МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА

ФАКУЛЬТЕТ ЕКОНОМІКИ І ПІДПРИЄМНИЦТВА

Кафедра інформаційних технологій

Допущено до захисту

«_____» _____ 2023 р.

Зав. кафедри, к.т.н., доцент

_____ **Ліщук Р.І**.

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ АДМІНІСТРУВАННЯ ВЕБ-ДОДАТКУ З ВІЗУАЛІЗАЦІЄЮ ДАНИХ

Виконав: студент 2 курсу, 21-мкн групи
Освітньо-професійної програми «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
Бухенко Артур Олександрович
керівник: д.т.н., професор
Кучерук Володимир Юрійович

Умань – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

УМАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ САДІВНИЦТВА

Факультет економіки і підприємництва
Кафедра інформаційних технологій
Назва ОПП "Комп'ютерні науки"
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (ініціали, прізвище)

«____» _____2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Студенту Бухенку Артуру Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна система для адміністрування веб-додатку з візуалізацією даних

керівник роботи Кучерук Володимир Юрійович, д.т.н., професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від "___"_____2023 року №_____

2. Строк подання студентом проекту (роботи)_____
3. Вихідні дані до проекту (роботи) Технології програмування – CSS, Node.js. JavaScript, Середовище написання коду Microsoft Visual Code.
4. Перелік питань, які потрібно розробити Аналіз предметної області, реалізація бази даних, реалізація інформаційної системи
5. Перелік графічного матеріалу
65 рисунків, 31 джерело та 7 додатків

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми кваліфікаційної роботи		
2	Одержання індивідуального завдання		
3	Складання календарного плану написання кваліфікаційної роботи		
4	Підготовка до виконання кваліфікаційної роботи - підбір та вивчення літератури - участь у виконанні науково-дослідних робіт - виконання курсових робіт - інші заходи		
5	Уточнення теми кваліфікаційної роботи та календарного плану-графіка, виходячи зі специфіки базового підприємства, установи		
6	Підготовка: - першого розділу - другого розділу - третього розділу - висновків		
7	Доопрацювання кваліфікаційної роботи з урахування зауважень наукового керівника		
8	Написання та оформлення роботи в остаточному варіанті		
9	Одержання відгуку наукового керівника		
10	Перевірка роботи на академічний плагіат		
11	Попередній захист роботи на кафедрі		
12	Затвердження роботи завідувачем кафедри		
13	Одержання рецензії		
14	Захист дипломної роботи		

Студент _____
(прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська кваліфікаційна робота містить 68 сторінок, 65 рисунків та 7 додатків.

Темою цієї кваліфікаційної роботи є: “Інформаційна система для адміністрування веб-додатку з візуалізацією даних”.

Для цієї кваліфікаційної роботи було створено веб-додаток, а саме, невеличкий інтернет-магазин одягу, для якого було розроблено адміністративну панель з основними функціями для керування цим магазином.

Також для кожної функції були присвоєні як публічні, так і приватні API ключі, які надають змогу отримувати легкий доступ до цих функцій в програмному вигляді.

Анотація

Головною метою було створення мінімалістичної адміністративної панелі, для легкого керування веб-додатком, а в нашому випадку, інтернет магазином.

Ключові слова: Адміністративна панель, адміністрування, керування, функції, веб-магазин, Next.js, Prisma, Clerk, веб-додаток, PlanetScale, JavaScript.

The summary

The main goal was to create a minimalist administrative panel for easy management of a web application, and in our case, an online store.

Keywords: Admin Panel, Administration, Management, Features, Web Store, Next.js, Prisma, Clerk, Web Application, PlanetScale, JavaScript.

ЗМІСТ

ВСТУП 6
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 7
1.1. Дослідження актуальності проблеми 7
1.2 . Аналіз продуктів-аналогів 8
1.3. Мета дослідження 12
РОЗДІЛ 2. ВИЗНАЧЕННЯ ЗАДАЧІ І МЕТОДИ ЇХ РЕАЛІЗАЦІЇ 13
2.1 Визначення задачі 13
2.2 Методи реалізації 13
2.3 База Даних 15
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ 19
3.1 Програмна реалізація 19
3.2 З’єднання з базой даних 29
3.3 Створення веб-додатку 40
ВИСНОВОК 51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 52
Додаток А
База даних 56
Додаток Б
Статистика магазину 60
Додаток В
Створення категорій 62
Додаток Г
Загальна кількість продаж 66
Додаток Д
Загальна сума продаж 67
Додаток Ж
Графік продаж 68
Додаток З
Статистика кількості товару 70

ВСТУП

Актуальність веб-додатків в сучасному світі надзвичайно висока через їхню важливу роль у різноманітних аспектах життя суспільства, бізнесу та технологій. Веб-додатки, які доступні через інтернет, забезпечують користувачам широкий спектр можливостей та впливають на різні сфери нашого життя. Шопінг на OLX, прослуховування музики в Spotify, перегляд погоди на Sinoptik – це все є веб-додатки, які б не могли працювали без адміністрування.

Адміністрування веб-додатком є складним процесом, що включає в себе координацію різноманітних аспектів, спрямованих на оптимальне функціонування та розвиток цифрового продукту в онлайн-середовищі.

Цей процес передбачає організацію ресурсів, роботу з командами, стратегічне планування та неперервну оптимізацію для досягнення успіху та відповідності потребам користувачів.

У сучасному цифровому світі, де об'єми даних несперечно ростуть, розробка ефективних інформаційних систем стає надзвичайно важливою. Динамічний розвиток веб-додатків і платформ вимагає не лише стабільної та надійної системи адміністрування, але й можливості візуалізації величезних обсягів даних для прийняття обґрунтованих управлінських рішень.

Метою цієї кваліфікаційної роботи є спростити адміністрування веб-додатком шляхом створення власної адміністративної панелі яка має тільки основні функції для цього.

Наукова новизна полягає в можливості власнику магазину легко керувати своїм веб-додатком, самому змінювати банера і категорії, додавати товар в магазин та переглядати статистику продаж і так далі, і це все в одному місці.

Практичне значення роботи полягається в створені інформаційної системи адміністрування з основними функціями для керування, які були визначені після аналізу продуктів-аналогів, та створення простго веб-додатку для цієї системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Дослідження актуальності проблеми

Інформаційна система для адміністрування веб-додатком — це програмне забезпечення, яке дозволяє керувати і контролювати різні аспекти веб-додатка. Це може включати управління контентом, користувачами, даними, налаштуваннями безпеки та іншими параметрами, пов'язаними з ефективною експлуатацією та підтримкою веб-сайту або додатку.

Головне значення адміністрування веб-додатків виявляється у декількох ключових аспектах:

- Стабільність та надійність: Адміністрування дозволяє підтримувати веб-додатки в робочому стані, запобігаючи збоям та забезпечуючи їхню безперервну роботу. Це особливо важливо для бізнес-середовищ, де доступність та безперебійна робота веб-додатків є критичними.
- Безпека: Адміністрування забезпечує захист веб-додатків від потенційних загроз, таких як кібератаки, віруси, зловживання даними тощо. Це включає в себе виявлення вразливостей та їхнє усунення для збереження конфіденційності та цілісності даних.
- Ефективність та продуктивність: Адміністрування спрямоване на оптимізацію роботи веб-додатків, забезпечуючи їхню високу продуктивність та швидкодію. Це включає управління ресурсами, виявлення й усунення факторів, що можуть сповільнювати роботу системи.
- Управління даними: Адміністрування забезпечує ефективне зберігання, обробку та передачу даних у веб-додатках. Це охоплює не лише технічні аспекти, але й розробку стратегій збереження та аналізу даних для оптимального використання.

Підтримка та розвиток: Адміністрування включає в себе постійну підтримку веб-додатків, їх оновлення та розвиток. Це означає впровадження нового функціоналу, виправлення помилок та вдосконалення інтерфейсу для кращого користувацького досвіду. Для цііє роботи в якості веб-додатку було обрано веб-магазин одягу.

У сучасному світі електронної торгівлі веб-магазини відіграють ключову роль у способі, яким ми здійснюємо покупки та взаємодіємо з товарами та послугами. Адміністрування таких платформ вимагає особливої уваги до технічної складності та обробки великих обсягів даних, які створюються та оброблюються щоденно.

1.2. Аналіз продуктів-аналогів

На сьогоднішній день є багато сервісів для адміністрування веб-додатками, системи різняться інтерфейсом но загалом функціонал в них однаковий, ось приклад декількох адмін-панелей:

- WooCommerce (WordPress): Це адмінпанель для управління електронним магазином на базі WordPress. Вона дозволяє керувати товарами, замовленнями, аналізувати продажі та керувати користувачами (рис 1.1).

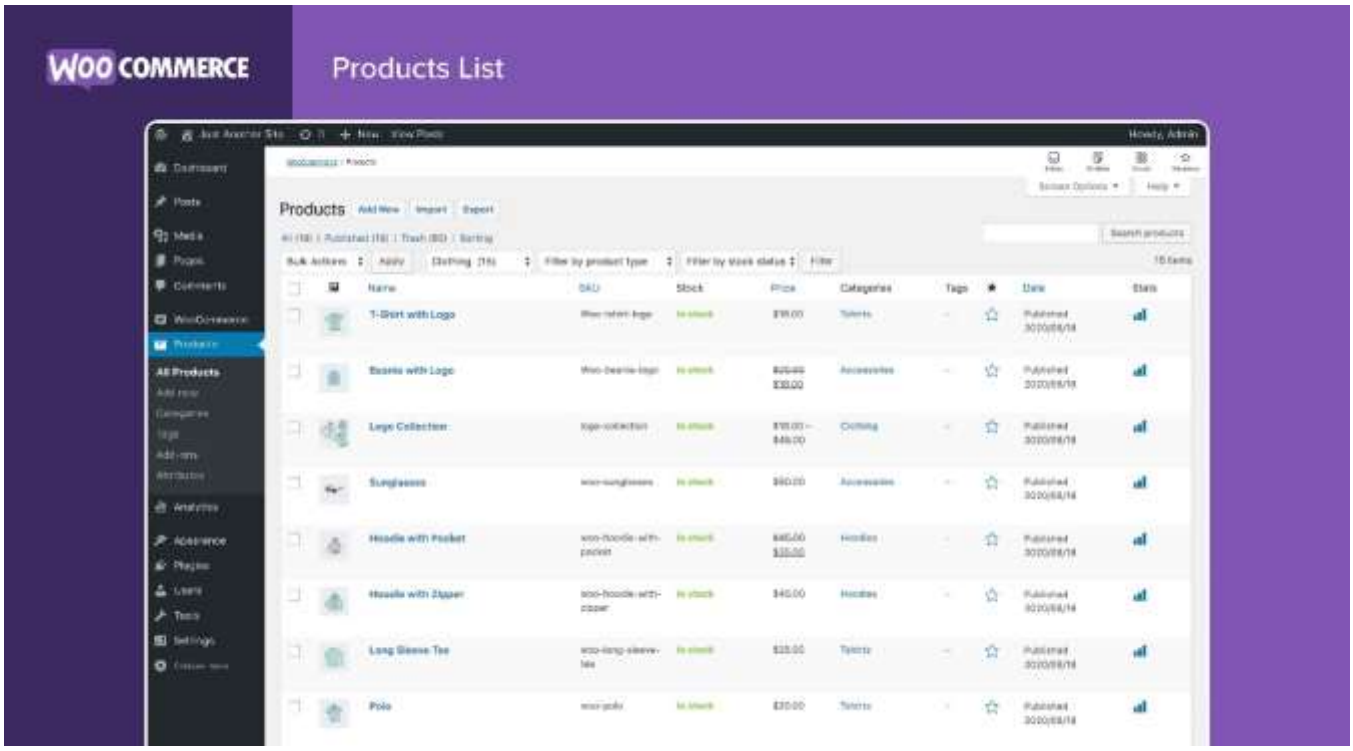


Рисунок 1.1 – Сторінка з списком товарів в адмін-панелі WooCommerce

Функції:

- Управління товарами: Додавання, редагування, видалення товарів, встановлення цін, налаштування атрибутів.
- Замовлення і оплата: Перегляд і обробка замовлень, керування оплатами, налаштування методів доставки.
- Аналітика: Отримання звітів про продажі, відвідуваність, популярність товарів.
- Керування користувачами: Реєстрація користувачів, налаштування облікових записів покупців.

- Magento Admin: Це адміністративна панель для управління веб-магазином на основі платформи Magento. Вона надає інтерфейс для додавання, редагування та видалення товарів, керування замовленнями, аналітики продажів та управління клієнтами.

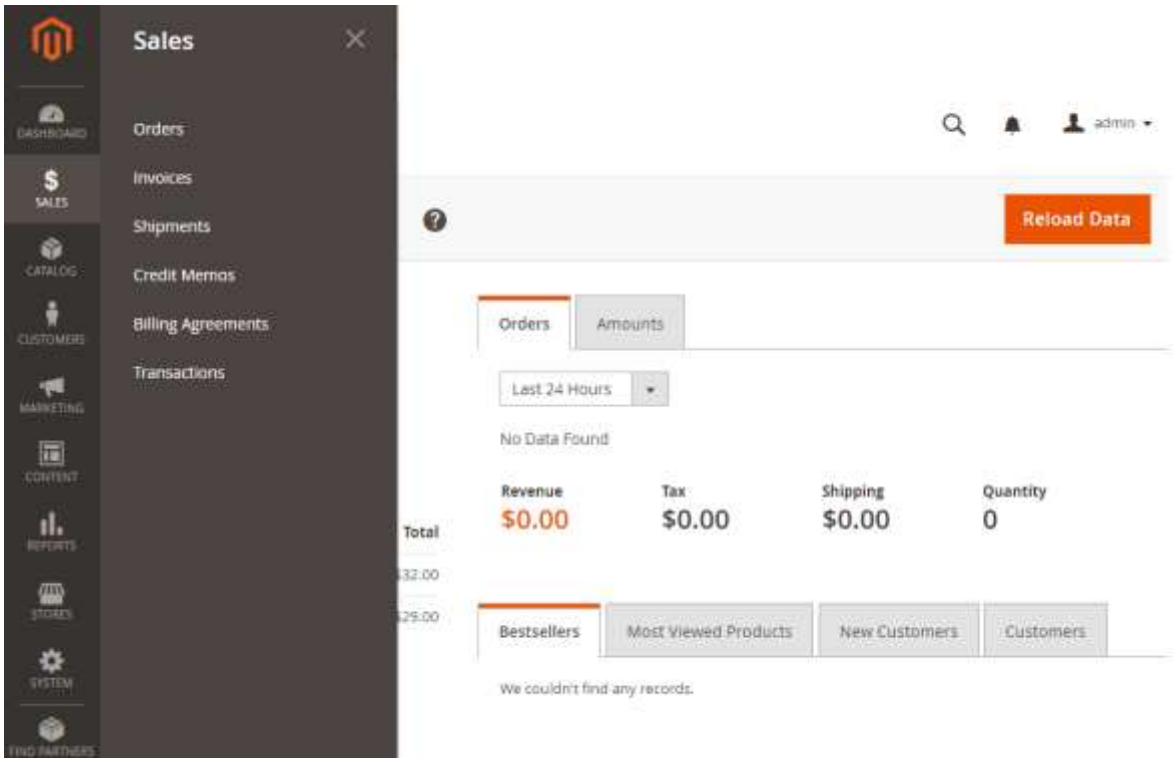


Рисунок 1.2 – Сторінка з статистикою продаж в адмін-панелі Magento Admin

Функції:

- Керування товарами: Додавання, видалення та редагування товарів, установка параметрів, описів, фотографій і цін.

