))

            const item = await Items.create({title,img: fileName, desc,
price, categoryId})

            return res.json(item)
        } catch (e){
            next(ApiError.badRequest(e.message))
        }
```

```
    }

    async get (req,res){
        let {categoryId, limit, page} = req.body
        let items;

        page = page || 1
        limit = limit || 8

        let offset = page * limit - limit
        if (!categoryId){
            items = await Items.findAndCountAll({limit, offset})
        }

        else {
            items = await Items.findAndCountAll({where: {categoryId},
limit, offset})
        }

        return res.json(items)
    }

    async getOne (req,res){
        const {id} = req.params
        const item = await Items.findOne({
            where:{id}
        })
        return res.json(item)
    }
}

module.exports = new ItemController()
```

UserController.js

```
const ApiError = require('../error/ApiError');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');
const { User, Basket } = require('../models/models');

const generateJwt = (id, email, role) => {
    return jwt.sign({ id, email, role }, process.env.SECRET_KEY, {
expiresIn: '24h' });
};

class UserController {
    async registration(req, res, next) {
        const { email, password, role } = req.body;
        if (!email || !password) {
            return next(ApiError.badRequest('Enter email or password'));
        }

        // Використання правильного синтаксису для 'where'
        const candidate = await User.findOne({ where: { email: email }
});

        if (candidate) {
```

```
            return next(ApiError.badRequest('User with this email already
exists'));
        }

        const hashPassword = await bcrypt.hash(password, 5);
        const user = await User.create({ email, role, password:
hashPassword });

        const basket = await Basket.create({ userId: user.id });

        const token = generateJwt(user.id, user.email, user.role);

        return res.json({ token });
    }

    async login(req, res, next) {
        const { email, password } = req.body;
        // Використання правильного синтаксису для 'where'
        const user = await User.findOne({ where: { email: email } })

        if (!user) {
            return next(ApiError.internal('User not found'));
        }

        const comparePassword = bcrypt.compareSync(password,
user.password);

        if (!comparePassword) {
            return next(ApiError.internal('Wrong password'));
        }

        const token = generateJwt(user.id, user.email, user.role);

        return res.json({ token });
    }

    async check(req, res, next) {
        const token = generateJwt(req.user.id, req.user.email,
req.user.role);
        return res.json({ token });
    }
}

module.exports = new UserController();
```

ApiError.js

```
class ApiError extends Error{
    constructor(status, message) {
        super();
        this.status = status
        this.message = message
    }

    static badRequest(message){
        return new ApiError(404,message)
    }
```

```
    static internal(message){
        return new ApiError(500,message)
    }

    static forbidden(message){
        return new ApiError(403,message)
    }
}

module.exports = ApiError
```

AuthMiddleware.js

```
const jwt = require('jsonwebtoken');

module.exports = function (req, res, next) {
    if (req.method === "OPTIONS") {
        return next();
    }

    try {
        const token = req.headers.authorization;

        if (!token || !token.startsWith('Bearer ')) {
            return res.status(401).json({ message: "Not authorized" });
        }

        const tokenValue = token.split(' ')[1];
        req.user = jwt.verify(tokenValue, process.env.SECRET_KEY); //
Додаємо розшифрований об'єкт з інформацією про користувача у req
        next();
    } catch (e) {
        console.error(e);
        return res.status(401).json({ message: "Not authorized" });
    }
};
```

CheckRoleMiddleware.js

```
const jwt = require('jsonwebtoken')

module.exports = function (role){
    return function (req, res, next) {
        if (req.method === "OPTIONS") {
            next()
        }

        try {
            const token = req.headers.authorization.split(' ')[1]

            if (!token) {
                return res.status(401).json({message: "Not
authorization"})
            }
```

```
            const decoded = jwt.verify(token, process.env.SECRET_KEY)
            if (decoded.role !== role){
                return res.status(403).json({message: "No access"})
            }
            req.user = decoded
            next()
        } catch (e) {
            res.status(401).json({message: "Not authorization"})
        }
    };
}
```

ErrorHandlingMiddleware.js

```
const ApiError = require('../error/ApiError')

module.exports = function (err,req,res,next){
    if(err instanceof ApiError){
        return res.status(err.status).json({message: err.message})
    }

    return res.status(500).json({message: "Error Mistake"})
}
```

models.js

```
const sequelize = require('../db')
const { DataTypes } = require('sequelize')

const User = sequelize.define('user', {
    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true
},
    email: { type: DataTypes.STRING, unique: true },
    password: { type: DataTypes.STRING },
    role: { type: DataTypes.STRING, defaultValue: "USER" }
})

const Basket = sequelize.define('basket', {
    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true
},
})

const BasketItem = sequelize.define('basket_item', {
    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true
},
})

const Items = sequelize.define('items', {
    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true
},
    title: { type: DataTypes.STRING, unique: true, allowNull: false },
    img: { type: DataTypes.STRING, allowNull: false },
    desc: { type: DataTypes.STRING, allowNull: false },
    price: { type: DataTypes.INTEGER, allowNull: false },
})
```