- Управління замовленнями: Перегляд, обробка та відстеження замовлень, створення звітів.

- Аналітика і звіти: Отримання даних про продажі, відвідуваність магазину, поведінку покупців.

- Управління клієнтами: Керування профілями клієнтів, зв'язок з ними, налаштування акцій, знижок.

- Панель адміністратора: Налаштування адміністративних прав доступу, керування працівниками магазину.

- Shopify Admin: Це інтерфейс управління веб-магазином на платформі Shopify. Його функціонал включає керування товарами, замовленнями, аналітику продажів та можливості маркетингу.

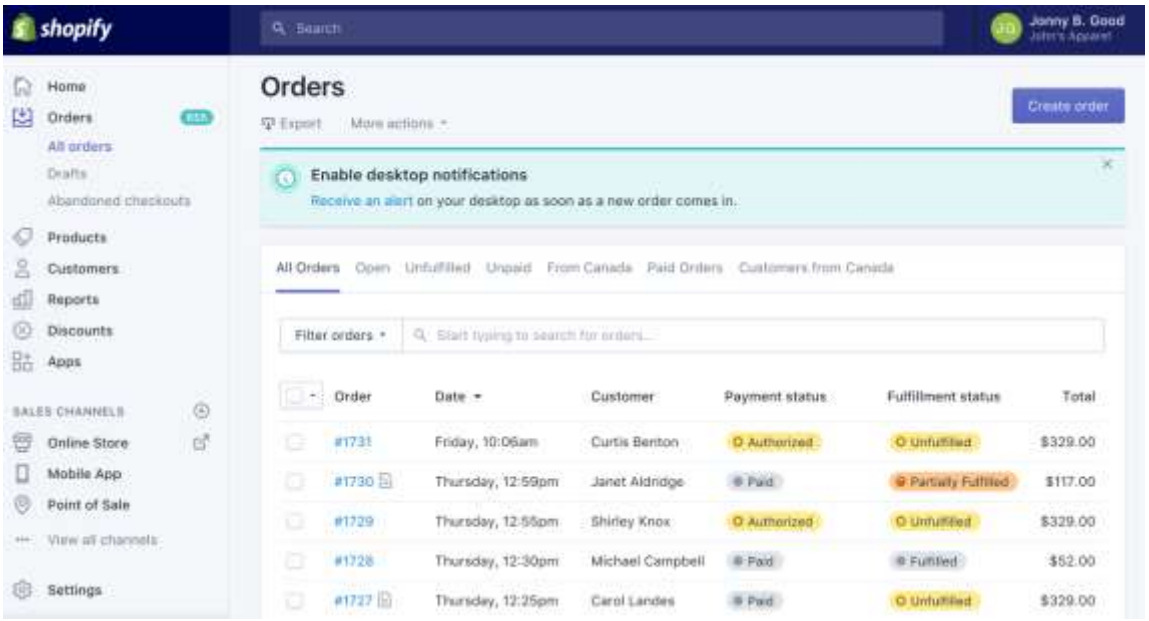


Рисунок 1.3 – Сторінка замовлень в адмін-панелі Shopify Admin

Функції:

- Керування товарами: Додавання, редагування, видалення товарів, налаштування варіацій, фотографій, цін.
- Управління замовленнями: Обробка замовлень, відстеження стану доставки, керування поверненнями.
- Аналітика та звіти: Отримання даних про продажі, відвідуваність, конверсію, популярність товарів.
- Керування маркетингом: Налаштування кампаній, промокодів, знижок, програм лояльності.

1.3. Мета дослідження

Визначивши актуальність та розглянувши аналоги інформаційних систем, було сформульовано мету проекту, яка полягає у створенні інформаційної системи. Основна мета даної системи є легке керування веб-додатком, з яким впорається навіть одна людина.

Ця інфромаційна система матиме такі переваги як:

- Просте керування товаром в магазині тобто, додання та видалення товару з магазину, створення категорій.
- Зручний моніторинг замовлень, перегляд стану та самого товару в замовлені, адресу та персональні дані покупця.
- Зрозуміла аналітика продаж, статистика продаж по місяцям, загальна сумма продаж за весь час, кількість здійснених замовлень.

РОЗДІЛ 2. ВИЗНАЧЕННЯ ЗАДАЧІ І МЕТОДИ ЇХ РЕАЛІЗАЦІЇ

2.1 Визначення задачі

Основна мета цієї кваліфікаційної роботи – це створення простої в керуванні та інтерфейсі інформаційної системи для адміністрування веб-додатком, з самими основними функціями: керування товаром(додання, видалення, редагування), керування самим магазином та візуалізацією даних.

До створеної інформаційної системи буде створено невеличкий веб-магазин, на якому і будуть працювати всі функції, та з якого і буде йти статистика в нашу інформаційну систему, з робочою функцією корзини та оплатою товару.

Перелік задач для досягнення поставленної мети роботи:

- Аналіз вже доступних адмін-панелей;
- Обрати засоби реалізації;
- Вибір бази даних;
- Створення веб-магазину

2.2 Методи реалізації

Мова програмування:

Для написання цієї системи було використано мову Java Script, а саме відкриту платформу на її остнові: Node.js.

Node.js - це вільна, відкрита платформа на основі JavaScript, побудована на движку Chrome V8 JavaScript від Google. Node.js дозволяє виконувати JavaScript на серверній стороні, що робить його ідеальним для розробки сучасних веб-додатків.

Основні особливості Node.js:

Non-blocking I/O: Node.js використовує асинхронні функції та події для обробки багатьох запитів без блокування інших операцій, що дозволяє досягти високої ефективності в опрацюванні багатоопераційних додатків.

- Скальованість: Його архітектура дозволяє легко масштабувати додатки горизонтально, тобто додавати нові сервери для обробки більшої кількості запитів.
- Платформонезалежність: Node.js працює на різних операційних системах, таких як Windows, macOS, Linux тощо.
- Пакетний менеджер npm: Node.js поставляється з пакетним менеджером npm (Node Package Manager), який дозволяє легко встановлювати, оновлювати та керувати залежностями в проектах.
- Використання на сервері: Node.js використовується для реалізації серверної частини веб-додатків, API, мережевих додатків, чат-ботів, інструментів командного рядка та інших типів програм.

Node.js є потужним інструментом для розробників, який дозволяє створювати швидкі та масштабовані додатки за допомогою JavaScript як на клієнтській, так і на серверній стороні.

Фреймворки:

В якості основного фреймоврка було обрано: Next.js

Next.js - це популярний фреймворк для розробки веб-додатків на основі React. Він надає набір інструментів для побудови сучасних, швидких і масштабованих веб-додатків. Основні особливості Next.js включають:

- Рендеринг на стороні сервера та на стороні клієнта : Next.js підтримує як статичне, так і серверне рендеринг, що дозволяє підбирати оптимальний метод відображення сторінок для покращення продуктивності та SEO.
- Статичне згортання (SSG): Здатність генерувати статичні файли під час збірки проекту, що покращує швидкість завантаження сторінок.

- Роутинг на основі файлів: Next.js використовує конвенцію іменування файлів для роутингу, що спрощує організацію структури додатку.
- API маршрутизація: Легко створювати API-маршрути для виконання серверних функцій.
- Завантаження на мінімальний час: Забезпечує можливість передзагрузки сторінок, що покращує швидкість переходу між сторінками додатку.

Next.js дозволяє розробникам створювати швидкі та потужні веб-додатки, використовуючи зручні інструменти React разом з низкою оптимізацій, які полегшують процес створення додатків.

Також в цьому проекті було використано фрейморк для CSS – Tailwind, який надає змогу швидко створювати і налаштовувати вигляд компонентів без написання власного CSS.

2.3 База Даних

В якості бази даних було обрано MySQL, але реалізація її була через ORM (Object-Relational Mapping)

ORM (Object-Relational Mapping) - це програмний підхід, що дозволяє розробникам використовувати об'єктно-орієнтований підхід до роботи з базами даних, які зазвичай базуються на реляційній моделі даних. Основна ідея ORM полягає в тому, що вона спрощує роботу з базою даних, приховуючи деталі взаємодії з нею за допомогою об'єктно-орієнтованого підходу. Замість того, щоб працювати з запитами SQL напряму, розробник використовує об'єкти та методи для взаємодії з даними в базі даних.

Для цього було обрано одну із відомих ORM систем – Prisma, який в свою чергу був під’єднаний до PlanetScale.

Prisma - це сучасний ORM (Object-Relational Mapping) та інструмент для керування базами даних, який надає зручний інтерфейс для взаємодії з реляційними базами даних у додатках. Основна мета Prisma - спростити роботу з базами даних і зробити їхнє управління більш ефективним.

Основні компоненти Prisma:

- Prisma Client: Це генерований автоматично клієнт, який надає зручний API для взаємодії з базою даних у вашому коді. Prisma Client автоматично генерує функції для створення, читання, оновлення та видалення даних на основі моделей, описаних у схемі.
- Prisma Schema: Розробники визначають структуру бази даних та моделі даних у файлі schema.prisma, використовуючи спеціальну мову. Тут описуються сутності, взаємозв'язки, типи даних та інші характеристики.
- Prisma Migrate: Цей інструмент дозволяє створювати та керувати міграціями бази даних, тобто змінами в схемі. Prisma Migrate використовується для зміни структури бази даних без втрати даних.

Prisma може працювати з різними базами даних, такими як PostgreSQL, MySQL та SQLite. Він дозволяє розробникам зосередитися на розробці додатків, не вдаючись до рутинних завдань, пов'язаних з управлінням базою даних, завдяки його зручному API та автоматизованим процесам міграцій та збереженню даних. Процес роботи бази даних Prisma включає наступні кроки:

- Моделювання даних: Користувач описує моделі даних за допомогою мови Prisma Schema. Це опис об'єктів бази даних, їх поля, типи даних, зв'язки між ними та інші параметри.
- Генерація коду: На основі описаних моделей Prisma генерує код для взаємодії з базою даних. Цей код може бути використаний для створення, читання, оновлення та видалення даних.
- Виконання операцій з базою даних: Після генерації коду програміст може взаємодіяти з базою даних через згенеровані методи. Prisma відстежує зміни в схемі бази даних та автоматично генерує необхідні міграції для збереження узгодженості моделі та реальної структури бази даних.

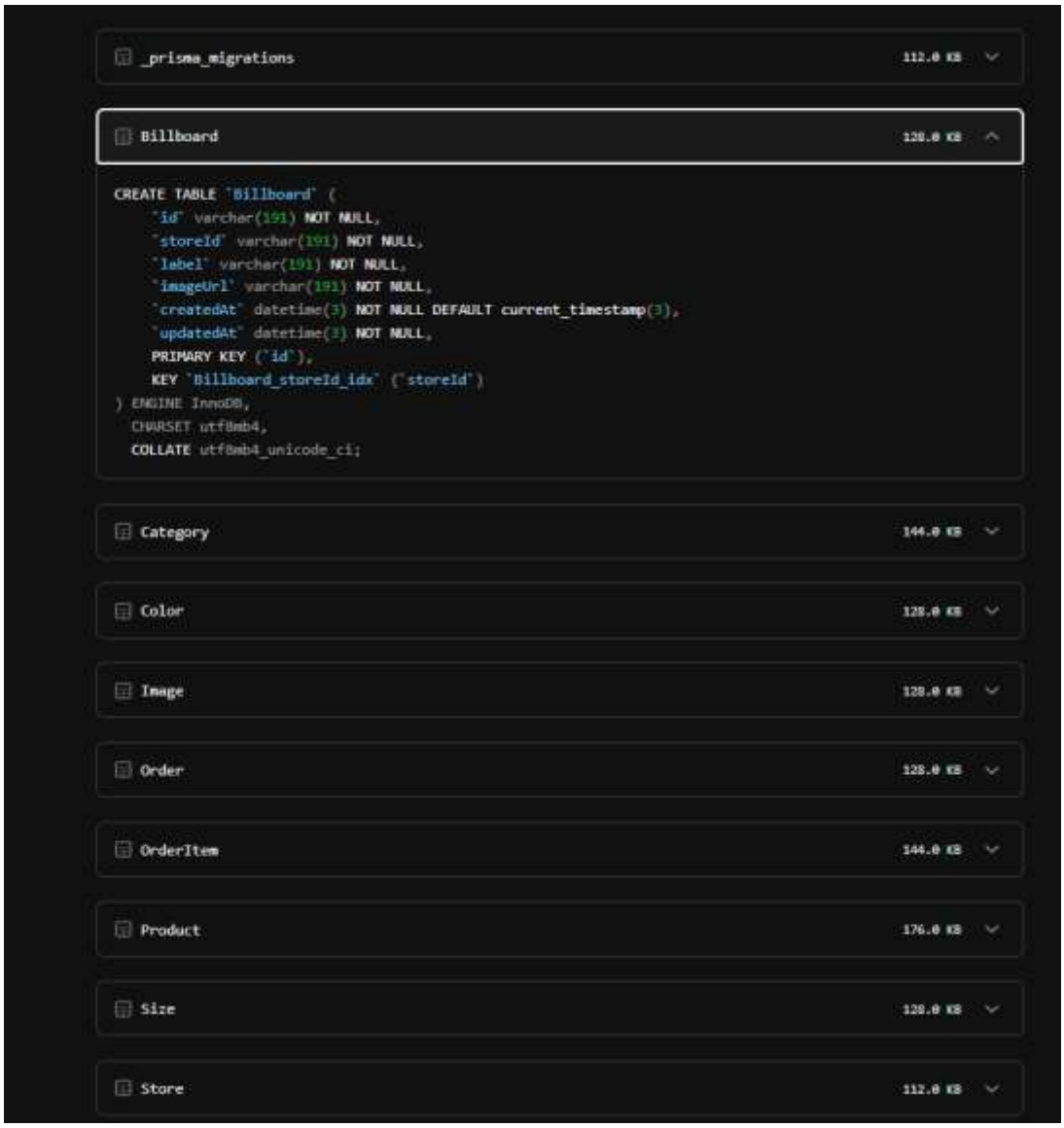


Рисунок 2.1 – Вигляд моделей функцій в базі даних “Prisma”

PlanetScale - це компанія, яка спеціалізується на управлінні базами даних, зокрема, реляційними базами даних. Вони надають облачний сервіс для роботи з базами даних, заснований на технології Vitess.

Основний продукт PlanetScale - це розподілена реляційна база даних, яка дозволяє легко масштабувати вашу базу даних, забезпечуючи високу доступність та надійність. Вони пропонують інструменти для керування схемою бази даних, резервним копіюванням, моніторингом та іншими аспектами управління даними.

PlanetScale використовує технологію Vitess, яка була розроблена у компанії YouTube для масштабування їхніх баз даних MySQL. Vitess дозволяє розподіляти

дані на кластері серверів та забезпечувати високу швидкодію при роботі з великими обсягами даних.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Програмна реалізація

Для реалізації нашої інформаційної системи було використано наступні технологічні системи:

- JavaScript – мова програмування;
- Next.js – фреймворк;
- Tailwind – CSS фреймворк;
- App Router – частина фреймворку Next.js, яка відповідає за маршрутизацію веб-додатку;

Для початку потрібно створити сам проект, створення проекту виконується командою:

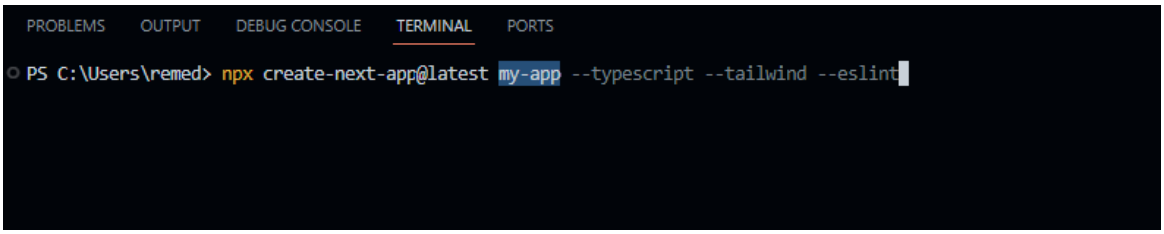


Рисунок 3.1 – Команда для створення проекту

Де замість “my-app пишеться назва додатка, в нашому випадку це – diplom. Після цього в нас з'являються основні папки та файли нашого проекту:

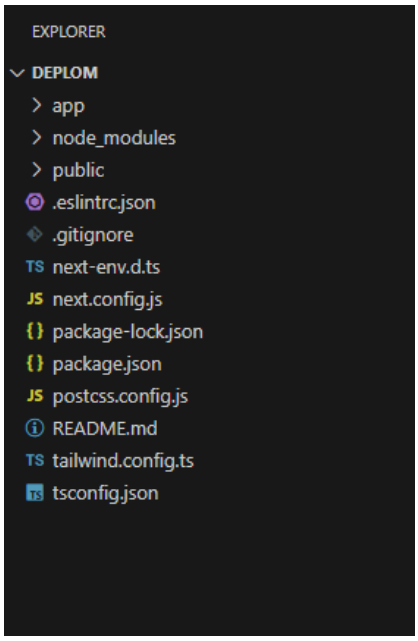


Рисунок 3.2 – Основні папки та файли проекту

В написані компоненті по типу: кнопок, таблиць, контекстних меню і так далі… нам допоможе shadcn/ui**.**

shadcn/ui – це набір багаторазових компонентів, які можна копіювати та вставляти у свої програми, все що портрібно – це вибрати потрібний компонент з веб-сайт, скопіюйте та вставте код у свій проект і налаштуйте його відповідно до своїх потреб. Для встановлення його в проект, використовуємо команду та відповідаємо на деякі запитання:

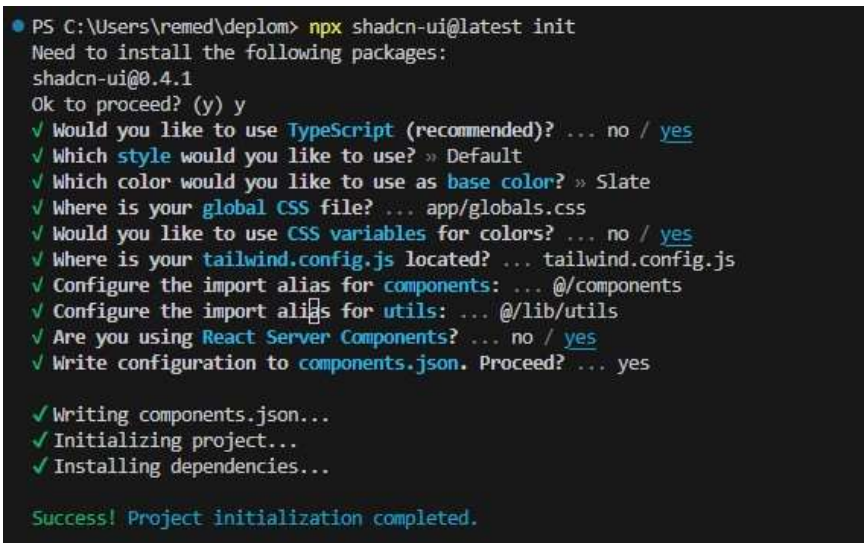


Рисунок 3.3 – Встановлення shadcn/ui

Робимо тестовий запуск, для перевірки, що все встановилось коректно і в нас немає помилок.

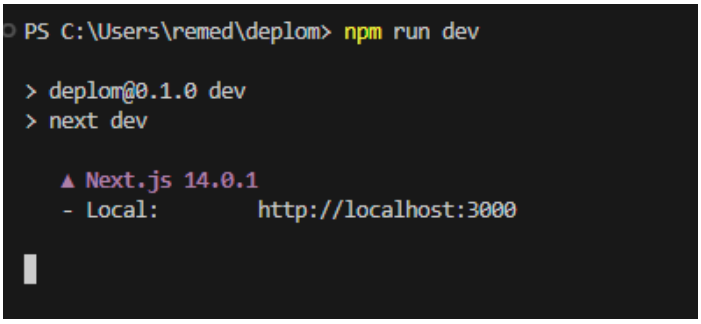


Рисунок 3.4 – Тестовий запуск проекту

І ми потрапляємо до стартової сторінки Next.js:

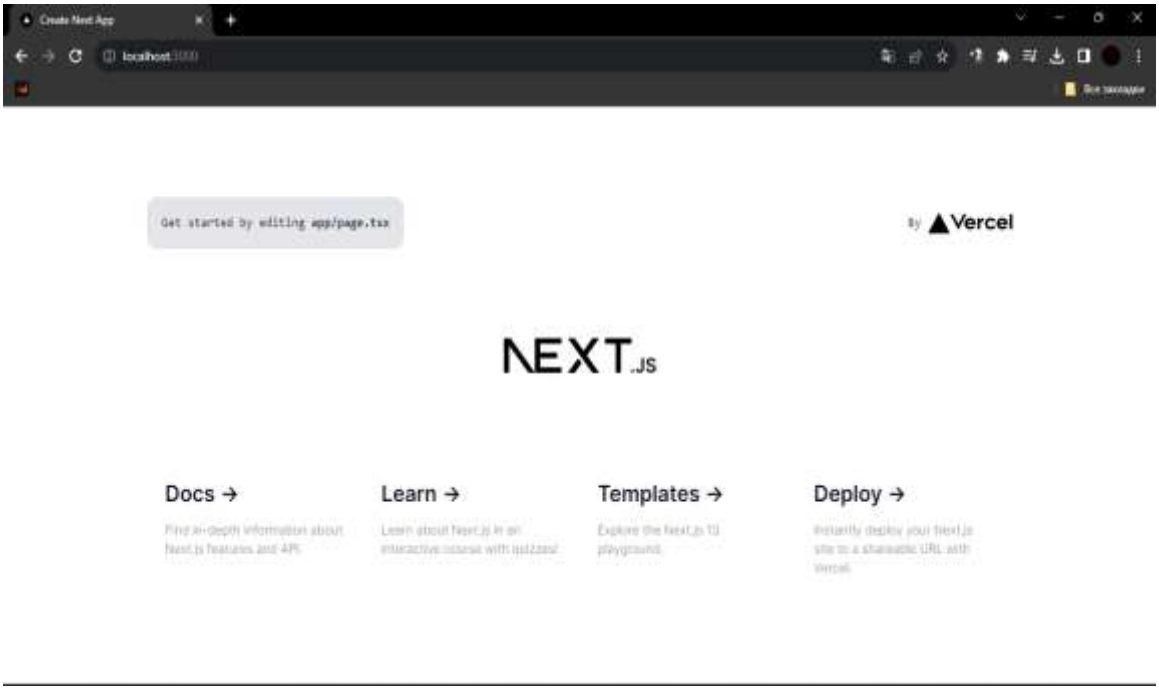


Рисунок 3.5 – Стартова сторінка Next.js

Наступний крок – створення головної сторінки, для цього в папці app створюється папка (root) в яку ми переміщаємо вже готову сторінку, і трішки змінюємо її:

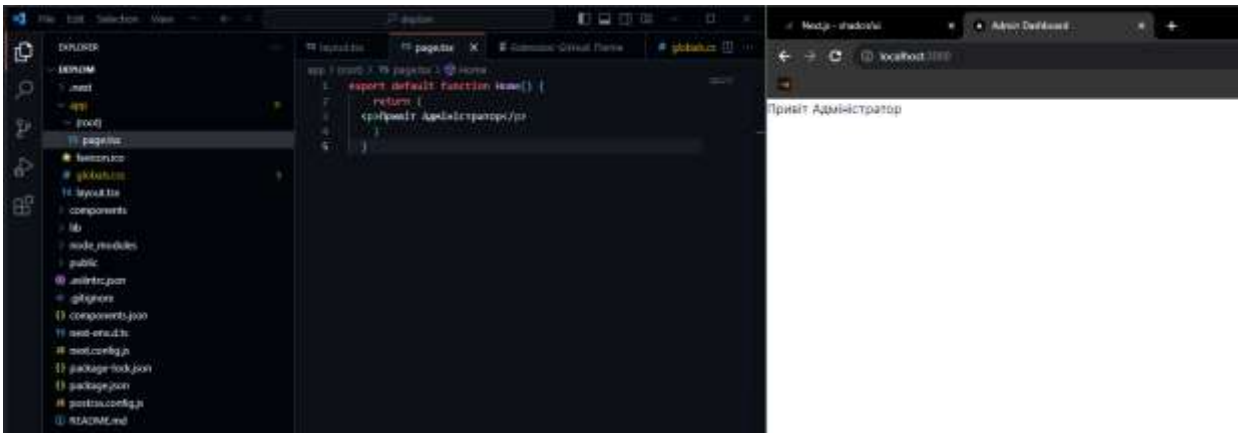


Рисунок 3.6 – Головна сторінка системи

Додамо тестову кнопку, для перевірки встановлених компонетів, для цього використовуємо команду:

```
npx shadcn-ui@latest add button
```

Рисуноу 3.7 – Команда для встановлення компоненту Button

Яка додає компонент “Button” в папку components/ui, яка і предзначена для цих компонентів.

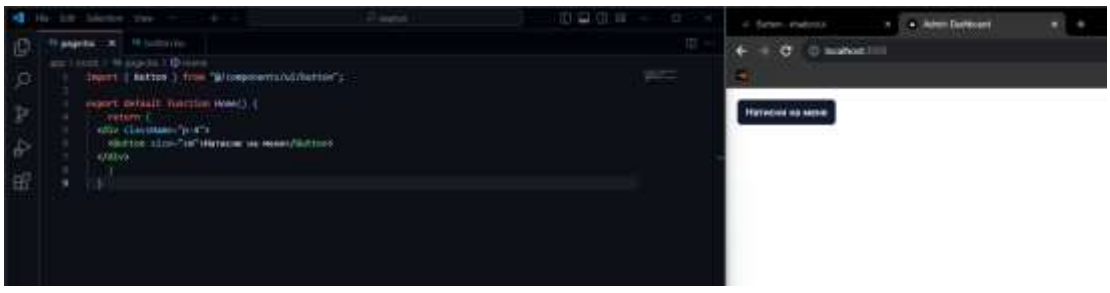


Рисунок 3.8 – Додання кнопки на головну сторінку

Як можна замітити по рисунку 3.8, для додання компоненту Button, використовується дія import цього компоненту з шляху @/components/ui/Button.

Також можливо замітити що для Button вже прописаний його розмір, в цьому нам допомагає Tailwind.

Ось ще декілька прикладів його використання:



Рисунок 3.9 – Змінна розміру кнопки на стандарний



Рисунок 3.10 – Змінна варіанта кнопки на “destructive”



Рисунок 3.11 – Змінна варіанта кнопки на “outline”

Одним із головних компонентів інформаційної системи – це авторизація. Для її створення було використано сервіс “Clerk”, який надає змогу додати систему авторизацію і налаштувати її під свій додаток. Все що потрібно – це зайти на сам сервіс, зареєструватись та створити проект, в нашому випадку він називається “Diplom”.