```
const Category = sequelize.define('category', {
    id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true
},
    name: { type: DataTypes.STRING, unique: true, allowNull: false }
})

User.hasOne(Basket)
Basket.belongsTo(User)

Basket.hasMany(BasketItem)
BasketItem.belongsTo(Basket)

Category.hasMany(Items)
Items.belongsTo(Category)

Items.hasOne(BasketItem)
BasketItem.belongsTo(Items)

module.exports = {
    User,
    Basket,
    BasketItem,
    Items,
    Category
}
```

basketRoutes.js

```
const Router = require('express');
const router = new Router();
const basketController = require('../controllers/basketController');
const authMiddleware = require('../middleware/authMiddleware');

router.post('/add', authMiddleware, basketController.addItem);
router.post('/remove', authMiddleware, basketController.removeItem);
router.get('/', authMiddleware, basketController.getBasket);

module.exports = router;
```

categoryRoutes.js

```
const Router = require('express')
const router = new Router()
const categoryController = require('../controllers/categoryController')
const checkRole = require('../middleware/checkRoleMilddleware')

router.post('/',checkRole('ADMIN'), categoryController.create)
router.get('/', categoryController.get)

module.exports = router
```

index.js(routes)

```
const Router = require('express')
const router = new Router()
const itemRouter = require('./itemRoutes')
const categoryRouter = require('./categoryRoutes')
const userRouter = require('./userRoutes')
const basketRouter = require('./basketRoutes')

router.use('/user', userRouter)
router.use('/item', itemRouter)
router.use('/category',categoryRouter)
router.use('/basket', basketRouter)

module.exports = router
```

itemRoutes.js

```
const Router = require('express')
const router = new Router()
const itemController = require('../controllers/itemController')
const checkRole = require('../middleware/checkRoleMilddleware')

router.post('/',checkRole('ADMIN'), itemController.create)
router.get('/', itemController.get)
router.get('/:id', itemController.getOne)

module.exports = router
```

userRoutes.js

```
const Router = require('express')
const router = new Router()
const userController = require('../controllers/userController')
const authMiddleware = require('../middleware/authMiddleware')

router.post('/registration', userController.registration)
router.post('/login', userController.login)
router.get('/auth',authMiddleware, userController.check)

module.exports = router
```

db.js

```
const {Sequelize} = require('sequelize')

module.exports = new Sequelize(
    process.env.DB_NAME,
    process.env.DB_USER,
    process.env.DB_PASSWORD,
    {
        dialect: 'postgres',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT
    }
)
```

.env

```
PORT=5000
DB_NAME=krispy_kreme
DB_USER=postgres
DB_PASSWORD=root
DB_HOST=localhost
DB_PORT=5432

SECRET_KEY=random_key
```

Index.js (server)

```
require('dotenv').config()
const express = require("express")
const sequelize = require('./db')
const models = require('./models/models')
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const errorHandler = require('./middleware/ErrorHandlingMiddleware')
const path = require('path')

const PORT = process.env.PORT || 5000

const app = express()
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname,'static')))
app.use(fileUpload({}))
app.use('/api',router)

//Errors, last middleware
app.use(errorHandler)

const start = async() => {
    try{
        await sequelize.authenticate()
        await sequelize.sync()
        app.listen(PORT, () => console.log('Server started on port ' +
PORT))
    }
    catch(e){
        console.log(e)
    }
}

start()
```

Додаток В

Програмні файли клієнтської частини

AppRouter.js

```
import React, {useContext} from 'react';
import {
    Routes,
    Route,
    Navigate
} from "react-router-dom";
import {authRoutes, publicRoutes} from "../routes";
import {SHOP_ROUTE} from "../utils/consts";
import {Context} from "../index";

const AppRouter = () => {
    const {user} = useContext(Context)

    return (
        <Routes>
            {user.isAuth && authRoutes.map(({path, Component}) =>
             <Route key={path} path={path} element={<Component />} />
            )}

            {publicRoutes.map(({path, Component}) =>
                <Route key={path} path={path} element={<Component />} />
            )}
            <Route path="*" element={<Navigate to={SHOP_ROUTE} replace
/>} />
        </Routes>
    );
};

export default AppRouter;
```

BillBoard.js

```
import React from 'react';
import {Col, Container, Row} from "react-bootstrap";
import '../index.css';
import {observer} from "mobx-react-lite";

const BillBoard = observer( () => {
    return (
        <div className="presentation">
            <Container>
                <Row className="justify-content-center">
                    <Col md={8} className="text-center">
                        {}
                    </Col>
                </Row>
            </Container>
        </div>
    );
}
```