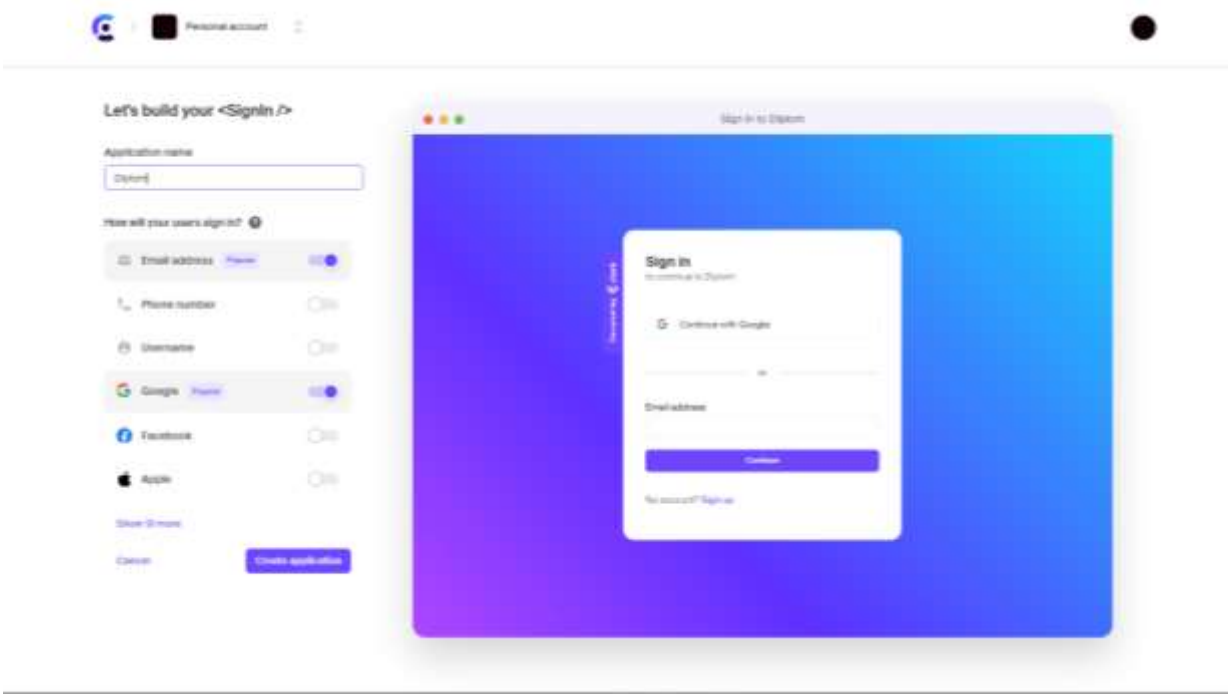


Рисунок 3.12 – Створення системи авторизації в сервісі “Clerk”

Як видно на Рисунку 3.12, окрім методу реєстрації через пошту та Google аккаунту, ми маємо ще купу додаткових методів авторизації, які додаються простим натисканям на кнопку.

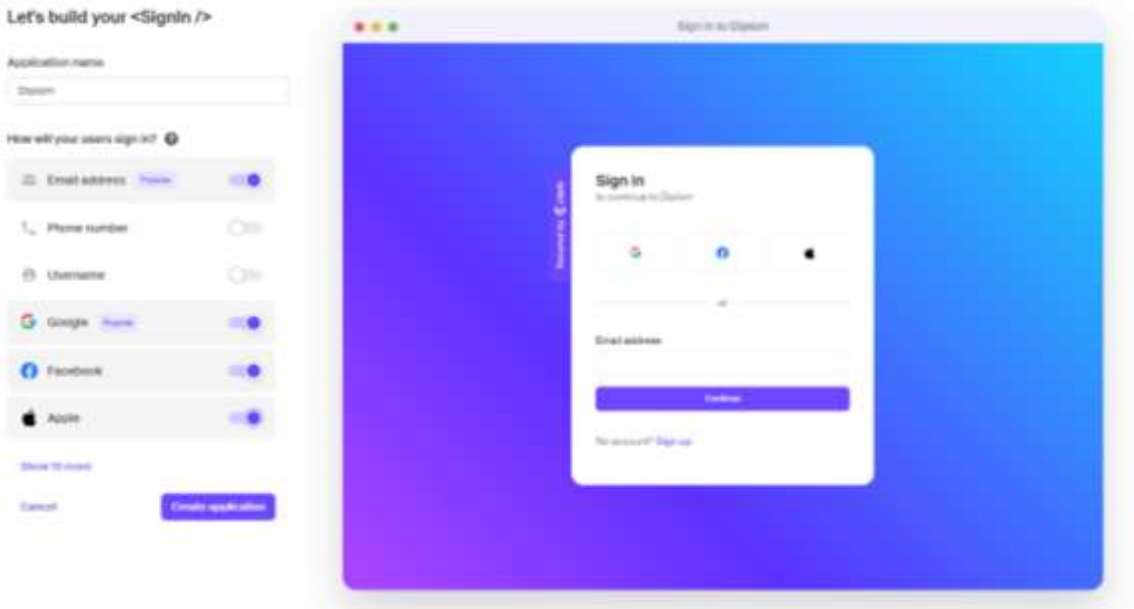


Рисунок 3.13 – Додання додаткових методів авторизації

Після створення проекту ми отримуємо два API ключа які потрібно вписати в наш .env файл. Файл з розширенням .env (Environment) - це текстовий файл, який часто використовується в розробці програмного забезпечення для зберігання конфігураційних змінних середовища. Його основна мета - зберігати конфіденційні або змінні дані, такі як паролі, ключі API, налаштування баз даних тощо, у вигляді пар ключ-значення.

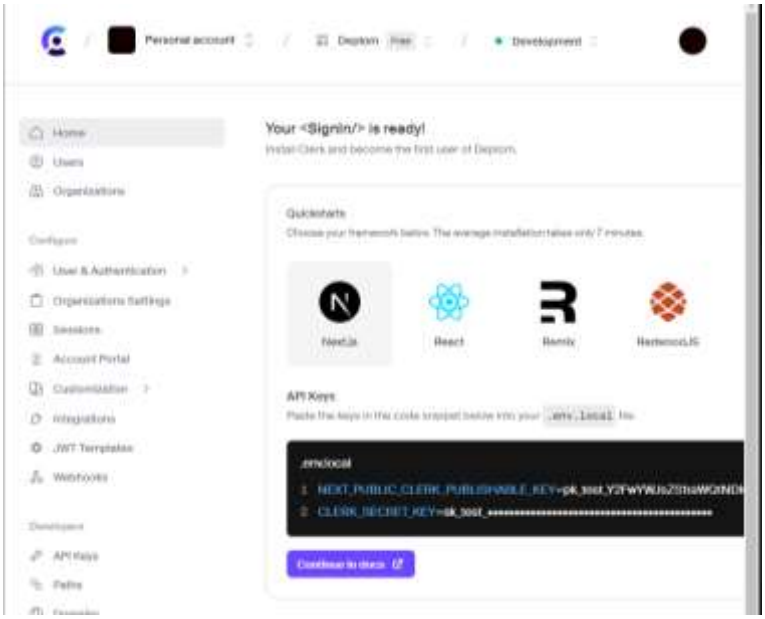


Рисунок 3.14 – Отримання API ключів

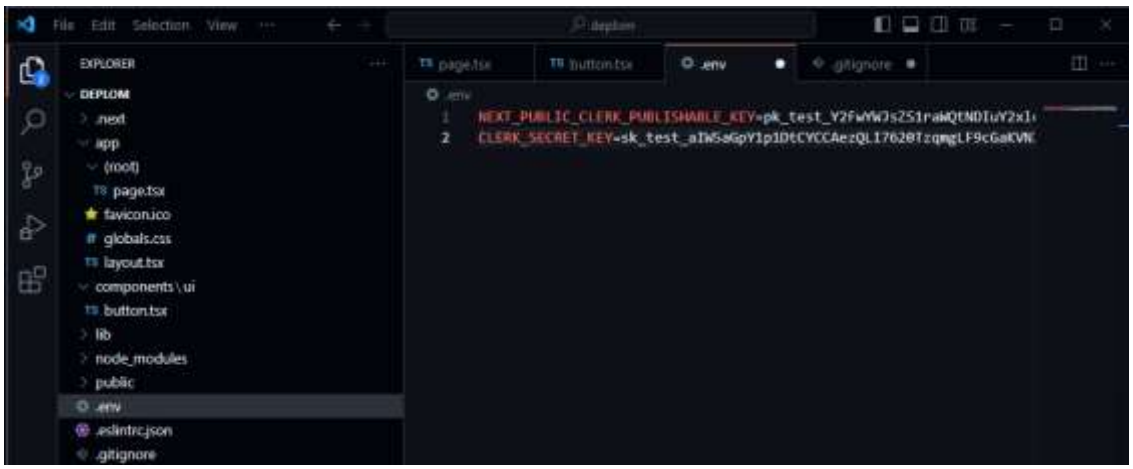


Рисунок 3.15 – Вставлення API ключів в .env файл

Далі потрібно встановити пакет Clerk в інформаційну систему командою:

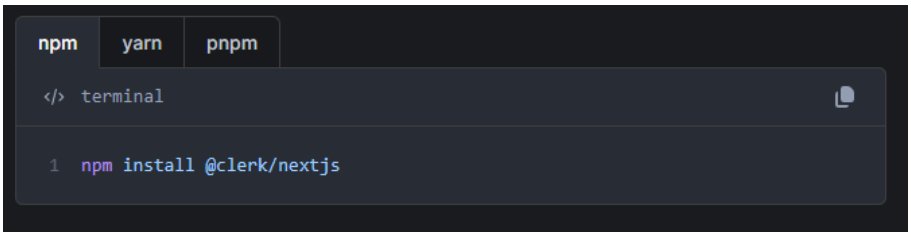


Рисунок 3.16 – Команда для встановлення пакету Clerk

Наступним кроком потрібно створити файл під назвою: middleware.ts, який являється модулем, який містить функції-посередники (middleware functions), які використовуються в розробці веб-додатків для обробки запитів до сервера перед їхнім фактичним обробленням. Ці функції можуть виконувати певні операції, такі як аутентифікація користувачів, перевірка дозволів, логування запитів, обробка помилок тощо. Та вписуємо туди код:

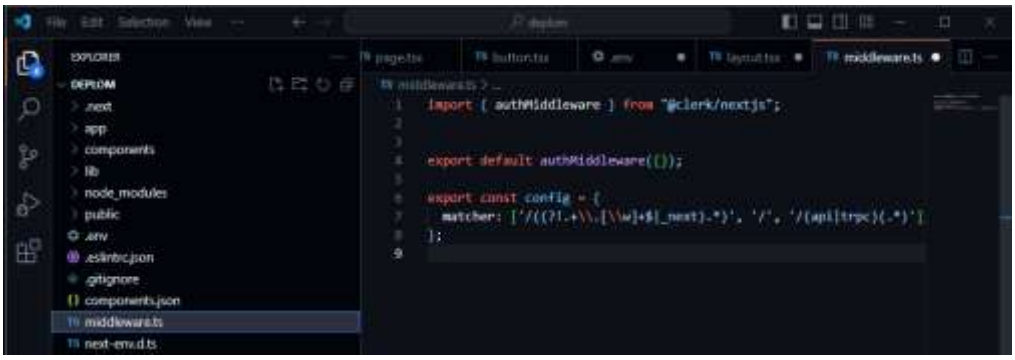


Рисунок 3.17 – Прописання файлу middleware.ts

Після перезавантаження проекту, ми потрапляємо на вікно авторизації від Clerk:

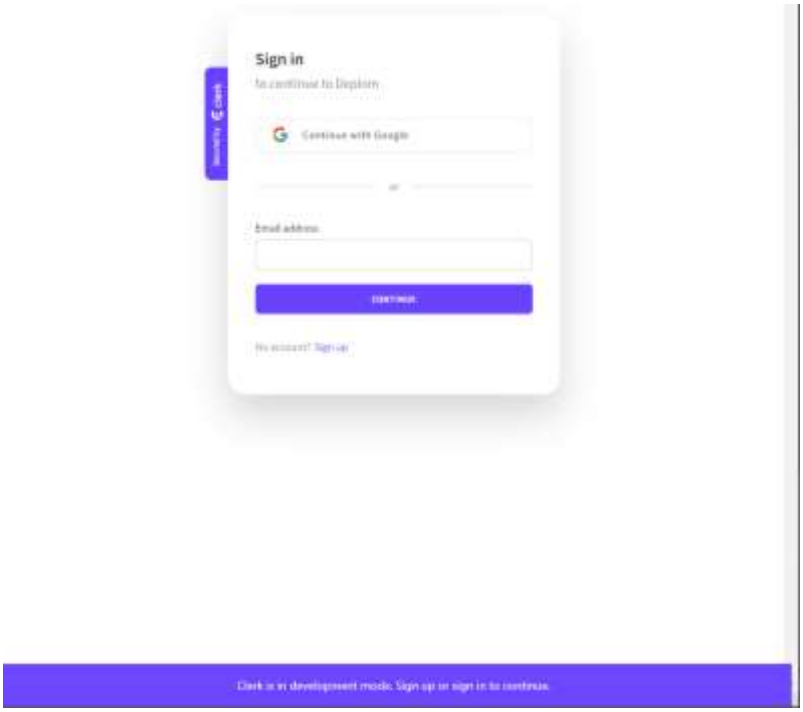


Рисунок 3.18 – Готове вікно авторизації “Clerk”

Після цього ми можемо зайти на сайт і побачати повідомлення про успішне підключення та деяку інформацію, таку як: Кількість користувачів, кількість активних користувачів, кількість реєстрацій та авторизацій за місяць:

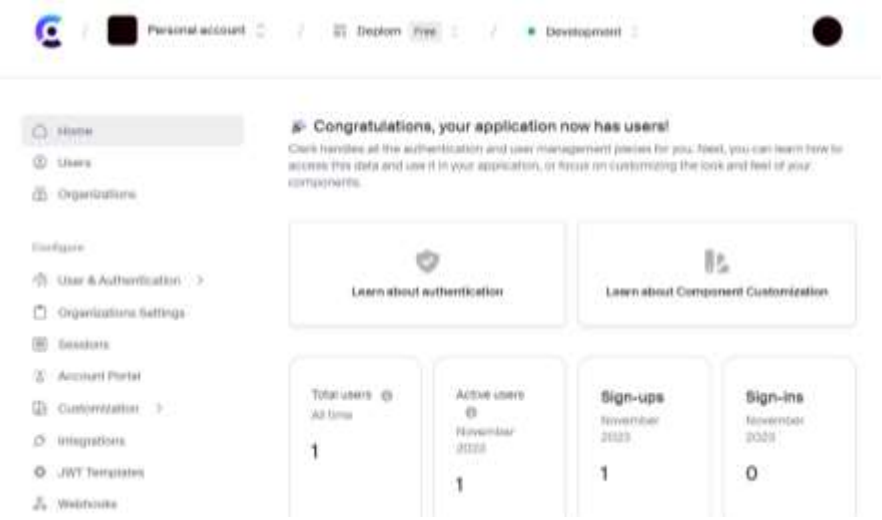


Рисунок 3.19 – Інформаційна панель в сервісі “Clerk”

Після авторизації відкривався пустий, білий екран, який було вирішено замінити на модель створення магазину в інформаційній системі;

Для цього завдання було створенно таку модель:

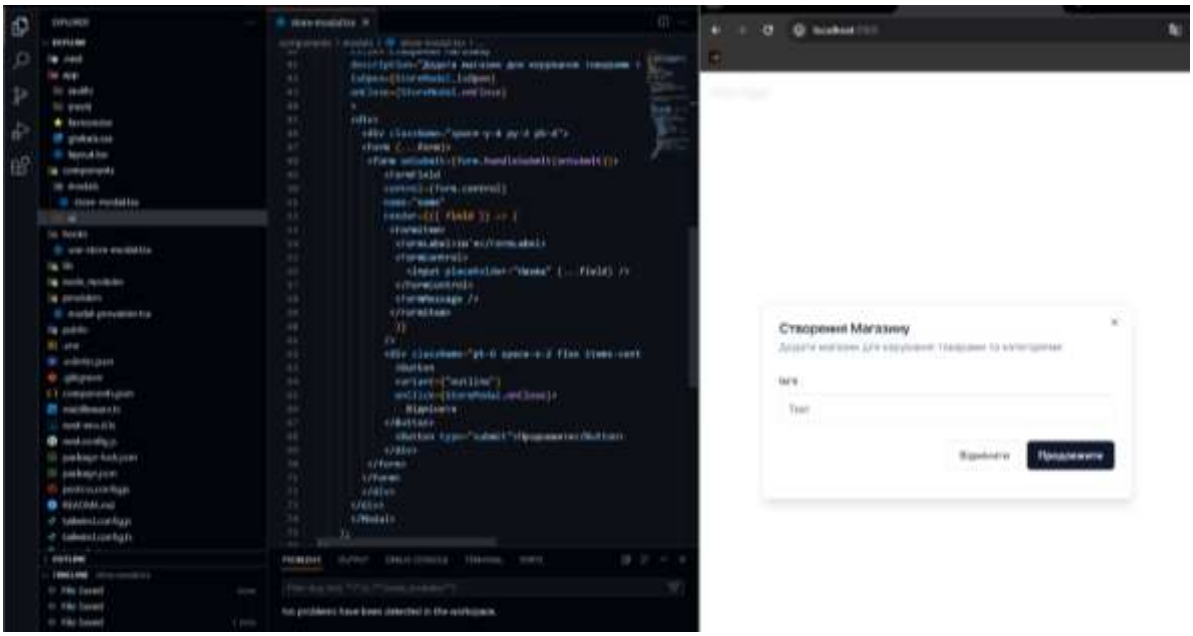


Рисунок 3.20 – Готова модель створення магазину

Та під’єднання її до головного екрану нашої інформаційної системи:

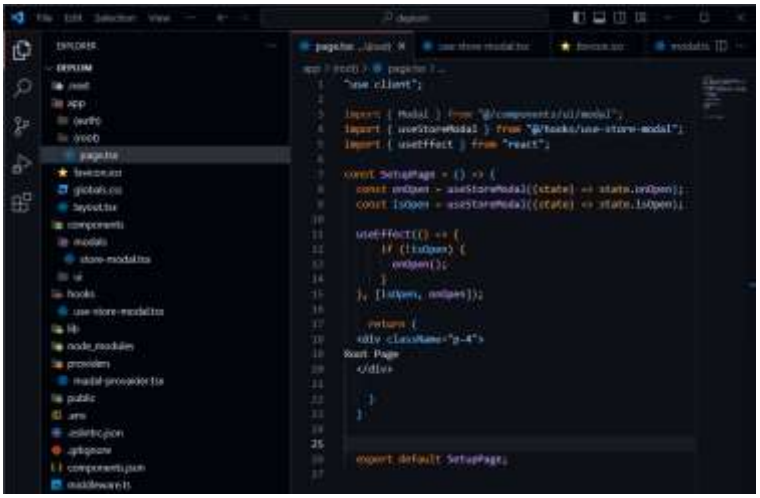


Рисунок 3.21 – Додання моделі на головний екран

3.2 З’єднання з базой даних

Далі нам потрібно з’єднати нашу систему та базу даних Prisma, для цього спочатку потрібно встановити Prisma та Prisma Client пакети в наш проект.

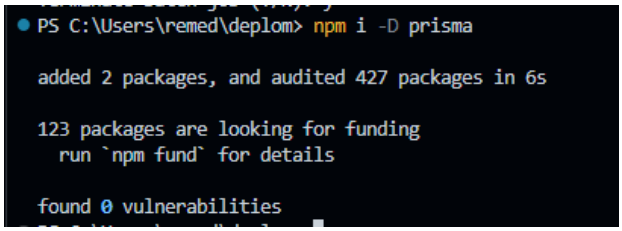


Рисунок 3.22 – Встановлення Prisma пакету

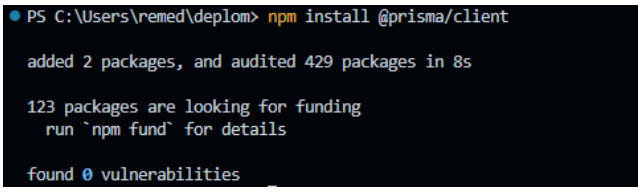


Рисунок 3.22 – Встановлення Prisma Client пакету

Після встановлення потрібно створити папку “Prisma” в якій буде створенно файл “schema.prisma”.

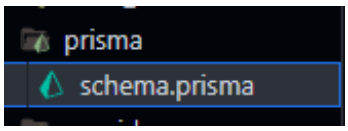


Рисунок 3.23 – Створення папки та файлу для Prisma

Запускаємо базу даних командою: npx prisma init, переходимо до сервісу “PlanetScale” та створюємо базу даних, після чого в нас з’являється вікно для з’єднання цієї бази даних з проектом:



Рисунок 3.24 – Вікно підключення бази даних “PlanetScale”

Для підключення нам потрібно скопіювати стірчку: DATABASE_URL = та вставити її в файл .env.



Рисунок 3.25 – Файл .env з силкою на базу даних

Наша база даних майже готова, залишилось тільки створити файл який буде допомагати в налаштувані та з’єднані з базою даних, під назвою “prismadb.ts”



Рисунок 3.26 – Створення файлу “prismadb.ts”

Надалі в ньому будуть прописуватись всі моделі, конфігурації та параметри підключення бази даних. Після підключення ми переходимо до PlanetScale, і

можемо побачити що в нас з’явилась 1 таблиця, яка означає що наша система під’єднана:



Рисунок 3.27 – Головна панель “PlanetScale”

API ключі — це унікальні ідентифікатори, які надають доступ до програмного інтерфейсу (API) певного сервісу чи додатку. Ці ключі використовуються для автентифікації користувачів, які намагаються отримати доступ до функцій чи даних через API.

В нашому випадку API ключі потрібні для доступу до функцій та підключення створеної моделі магазину в адміністративній панелі до створеного в подалі веб-додатку.

Створюємо папку “api” в якій будуть знаходитись api ключі всіх функцій та окремо api ключ магазину, для цього в папці “api” створюється папка “stores”, в якому створюється файл “route.ts”;

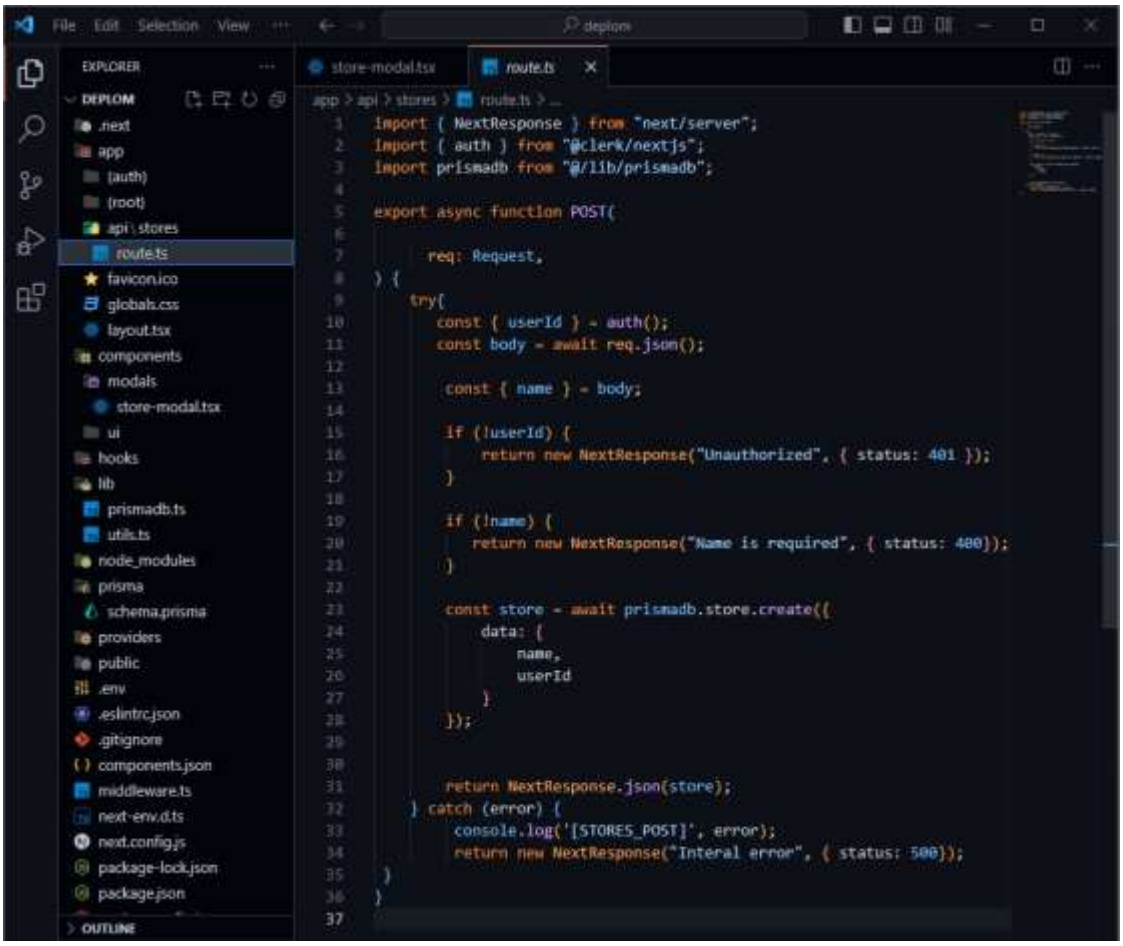


Рисунок 3.28 – Створення та налаштування API для магазину

Переходимо до проекту, і спробуємо створити магазин в системі (рис.3.29)

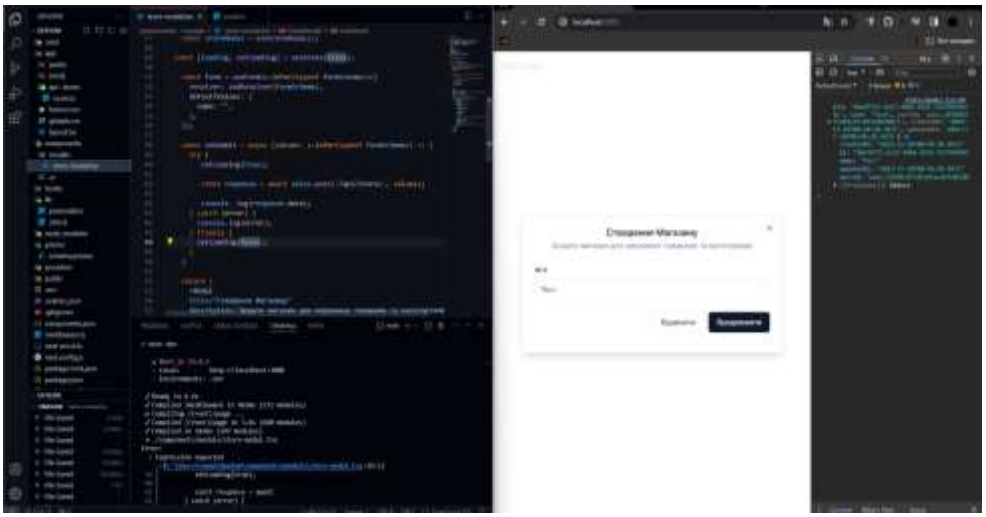


Рисунок 3.29 – Успішне створення магазину

Після введення назви, та натискання кнопки “Продовжити”, в консолі браузера ми можемо побачити, що наш магазин створився, і отримав свій api ключ. Перейдемо до стоврення навігаційної панелі.

В навігаційній панелі будуть розміщен всі функції цієї інформаційої системи, такі як:

- Перемикач магазинів
- Огляд статистики
- Банера
- Категорії
- Розміри
- Кольори
- Товар
- Замовлення
- Налаштування

Почнемо з самого першого але спочатку потрібно створити саму навігаційну панель, для цього створюється і прописується файл navbar.tsx, в цьому файлі також прописаний наш перемикач магазинів (Рис. 3.28).

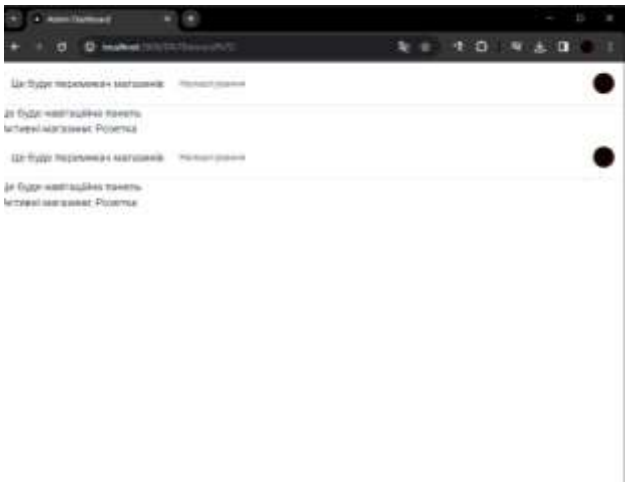


Рисунок 3.28 – Створення навігаційної панелі

Для подальшого створення функцій потрібно встановити необхідні компонети, для завантаження зображень та створення таблиць.

З першим нам допоможе сервіс “Cloudinary”.

Встановлюємо пакет Cloudinary в наш проект (рис. 3.29).



Рисунок 3.29 – Встановлення Cloudinary

Після авторизації в Cloudinary, та створення там проекту, ми отримуємо API ключі для підключення сервісу в систему (рис. 2.30).