```
);

export default BillBoard;
```

CartDropdown.js

```
import React, { useContext, useEffect } from 'react';
import { observer } from "mobx-react-lite";
import Order from './Order';
import { Context } from "../index";

const CartDropdown = observer(() => {
    const { item: itemStore } = useContext(Context);

    useEffect(() => {
        itemStore.fetchBasket(); // Завантаження кошика при монтуванні
компонента
    }, [itemStore]);

    const showOrders = () => {
        let summa = 0;
        itemStore.orders.forEach(item => summa +=
parseFloat(item.price));

        return (
            <div>
                {itemStore.orders.map(item => (
                    <Order key={item.id} onDelete={() =>
itemStore.deleteOrder(item.id)} item={item} />
                ))}
                <p className='summa'>Сумма : {new
Intl.NumberFormat().format(summa)}$</p>
            </div>
        );
    };

    const showNothing = () => {
        return (
            <div className='empty'>
                <h2>Нічого немає</h2>
            </div>
        );
    };

    return (
        <div className='shop-card'>
            {itemStore.orders.length > 0 ? showOrders() : showNothing()}
        </div>
    );
});

export default CartDropdown;
```

CategoryBar.js

```
import React, { useContext } from 'react';
import { Context } from "../index";
import { observer } from "mobx-react-lite";

const CategoryBar = observer(() => {
    const { item } = useContext(Context);

    const handleClick = (category) => {
        item.setSelectedCategory(category);
    };

    return (
        <div className="categories mt-3">
            {item.categories.map(category => (
                <div
                    onClick={() => handleClick(category)}
                    key={category.id}
                    className={`category-item ${item.selectedCategory.id
=== category.id ? 'active' : ''}`}
                >
                    {category.name}
                </div>
            ))}
        </div>
    );
});

export default CategoryBar;
```

Footer.js

```
import React from 'react'
import { FaInstagram } from "react-icons/fa";
import { FaFacebook } from "react-icons/fa";
import { FaTwitter } from "react-icons/fa";

export default function Footer() {
    return (
        <footer>
            <h3>All ruts are protected</h3>
            <div>
                <ul>
                    <li>
                        <a
href="https://www.facebook.com/KrispyKreme?locale=ru_RU" target="_blank"
rel="noreferrer" ><FaFacebook className='icon'/></a>
                    </li>
                    <li>
                        <a href="https://twitter.com/krispykreme"
target="_blank" rel="noreferrer"><FaTwitter className='icon'/></a>
                    </li>
                    <li>
                        <a href="https://www.instagram.com/krispykreme/"
```

```
target="_blank" rel="noreferrer"><FaInstagram className='icon'/></a>
                    </li>
                </ul>
            </div>
        </footer>
    )
}
```

Item.js

```
import React, { useContext } from 'react';
import { Image } from 'react-bootstrap';
import { Context } from '../index';

const Item = ({ item, onImageClick, onAdd }) => {
    const { item: itemStore } = useContext(Context);

    const handleAddToOrder = () => {
        itemStore.addToOrder(item); // Додати товар до кошика
    };

    return (
        <div className="item">
            <Image src={process.env.REACT_APP_API_URL + item.img}
alt={item.title} onClick={() => onImageClick(item)} />
            <h2>{item.title}</h2>
            <p>{item.desc}</p>
            <b>{item.price}$</b>
            <div className="addToCard" onClick={handleAddToOrder}>+</div>
        </div>
    );
};

export default Item;
```

ItemList.js

```
import React, { useContext, useState } from 'react';
import { observer } from 'mobx-react-lite';
import { Context } from '../index';
import ShowFullItem from './ShowFullItem';
import Item from "./Item";

const ItemList = observer(() => {
    const { item: itemStore } = useContext(Context);
    const [selectedItem, setSelectedItem] = useState(null);

    if (!Array.isArray(itemStore.items)) {
        return <div>No items available.</div>;
    }

    const filteredItems = itemStore.items.filter(
        item => !itemStore.selectedCategory.id || item.categoryId ===
```

```
itemStore.selectedCategory.id
    );

    const handleImageClick = (item) => {
        setSelectedItem(item);
    };

    const handleCloseFullItem = () => {
        setSelectedItem(null);
    };

    const handleAddToOrder = (item) => {
        itemStore.addToOrder(item);
    };

    return (
        <main>
            {filteredItems.map(item => (
                <Item
                    key={item.id}
                    item={item}
                    onImageClick={handleImageClick}
                    onAdd={handleAddToOrder}
                />
            ))}
            {selectedItem && (
                <ShowFullItem
                    item={selectedItem}
                    onAdd={handleAddToOrder}
                    onClose={handleCloseFullItem}
                />
            )}
        </main>
    );
});

export default ItemList;
```

NavBar.js

```
import React, {useContext, useState} from 'react';
import { Context } from "../index";
import { Button, Container, Nav, Navbar, NavLink } from "react-
bootstrap";
import {ABOUT_ROUTE, CONTACTS_ROUTE, LOGIN_ROUTE, SHOP_ROUTE} from
"../utils/consts";
import { observer } from "mobx-react-lite";
import {Link, useNavigate} from 'react-router-dom';
import '../index.css';
import CartDropdown from "./CartDropdown";
import {FaShoppingCart} from "react-icons/fa";

const NavBar = observer(({ cartItems, deleteOrder, addToOrder }) => {
    const { user } = useContext(Context);
    const navigate = useNavigate()
```