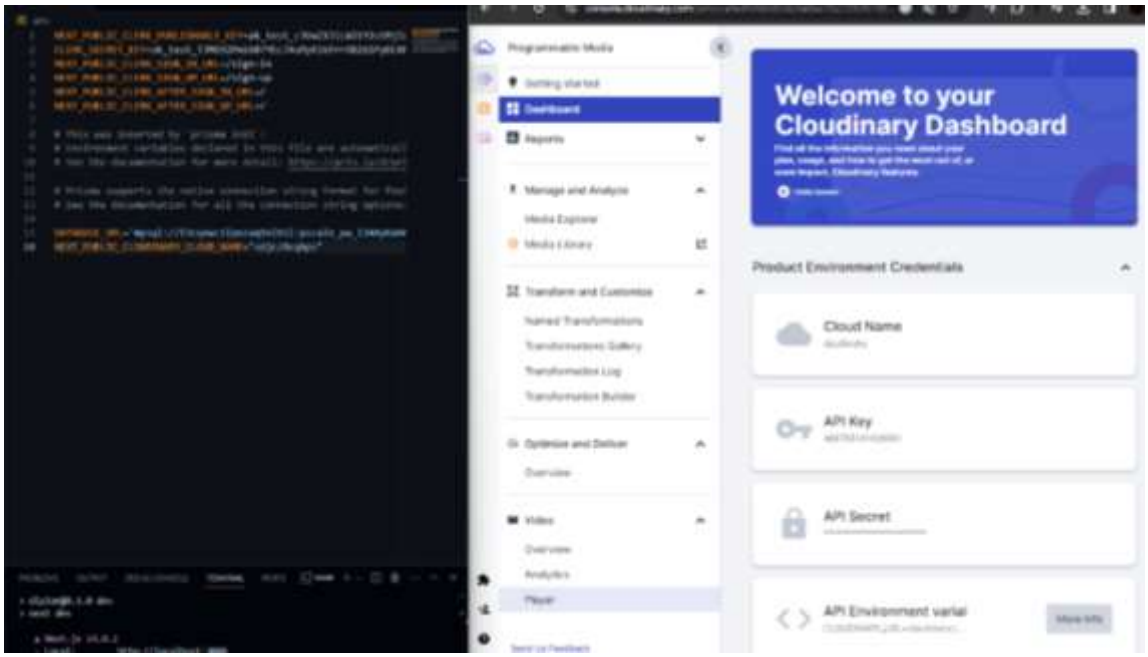


Рисунок 3.30 – Підключення Cloudinary

Для створення таблиць було встановленно пакет компонентів “Data Table” на бібліотеці “React” (рис 3.31).

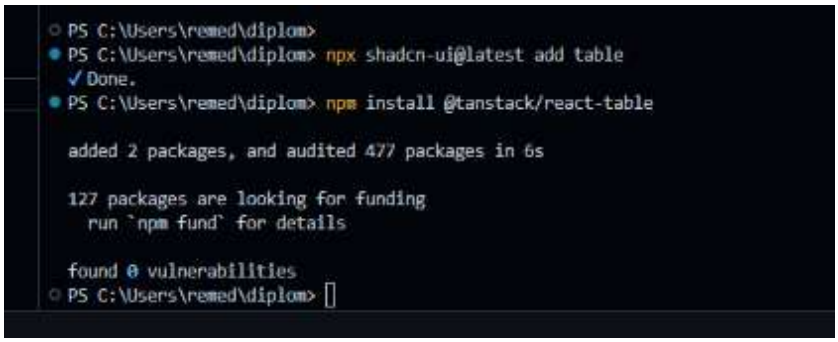


Рисунок 3.31 – Встановлення пакетів “Data Table”

Тепер створимо першу функцію в нашій система – це будуть, головні банера магазину, великі зображення які знаходяться зверху всіх сторінок магазину і показують інформацію про акції, знижки або на яких написано назву категорії, в якій перебуває покупець. Для цього в папці app створюється папка “dashboards” в якій створюється папка “storeId”, тут створюються папки самих функцій “billboards” “categories” і так далі.

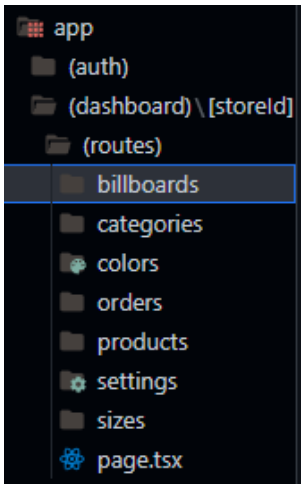


Рисунок 3.32 – Розташування функцій в проекті

Створення функцій схоже між собою, в кожній функції створюється папка з назвою функці та додатком “Id” для їх унікальної ідентифікації, тут створюється файл “page”– це є головна сторінка функції, та папка components, в якій будуть знаходитись набір окремих елементів інтерфейсу (таблиці, вікна, контекстні меню) (рис 3.33).

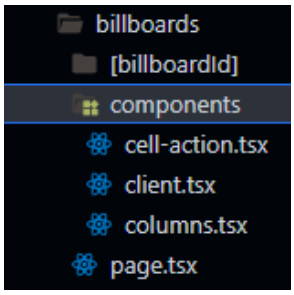


Рисунок 3.33 – Готовий вигляд функції в проекті

Розглянемо ці три файли, перший файл під назвою “cell-action ”, цей файл відповідає за контекстне меню “Дії” (рис 3.34) в якому можливо: копіювати id банера, змінювати його та видалити.

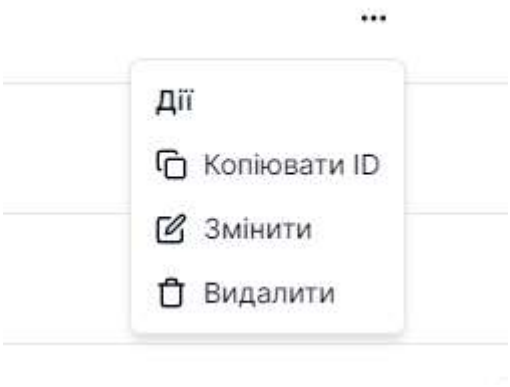


Рисунок 3.34 – Контекстне меню “Дії”

Наступний файл “client” в ньому прописується назва функції, його опис та кнопка, в нашому випадку – це функція “Банер” (рис 3.35-36).

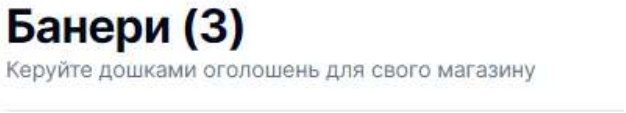


Рисунок 3.35 – Назва функції та її опис

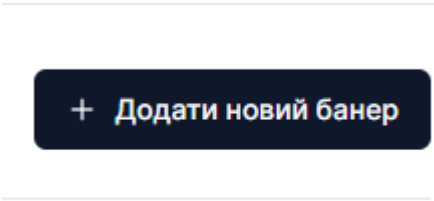


Рисунок 3.36 – Кнопка для функції “Банер”

Останій файл “columns” який відповідає за таблиці на сторінках функцій, в функції “Банер” ми маємо тільки два стовпчика в таблиці: Назва і Дата; (рис 3.37).



Рисунок 3.37 – Вигляд таблиць в функціях

Готовий вигляд Адміністративної панелі:

- Функція “Банер” (рис 3.38)

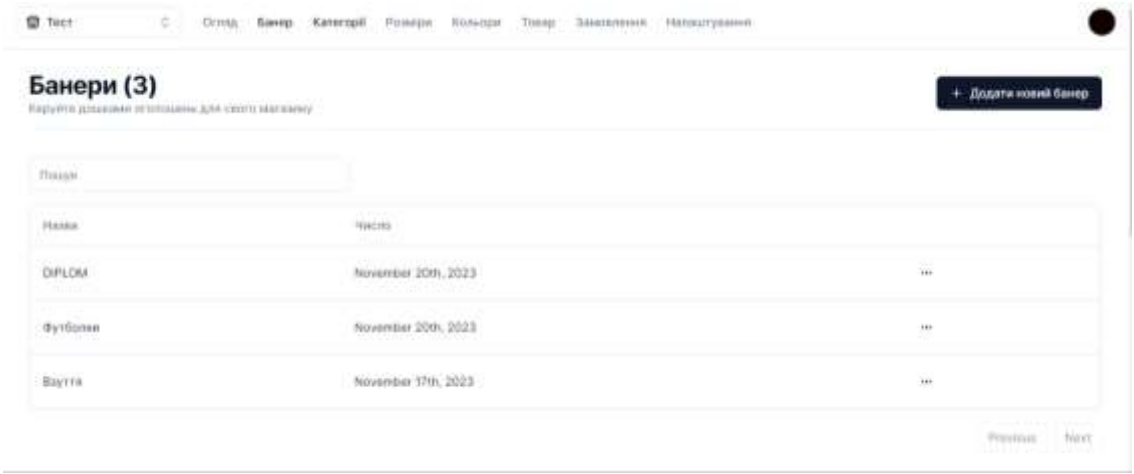
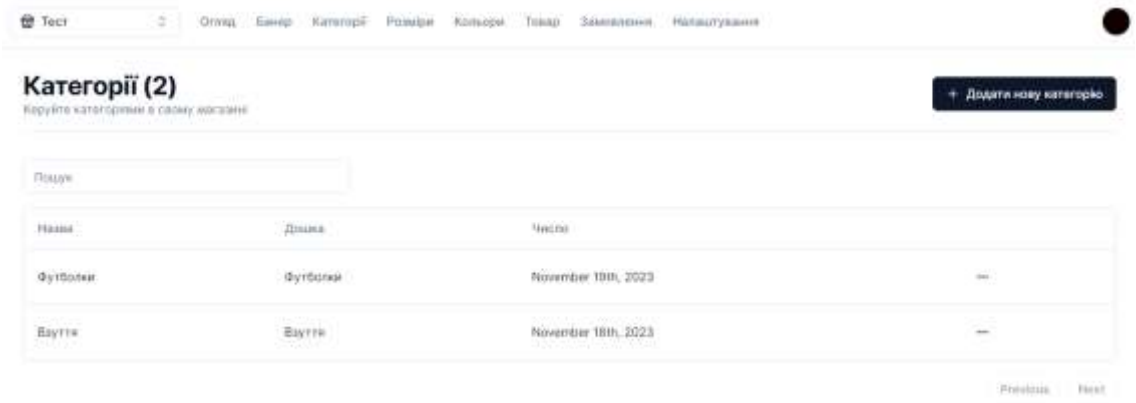


Рисунок 3.38

- Функція “Категоії” (рис 3.39)



3.39

- Функція “Розміри” (рис 3.40)

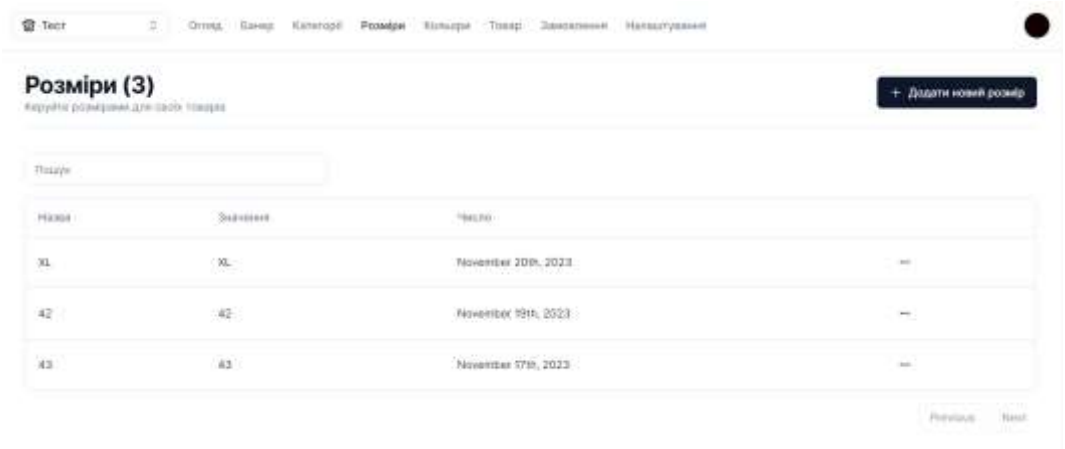


Рисунок 3.40

- Функція “Кольори” (рис 3.41)

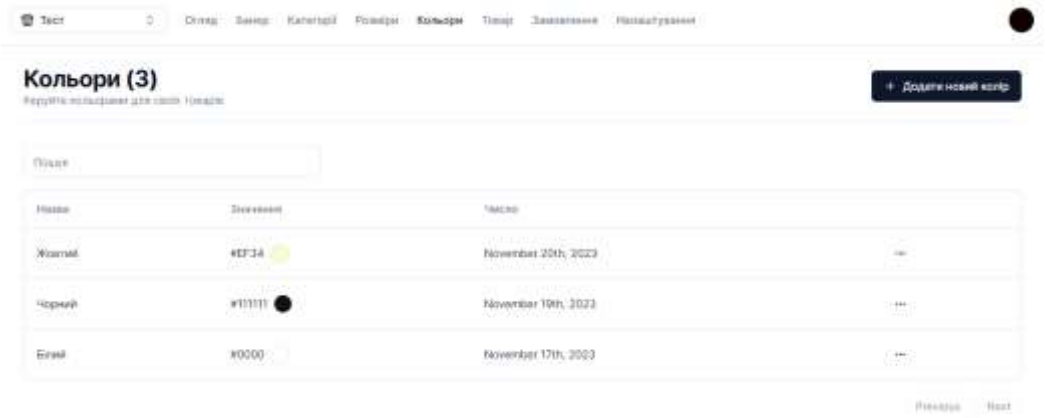


Рисунок 3.41

- Функція “Товар” (рис 3.42)

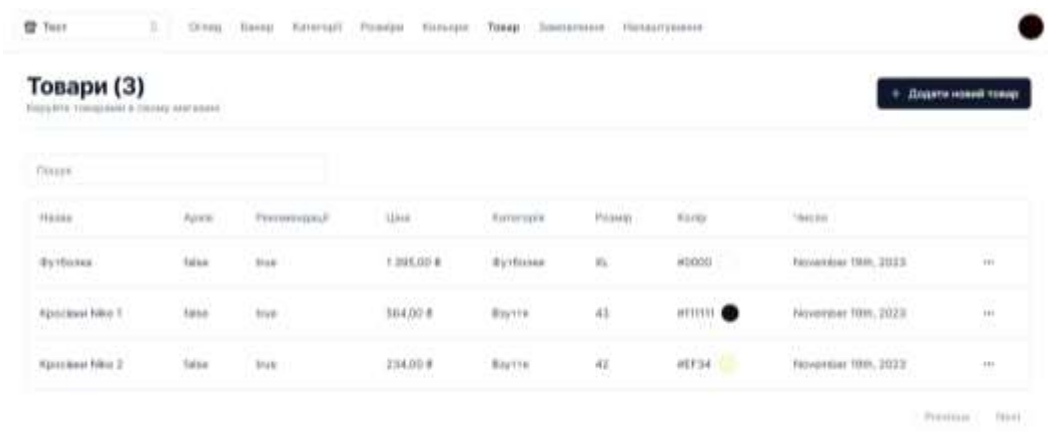


Рисунок 3.42

- Функція “Замовлення” (рис 3.44)