```
    const logOut = () => {
        user.setUser({});
        user.setIsAuth(false);
        localStorage.removeItem('token');
        navigate(LOGIN_ROUTE);
    };


    const [cardOpen, setCardOpen] = useState(false);

    const handleToggleCart = () => {
        setCardOpen(!cardOpen);
    };

    return (
        <Navbar bg="light" data-bs-theme="light">
            <Container>
                <Link to={SHOP_ROUTE} className={'logo'}>Krispy
Kreme</Link>
                <div className="shop-cart" onClick={handleToggleCart}>
                    <FaShoppingCart className={`shop-button ${cardOpen &&
'active'}`} />
                </div>
                {cardOpen && <CartDropdown />}
                {user.isAuth ? (
                    <Nav className="ml-auto">
                        <Button as={Link} to={ABOUT_ROUTE}
variant={"outline-dark"} className="m-lg-2">About Us</Button>
                        <Button as={Link} to={CONTACTS_ROUTE}
variant={"outline-dark"} className="m-lg-2">Contacts</Button>
                        <Button onClick={() => logOut()}
variant={"outline-dark"} className="m-lg-2">Out</Button>
                    </Nav>
                ) : (
                    <Nav className="ml-auto">
                        <Button as={Link} to={ABOUT_ROUTE}
variant={"outline-dark"} className="m-lg-2">About Us</Button>
                        <Button as={Link} to={CONTACTS_ROUTE}
variant={"outline-dark"} className="m-lg-2">Contacts</Button>
                        <Button as={Link} to={LOGIN_ROUTE}
variant={"outline-dark"} className="m-lg-2">Login</Button>
                    </Nav>
                )}
            </Container>
        </Navbar>
    );
});

export default NavBar;
```

Order.js

```
import React from 'react';
import { FaTrash } from 'react-icons/fa';
import { Image } from "react-bootstrap";
```

```
import { observer } from "mobx-react-lite"; // Імпортуємо observer

const Order = observer(({ item, onDelete }) => {
    return (
        <div className="order-item">
                <Image className={'img'}
src={process.env.REACT_APP_API_URL + item.img} alt={item.title} />
                    <h2>{item.title}</h2>
                    <b>{item.price}$</b>
                <FaTrash className='delete-item' onClick={() =>
onDelete(item.id)} />
        </div>
    );
});

export default Order;
```

ShowFullItem.js

```
import React from 'react';
import '../index.css'; // Підключаємо файли стилів

const ShowFullItem = ({ item, onAdd, onClose }) => {
    return (
        <div className='full-item'>
            <div className="card">
                <img src={process.env.REACT_APP_API_URL + item.img}
alt={item.title} />
                <h2>{item.title}</h2>
                <p>{item.desc}</p>
                <b>{item.price}$</b>
                <div className='buttons'>
                    <div className='addToCard' onClick={() =>
onAdd(item)}>+</div>
                    <button onClick={onClose}>Закрити</button>
                </div>
            </div>
        </div>
    );
};

export default ShowFullItem;
```

index.js(http)

```
import axios from "axios";
import config from "bootstrap/js/src/util/config";

const $host = axios.create({
    baseURL: process.env.REACT_APP_API_URL || 'http://localhost:5000'
})

const $authHost = axios.create({
```

```
    baseURL: process.env.REACT_APP_API_URL || 'http://localhost:5000'
})

const authInterceptor = config => {
    config.headers.authorization = `Bearer
${localStorage.getItem('token')}`
    return config
}

$authHost.interceptors.request.use(authInterceptor)

export {
    $host,
    $authHost
}
```

itemApi.js

```
import {$host} from "./index";

export const gettingCategory = async () => {
    try {
        const { data } = await $host.get('api/category');
        return data;
    } catch (error) {
        console.error("Error fetching categories:", error);
        throw error;
    }
};

export const gettingItems = async () => {
    try {
        const { data } = await $host.get('api/item');
        return data;
    } catch (error) {
        console.error("Error fetching categories:", error);
        throw error;
    }
};

export const fetchBasketItemsFromAPI = async () => {
    try {
        const response = await fetch('/api/basket');
        if (!response.ok) {
            const errorText = await response.text();
            throw new Error(`Failed to fetch basket items:
${errorText}`);
        }
        return await response.json();
    } catch (error) {
        console.error('Error fetching basket items:', error);
        return [];
    }
};
```

userApi.js

```
import { $host, $authHost } from './index';
import { jwtDecode } from 'jwt-decode';

export const registration = async (email, password) => {
    const { data } = await $host.post('api/user/registration', { email,
password, role: 'ADMIN' });
    console.log('Registration response:', data);
    localStorage.setItem('token', data.token);
    return jwtDecode(data.token);}

export const login = async (email, password) => {
    const { data } = await $host.post('api/user/login', { email, password
});
    console.log('Login response:', data);
    localStorage.setItem('token', data.token);
    return jwtDecode(data.token);
}

export const check = async () => {
    const token = localStorage.getItem('token'); // Отримання токену з
localStorage
    if (!token) {
        throw new Error('No token found');
    }

    const { data } = await $authHost.get('api/user/auth', {
        headers: {
            Authorization: `Bearer ${token}` // Додавання токену у
заголовок Authorization
        }
    });
    console.log('Auth check response:', data);
    return jwtDecode(data.token);
}
```

AboutUs.js