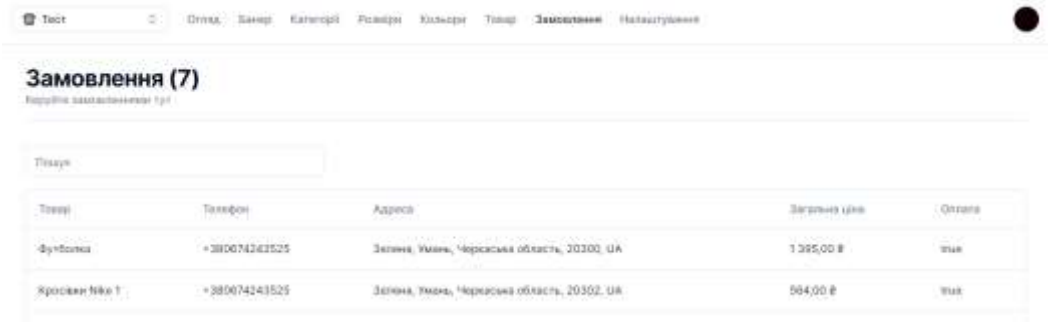


Рисунок 3.44

- Функція “Налаштування” (рис 3.45)

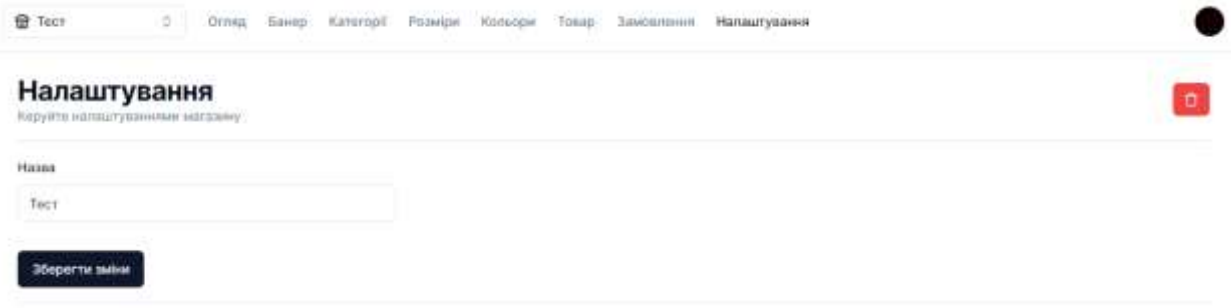


Рисунок 3.45

Всі ці функції вносяться в адміністративну панель вручну, окрім замовлень, про них буде потім, також ці функції поєднані між собою.

Наприклад якщо в функції “Банер” була створена функція “Категорія”, в якій будуть додані товари, то видалення банера та категорії буде закінчуватись помилкою. (рис 3.45 - 3.46).

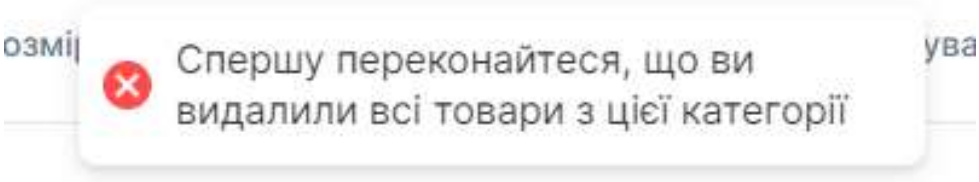


Рисунок 3.46 – Помилка при видалені категорії

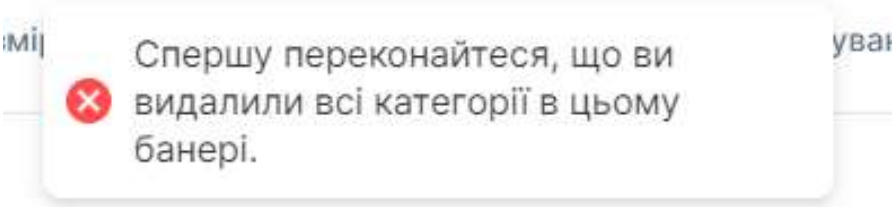


Рисунок 3.46 – Помилка при видалені банера

3.3 Створення веб-додатку

Для створення веб-додатку використовується та ж сама команда, як і для адміністративної панелі (рис 3.1) тільки міняється назва проекту на “diplom-store”. Після чого створюється головна сторінка магазину де будуть відображатись банер з назвою магазина та рекомендовані товари, але зараз вона має такий вигляд.

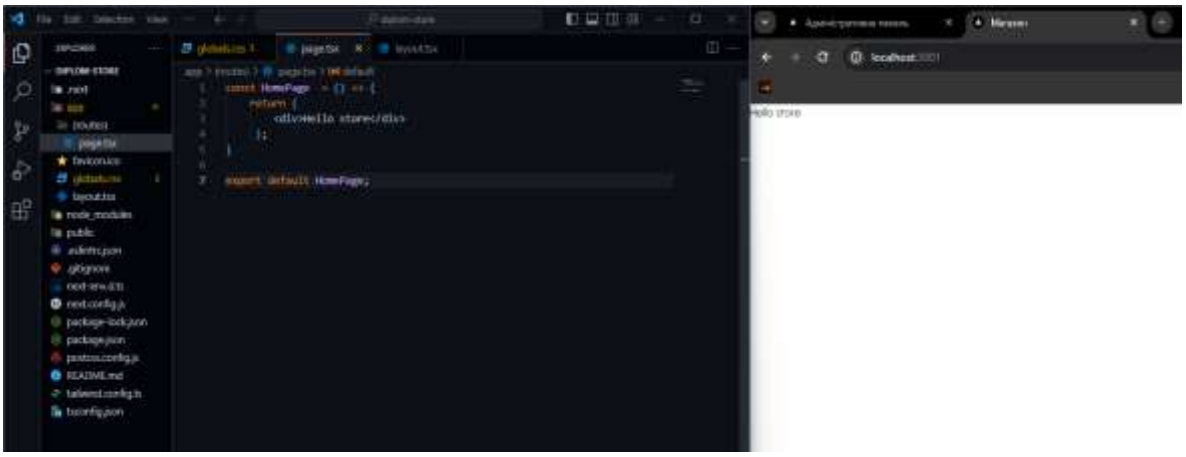


Рисунок 3.47 – Початковий вигляд головної сторінки магазину

Для взаємної роботи інформаційної системи і веб-додатку потрібно створити файл “.env” через який і буде виконано з’єднання, для цього в файлі .env прописується змінна середовища “NEXT_PUBLIK_API_URL”.
А в значенні цієї зміної вставляємо API силку, яку отримуємо в налаштуваннях адміністративної сторінки (рис 3.48).



Рисунок 3.49 – API силка в вкладкі налаштування



Рисунок – 3.50 Готовий вигляд “.env” файлу магазину

Перейдемо до створення панелі навігації, де будуть знаходитись назва магазину, категорії та кошик з товаром. Категорії, банера та товари відображаються за допомогою .env файлу. Для прикладу, відображення категорій з адміністративної панелі в магазині:

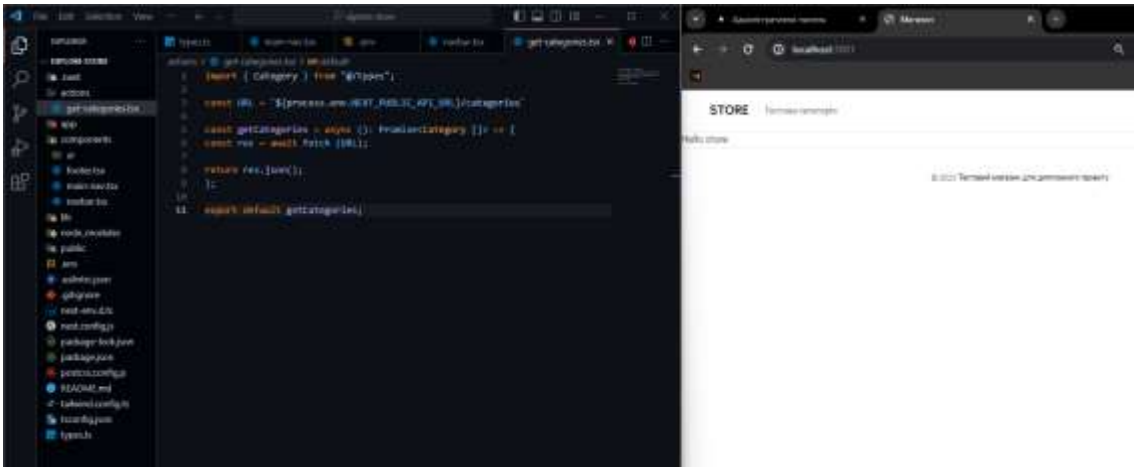
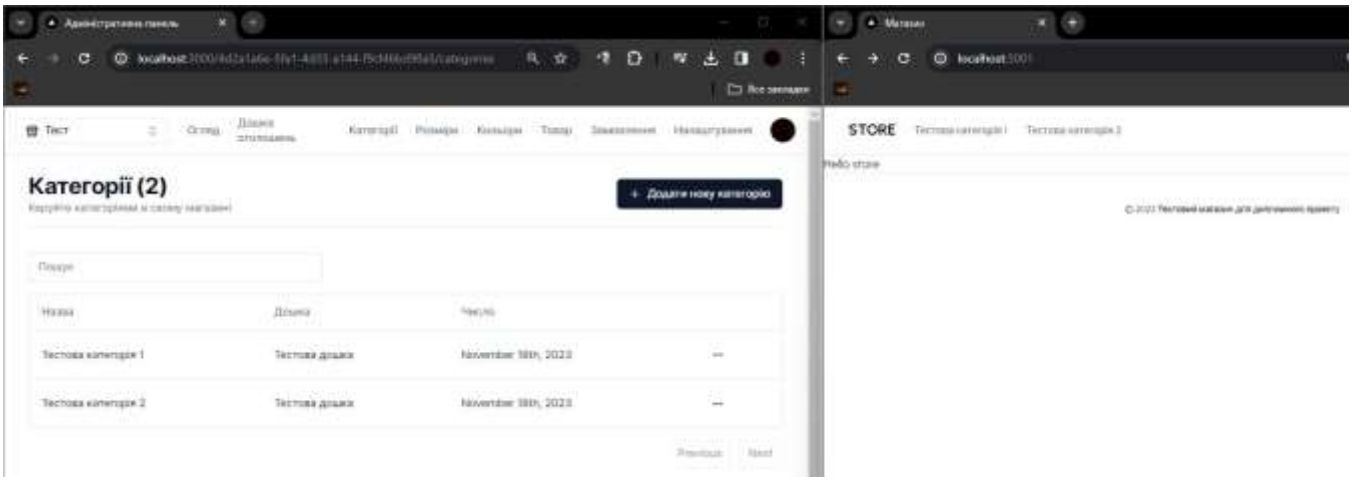


Рисунок 3.51 – Додання категорій в навігаційну панель магазину



Рискунок 3.52 – Приклад відображення категорій в адміністративній панелі та магазині

Перейдемо до додання товару в магазин, спочатку його потрібно створити в адміністративній панелі:



Рисунок 3.53 – Функція додання товарів в інформаційній системі

Після створення товару, перейшовши в його редагування через контекстне меню, ми маємо можливість додати певний товар, на головну сторінку в вигляді рекомендації, або ж перенести товар в архів, тобто повністю прибратий його з магазину (рис 3.54), но не з інформаційної системи.

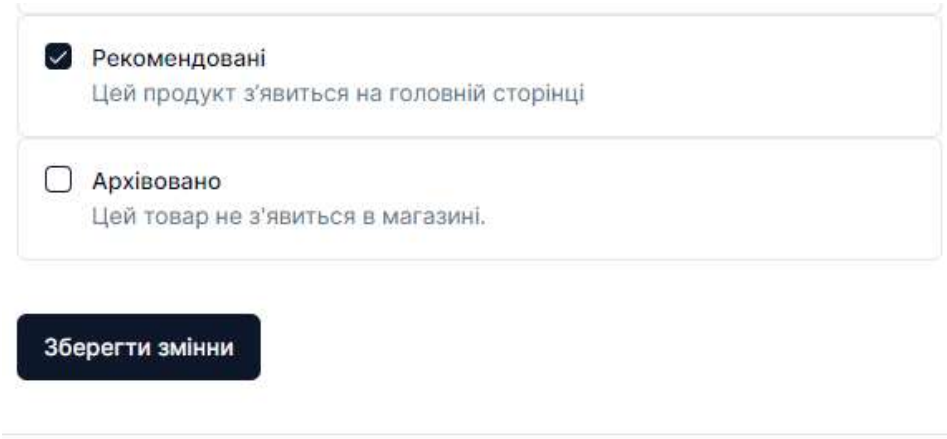


Рисунок 3.54

Після чого, товар позначений як рекомендований, з’являється на головній сторінці магазину (рис 3.55)