```
import React from 'react';
import { Container, Row, Col } from 'react-bootstrap';

const AboutUs = () => {
    return (
        <Container className="mt-5">
            <Row>
                <div className={'about_us'}>
                    <h1>Про нас</h1>
                    <p>
                        Ласкаво просимо до Krispy Kreme! Ми раді вітати
вас у нашому світі солодощів.
                        Ми створюємо найсмачніші пончики, використовуючи
тільки найкращі інгредієнти і вкладаючи всю нашу любов у кожен виріб.
                        Наша мета - зробити кожен ваш день трішки
солодшим і радіснішим. Відвідайте нас і відчуйте цей неповторний смак!
```

```
                    </p>
                </div>
            </Row>
        </Container>
    );
};

export default AboutUs;
```

Auth.js

```
import React, {useContext, useState} from 'react';
import { Button, Card, Col, Container, Form, Row } from "react-
bootstrap";
import {LOGIN_ROUTE, REGISTRATION_ROUTE, SHOP_ROUTE} from
"../utils/consts";
import {NavLink, redirect, useLocation} from "react-router-dom";
import {login, registration} from "../http/userAPI";
import {observer} from "mobx-react-lite";
import {Context} from "../index";
import {useNavigate} from "react-router-dom";

const Auth = observer( () => {
    const {user} = useContext(Context)
    const location = useLocation();
    const isLogin = location.pathname === LOGIN_ROUTE;

    const navigate = useNavigate();
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const click = async () => {
            try {
                let data;
                if (isLogin) {
                    data = await login(email, password);
                } else {
                    data = await registration(email, password);
                }

                console.log('Auth click user data:', data);
                user.setUser(data);
                user.setIsAuth(true);
                navigate('/');
            } catch (e) {
                console.error('Auth error:', e);
                alert(e.response.data.message);
            }
        };
    return (
        <Container
            className="d-flex justify-content-center align-items-center"
            style={{ height: '100vh' }}
        >
            <Card style={{ width: 400 }} className="shadow p-4">
                <h2 className="text-center mb-4">{isLogin ? "Авторизація"
: "Реєстрація"}</h2>
```

```
                <Form>
                    <Form.Group controlId="formBasicEmail" className="mb-
3">
                        <Form.Control type="email" placeholder="Введіть
email..."
                        value={email}
                        onChange={e => setEmail(e.target.value)}
                        />
                    </Form.Group>

                    <Form.Group controlId="formBasicPassword"
className="mb-3">
                        <Form.Control type="password"
placeholder="Введіть пароль..."
                                      value={password}
                                      onChange={e =>
setPassword(e.target.value)}
                        />
                    </Form.Group>

                    {!isLogin && (
                        <Form.Group controlId="formBasicConfirmPassword"
className="mb-3">
                            <Form.Control type="password"
placeholder="Підтвердіть пароль..." />
                        </Form.Group>
                    )}

                    <Row className="d-flex justify-content-between align-
items-center mb-3">
                        {isLogin ? (
                            <Col>
                                Немає аккаунта? <NavLink
to={REGISTRATION_ROUTE}>Реєстрація</NavLink>
                            </Col>
                        ) : (
                            <Col>
                                Є аккаунт? <NavLink
to={LOGIN_ROUTE}>Увійти</NavLink>
                            </Col>
                        )}
                        <Col className="text-end">
                            <Button variant="dark" type="submit"
onClick={click}>
                                {isLogin ? "Увійти" : "Зареєструватися"}
                            </Button>
                        </Col>
                    </Row>
                </Form>
            </Card>
        </Container>
    );
}
);

export default Auth;
```

Contacts.js

```
import React from 'react';
import { Container, Row, Col, Form, Button, Card } from 'react-
bootstrap';
import '../index.css';

const Contacts = () => {
    return (
        <Container className="mt-5">
            <Row className="justify-content-md-center">
                <Col md={8}>
                    <Card className="contact-card">
                        <Card.Body>
                            <h1 className="text-center mb-
4">Контакти</h1>
                            <p className="text-center">Якщо у вас є
питання або пропозиції, будь ласка, зв'яжіться з нами, використовуючи
форму нижче:</p>
                            <Form>
                                <Form.Group controlId="formName"
className="mb-3">
                                    <Form.Label>Ім'я</Form.Label>
                                    <Form.Control type="text"
placeholder="Введіть ваше ім'я" />
                                </Form.Group>

                                <Form.Group controlId="formEmail"
className="mb-3">
                                    <Form.Label>Email</Form.Label>
                                    <Form.Control type="email"
placeholder="Введіть ваш email" />
                                </Form.Group>

                                <Form.Group controlId="formMessage"
className="mb-3">
                                    <Form.Label>Повідомлення</Form.Label>
                                    <Form.Control as="textarea" rows={3}
placeholder="Ваше повідомлення" />
                                </Form.Group>

                                <Button variant="dark" type="submit"
className="w-100">
                                    Надіслати
                                </Button>
                            </Form>
                        </Card.Body>
                    </Card>
                </Col>
            </Row>
            <Row className="justify-content-md-center mt-5">
                <Col md={8}>
                    <Card className="contact-card">
                        <Card.Body>
                            <h2 className="text-center mb-4">Наші
контакти</h2>
                            <p><strong>Адреса:</strong> 370 Knollwood St,
Winston-Salem, NC 27103, USA</p>
```

```
                            <p><strong>Телефон:</strong> +1 800-457-
4779</p>
                            <p><strong>Email:</strong>
info@krispykreme.com</p>
                            <h2 className="text-center mt-4 mb-4">Робочі
години</h2>
                            <p><strong>Понеділок - П'ятниця:</strong>
8:00 - 17:00</p>
                            <p><strong>Субота:</strong> 8:00 - 12:00</p>
                            <p><strong>Неділя:</strong> Закрито</p>
                        </Card.Body>
                    </Card>
                </Col>
            </Row>
        </Container>
    );
};

export default Contacts;
```

Shop.js

```
import React, { useContext, useEffect } from 'react';
import { Container } from "react-bootstrap";
import BillBoard from "../components/BillBoard";
import CategoryBar from "../components/CategoryBar";
import ItemList from "../components/ItemList";
import { observer } from "mobx-react-lite";
import { Context } from "../index";
import { gettingCategory, gettingItems } from "../http/itemAPI";

const Shop = observer(() => {
    const { item } = useContext(Context);

    useEffect(() => {
        gettingCategory().then(data => item.setCategories(data));
        gettingItems().then(data => item.setItems(data.rows));
    }, []);


    useEffect(() => {
        console.log("Selected Category:", item.selectedCategory);
        console.log("Filtered Items:",
item.getItemsByCategory(item.selectedCategory.id));
    }, [item.selectedCategory, item.items]);

    const filteredItems =
item.getItemsByCategory(item.selectedCategory.id);

    return (
        <Container>
            <BillBoard />
            <CategoryBar />
            <ItemList />
        </Container>
    );
```

```
});

export default Shop;
```

ItemStore.js

```
import {fetchBasketItemsFromAPI, gettingItems} from "../http/itemAPI";
import { makeAutoObservable, runInAction, action } from "mobx";

class ItemStore {
    constructor() {
        this._categories = [];
        this._items = [];
        this.orders = [];
        this._selectedCategory = {};
        makeAutoObservable(this, {
            addToOrder: action,
            deleteOrder: action,
            fetchBasket: action
        });
    }

    get categories() {
        return this._categories;
    }

    setCategories(value) {
        this._categories = value;
    }

    get items() {
        return this._items;
    }

    setItems(value) {
        this._items = value;
    }

    get selectedCategory() {
        return this._selectedCategory;
    }

    setSelectedCategory(value) {
        this._selectedCategory = value;
    }

    addToOrder(item) {
        const isInArray = this.orders.some(el => el.id === item.id);
        if (!isInArray) {
            this.orders = [...this.orders, item];
        }
    }

    deleteOrder(id) {
        this.orders = this.orders.filter(order => order.id !== id);
```

```
    }

    async fetchBasket() {
        try {
            const basketItems = await fetchBasketItemsFromAPI();
            runInAction(() => {
                this.orders = basketItems;
            });
        } catch (error) {
            console.error("Failed to fetch basket items", error);
        }
    }

    getItemsByCategory(categoryId) {
        return this._items.filter(item => item.categoryId ===
categoryId);
    }

}

export default ItemStore;
```

UserStore.js

```
import {makeAutoObservable} from "mobx";

export default class UserStore{
    constructor() {
        this._isAuth = false
        this._user = {}
        makeAutoObservable(this)
    }

    setIsAuth(bool){
        this._isAuth = bool
    }

    setUser(user){
        this._user = user
    }

    get isAuth() {
        return this._isAuth;
    }

    get user() {
        return this._user;
    }
}
```

consts.js

```
export const ADMIN_ROUTE = '/admin'
export const LOGIN_ROUTE = '/login'
export const REGISTRATION_ROUTE = '/registration'
```

```
export const SHOP_ROUTE = '/'
export const BASKET_ROUTE = '/basket'

export const ABOUT_ROUTE = '/about'

export const CONTACTS_ROUTE = '/contacts'
```

App.js

```
import React, { useContext, useEffect, useState } from 'react';
import { BrowserRouter } from 'react-router-dom';
import { Spinner } from 'react-bootstrap';
import NavBar from './components/NavBar';
import AppRouter from './components/AppRouter';
import { observer } from 'mobx-react-lite';
import { Context } from './index';
import { check } from './http/userAPI';
import Footer from "./components/Footer";

const App = observer(() => {
    const { user } = useContext(Context);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        const fetchUserData = async () => {
            try {
                const token = localStorage.getItem('token'); // Отримуємо
токен з localStorage
                if (!token) {
                    throw new Error('No token found'); // Якщо токен
відсутній, генеруємо помилку
                }

                await check(); // Перевіряємо токен на сервері
                user.setUser(true);
                user.setIsAuth(true);
            } catch (error) {
                console.error(error.message);
            } finally {
                setLoading(false);
            }
        };

        fetchUserData();
    }, [user]);

    if (loading) {
        return <Spinner animation={'grow'} />;
    }

    return (
        <BrowserRouter>
            <NavBar />
            <AppRouter />
            <Footer />
        </BrowserRouter>
    );
});
```

```
export default App;
```

index.js

```
import React, {createContext} from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import UserStore from "./store/UserStore";
import ItemStore from "./store/ItemStore";

import 'bootstrap/dist/css/bootstrap.min.css';

export const Context = createContext(null)

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

    <Context.Provider value={{
        user: new UserStore(),
        item: new ItemStore()
    }}>
        <App />
    </Context.Provider>
);
```

routes.js

```
import Admin from './pages/Admin';
import {
    ABOUT_ROUTE,
    ADMIN_ROUTE,
    BASKET_ROUTE,
    CONTACTS_ROUTE,
    LOGIN_ROUTE,
    REGISTRATION_ROUTE,
    SHOP_ROUTE,
    ITEM_PAGE
} from './utils/consts';
import Basket from "./pages/Basket";
import Shop from './pages/Shop';
import Auth from "./pages/Auth";
import AboutUs from "./pages/AboutUs";
import Contacts from "./pages/Contacts";
import ItemPage from "./pages/ItemPage";

export const authRoutes = [
    {
        path: ADMIN_ROUTE,
        Component: Admin
    },
    {
        path: BASKET_ROUTE,
        Component: Basket
    },
]
```

```
export const publicRoutes = [
    {
        path: SHOP_ROUTE,
        Component: Shop
    },
    {
        path: LOGIN_ROUTE,
        Component: Auth
    },
    {
        path: REGISTRATION_ROUTE,
        Component: Auth
    },
    {
        path: ABOUT_ROUTE,
        Component: AboutUs
    },
    {
        path: CONTACTS_ROUTE,
        Component: Contacts
    },
    {
        path: ITEM_PAGE,
        Component: ItemPage
    }
]
```

index.css

```
@import
url('https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,300.
.600;1,100..900&display=swap');

body {
  background: #fff;
  color: #222;
  font-family: 'Montserrat', sans-serif;
  font-weight: 300;

}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}

.logo {
  font-weight: 600;
  font-size: 20px;
  color: black;
  text-decoration: none;
}

.presentation, .about_us {
  margin: 50px 0;
  background: url('./img/fon1.jpg') no-repeat;
  width: 100%;
```

```
  height: 500px;
  background-size: cover;
  background-position: center center;
  background-blend-mode: multiply;
  background-color: #898484;
  position: relative;
  display: flex;
  align-items: center;
  justify-content: center;
  text-align: left;
  color: white;
  border-radius: 20px;
}

.presentation::after {
  content: 'Запрошуємо вас у солодкий світ пончиків!';
  position: absolute;
  top: 100px;
  left: 50px;
  width: 700px;
  font-size: 40px;
  font-weight: 600;
  color: #fff;
}

.presentation::before {
  content: 'Ласкаво просимо до нашого онлайн магазину, де мрії стають
солодкими реальностями. У нас ви знайдете широкий вибір різноманітних
пончиків, приготованих з використанням тільки найкращих інгредієнтів та з
любовю.';
  position: absolute;
  top: 250px;
  left: 50px;
  width: 600px;
  font-size: 20px;
  font-weight: 500;
  color: #fff;
}

.about_us h1{
  position: absolute;
  top: 100px;
  left: 300px;
  width: 700px;
  font-size: 40px;
  font-weight: 600;
  color: #fff;
  text-align: center;
}

.about_us p{
  position: absolute;
  top: 250px;
  left: 350px;
  width: 600px;
  font-size: 20px;
  font-weight: 500;
  color: #fff;
```

```
  text-align: center;
}
.categories {
  margin-top: 20px;
}

.category-item {
  display: inline-block;
  background: #f2f2f2;
  border-radius: 50px;
  padding: 10px 20px;
  margin-bottom: 25px;
  margin-right: 15px;
  cursor: pointer;
  border: 1px solid transparent;
  transition: all 500ms ease;
}

.category-item:hover {
  border-color: silver;
  transform: scale(1.1);
}

main {
  display: flex;
  width: 100%;
  flex-wrap: wrap;
  justify-content: space-between;
  margin-top: 50px;
}

main .item {
  width: 30%;
  margin-bottom: 50px;
  background: #f5f5f5;
  overflow: hidden;
  position: relative;
  padding-bottom: 20px;
  border-radius: 10px;
}

main .item img,
.full-item img {
  width: 100%;
  height: auto;
  border-radius: 10px 10px 0 0;
  transition: transform 500ms ease;
  cursor: pointer;
}

main .item img:hover,
.full-item img:hover {
  transform: scale(1.05);
}

main h2, main p, .full-item h2, .full-item p {
  margin: 10px 20px;
  color: #333;
```

```
}

main b, .full-item b {
  margin: 10px 20px;
  color: #5fa36a;
}

main .addToCard,
.full-item .addToCard {
  position: absolute;
  right: 20px;
  bottom: 20px;
  background: #ca5252;
  width: 35px;
  height: 35px;
  text-align: center;
  line-height: 35px;
  color: #fff;
  border-radius: 50%;
  cursor: pointer;
  font-weight: 600;
  transition: transform 500ms ease;
}

main .addToCard:hover,
.full-item .addToCard:hover {
  transform: scale(1.5) translateY(-5px);
}

 .shop-card {
  position: absolute;
  top: 50px;
  right: 0;
  width: 450px;
  background:#fafafa;
  -webkit-box-shadow: 9px 10px 41px -6px rgba(150,163,162,0.55);
  -moz-box-shadow: 9px 10px 41px -6px rgba(150,163,162,0.55);
  box-shadow: 9px 10px 41px -6px rgba(150,163,162,0.55);
  z-index: 1000;

  padding: 20px;
  padding-bottom: 0;
}

 .shop-button {
  float: left;
   margin-left: 800px;
  cursor: pointer;
  transition: color, transform 500ms ease;
   z-index: 30000;
}

 .shop-button:hover,
 .shop-button.active {
  color: #dc3d3d;
  transform: scale(1.5);
}
```

```
.shop-card .empty h2 {
  font-size: 20px;
  margin-bottom: 20px;
  margin-left: 35%;
  color: #797979;
}

.shop-card .order-item {
  width: 100%;
  float: left;
  margin-bottom: 20px;
}

.shop-card .order-item .img {
  width: 70px;
  float: left;
  margin-right: 20px;
}

.shop-card .order-item h2 {
  font-size: 20px;
  margin-bottom: 10px;
}

.shop-card .order-item p {
  color: #797979;
  font-weight: 600;
}

.shop-card .order-item .delete-item {
  color: #ca5252;
  float: right;
  position: relative;
  top: -25px;
  cursor: pointer;
  transition: color, transform 500ms ease;
}

 .shop-card .order-item .delete-item:hover {
  color: #d83030;
  transform: scale(1.5);
}
 .shop-card .summa {
  float: left;
  width: 100%;
  font-weight: 600;
  font-size: 20px;
  margin-bottom: 20px;
}

footer {
  position: relative;
  text-align: center;
  margin-top: 100px;
}


Footer ul {
```

```
  position:absolute;
  top:50%;
  left:50%;
  transform: translate(-50%, -50%);
  margin-top: 50px;
  padding:0;
  display:flex;
}

Footer ul li {
  list-style: none;
}

Footer ul li a {
  position: relative;
  width:50px;
  height:50px;
  display:block;
  text-align:center;
  margin:0 10px;
  border-radius: 50%;
  padding: 6px;
  box-sizing: border-box;
  text-decoration:none;
  box-shadow: 0 10px 15px rgba(0,0,0,0.3);
  background: linear-gradient(0deg, #ddd, #fff);
  transition: .5s;
}

Footer ul li a:hover {
  box-shadow: 0 2px 5px rgba(0,0,0,0.3);
}

Footer ul li a .icon {
  position: relative;
  width: 100%;
  height: 100%;
  color: #222;
}

.contact-card {
  border: none;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
  border-radius: 8px;
  padding: 20px;
}

.contact-card h1, .contact-card h2 {
  font-family: 'Arial', sans-serif;
  color: #333;
}

.contact-card p {
  font-family: 'Arial', sans-serif;
  color: #555;
}

.contact-card .form-control {
```

```
  border-radius: 5px;
  padding: 10px;
}

.contact-card .btn {
  background-color: #333;
  color: #fff;
  border-radius: 5px;
  padding: 10px;
  font-size: 16px;
}

.contact-card .btn:hover {
  background-color: #555;
}
.item {
  position: relative;
  margin-bottom: 20px;
}

.item img {
  width: 100%;
  cursor: pointer;
}


.item {
  width: 30%;
  margin-bottom: 50px;
  background: #f5f5f5;
  overflow: hidden;
  position: relative;
  padding-bottom: 20px;
}

.item img {
  width: 100%;
  height: auto;
  border-radius: 10px 10px 0 0;
  transition: transform 500ms ease;
  cursor: pointer;
}

.item img:hover {
  transform: scale(1.05);
}

.full-item {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 10;
}
```

```
.card {
  width: 60%;
  max-width: 500px;
  background-color: rgba(255, 255, 255, 0.9);
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0px 0px 200px rgba(35, 32, 32, 0.62);
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
}

.card img {
  width: 100%;
  height: auto;
  border-radius: 10px;
  margin-bottom: 20px;
  transition: transform 500ms ease;
  cursor: pointer;
}

.card img:hover {
  transform: scale(1.05);
}

.card h2,
.card p {
  margin: 10px 0;
  color: #333;
}

.card b {
  margin: 10px 0;
  color: #5fa36a;
}

.buttons {
  display: flex;
  justify-content: space-between;
  margin-top: auto;
  width: 100%;
}

.buttons button {
  background: #ca5252;
  color: #fff;
  padding: 10px 20px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-weight: 600;
  transition: transform 500ms ease;
}

.buttons button:hover {
  background-color: #333;
```

```
  transform: scale(1.05);
}
```

.env

```
REACT_APP_API_URL='http://localhost:5000/'
```