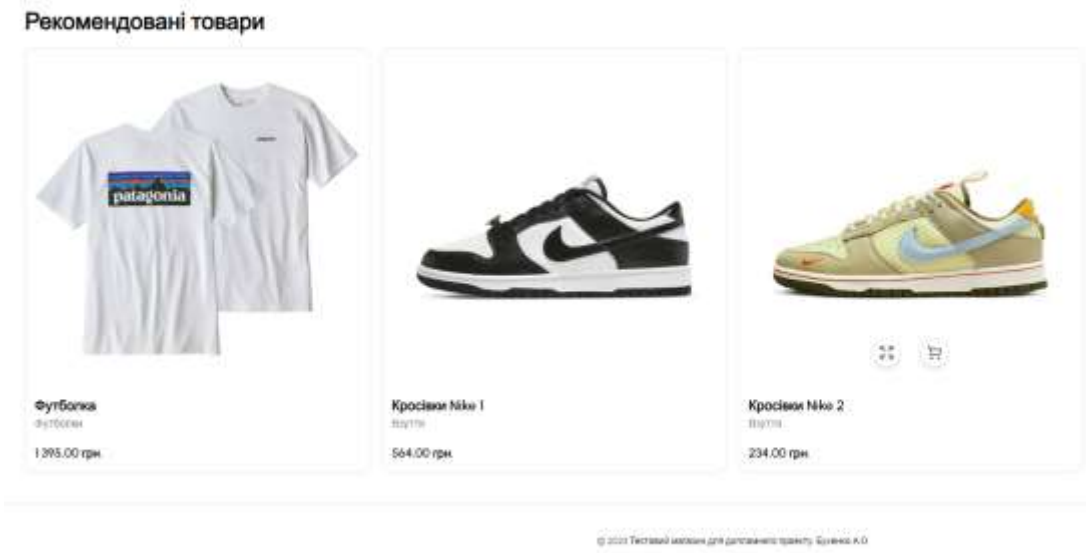


Рисунок 3.55 – Рекомендований товар на головній сторінці магазину

Також для кожного товару було створенно його персональна сторінка, з інформацією про товар (колір та розмір) та додатковим зображенням цього твору (рис 3.56)

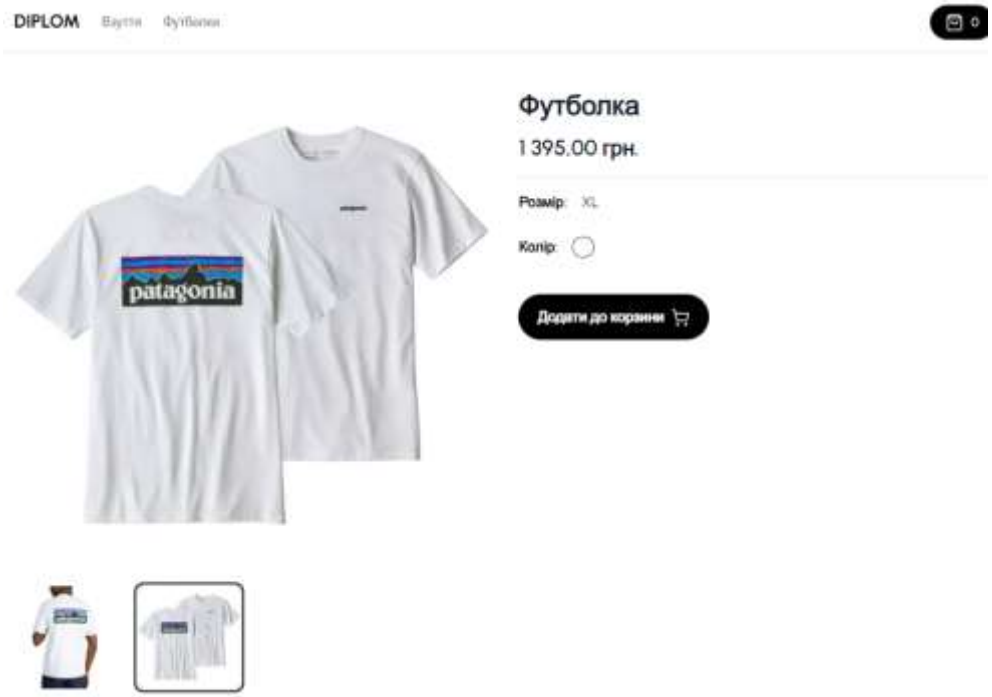


Рисунок 3.56 – Персональна сторінка товару

Наступним кроком було створення кошику для покупок в вигляді інтерактивної іконки яка показує кількість товру в кошику та персональною сторінкою, де відображається весь товар в кошику, загальною сумою всіх товарів та кнопкою оплати (рис 3.57, 3.58).

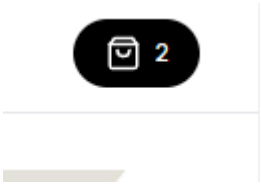


Рисунок 3.57 – Іконка кошика в навігаційній панелі

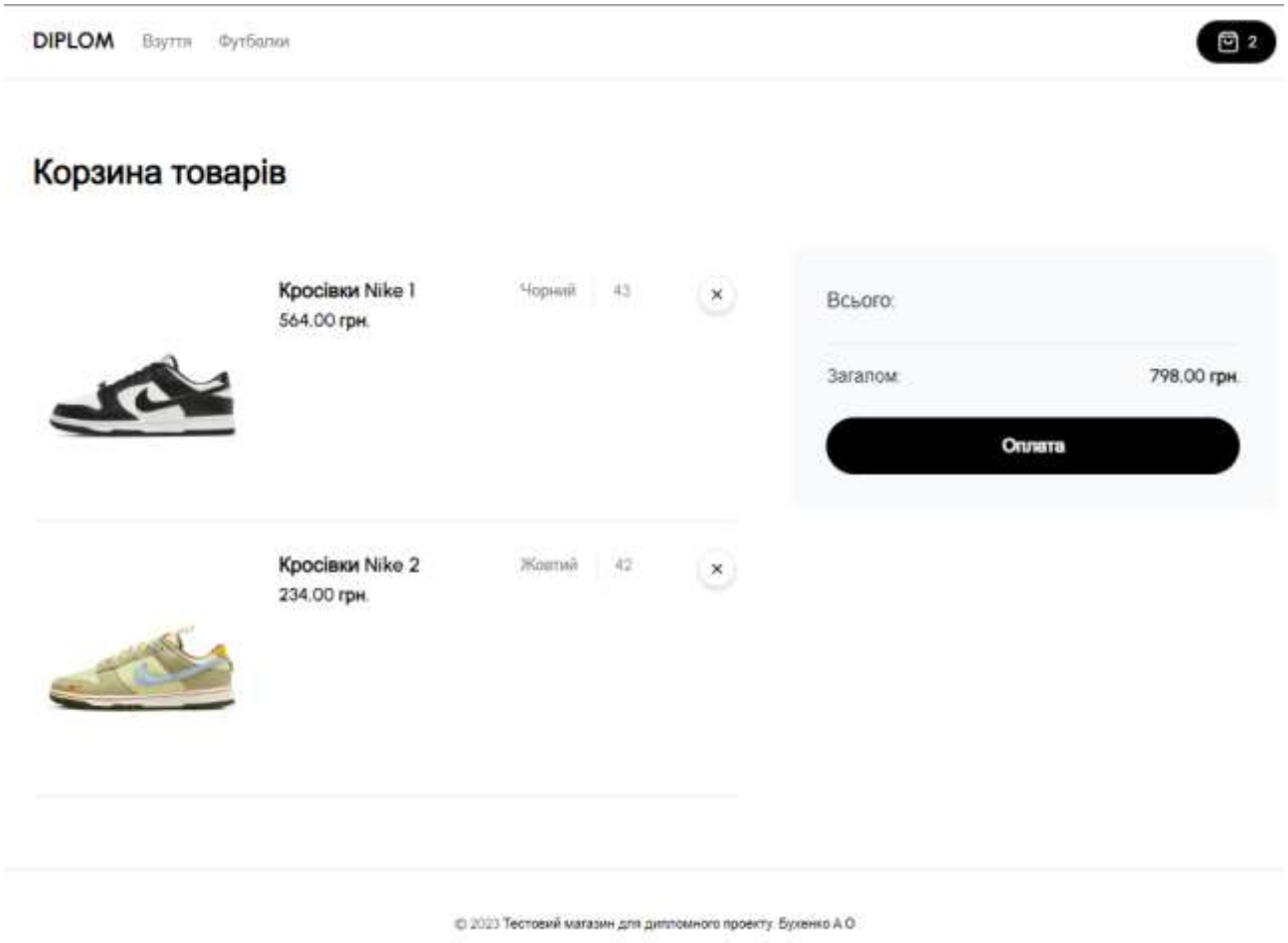


Рисунок 3.58 – Персональна сторінка кошика

Перейдемо до підключення функції оплати в магазин, для цього ми будемо використовувати сервіс “Stripe”, який дозволяє додавати функцію оплати як для реального інтернет магазину, так і для локального або тестового веб-магазину, як наш;

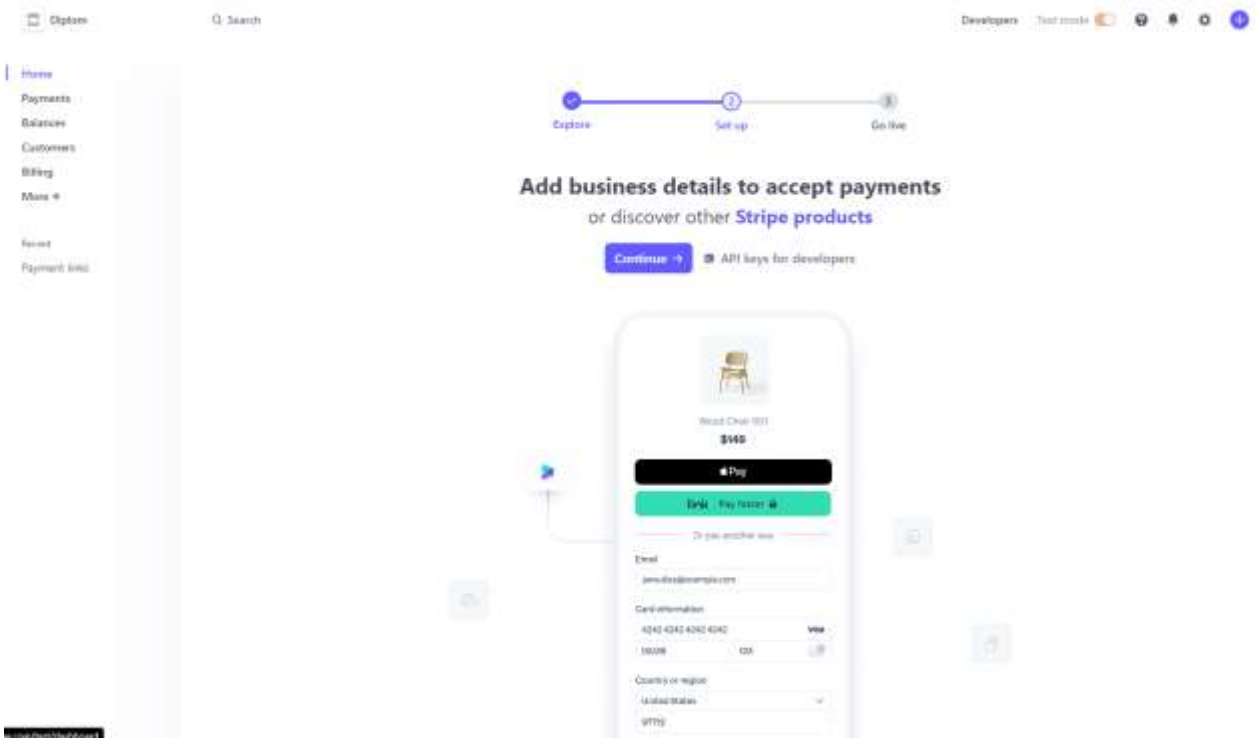


Рисунок 3.59 – Головна сторінка сервісу “Stripe”

Підлкючення схоже з іншими сервісами, які використовувалися в цій роботі. Потрібно авторизуватись та створити новий проект, після чого отримуємо API ключі, які потрібно додати до файлу “.env”, тільки не в веб-магазині, а в інформаційній системі (рис 3.60)

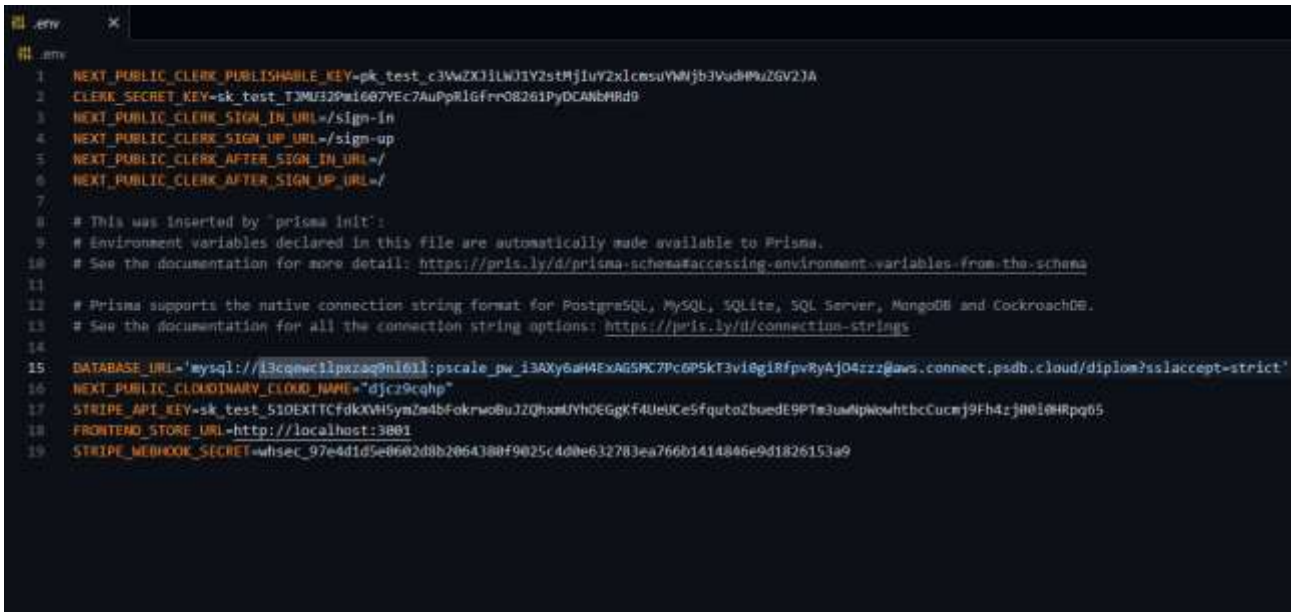


Рисунок 3.60 – Фінальний вигляд файлу “.env” інформаційної системи

Після підключення “Stripe” до інформаційної системо, його потрібно завантажити на комп’ютер, і запустити через термінал (рис 3.61)



Рисунок 3.61 – Запуск “Stripe”

Переходимо до кошику в магазині, і натискаємо на кнопку “Оплата”, після чого нас перекидає на сервіс оплати “Stripe” (рис 3.62), де ми бачимо назву нашого додатку, нашу суму оплати, методи оплати і поля для вводу інформації для оплати;

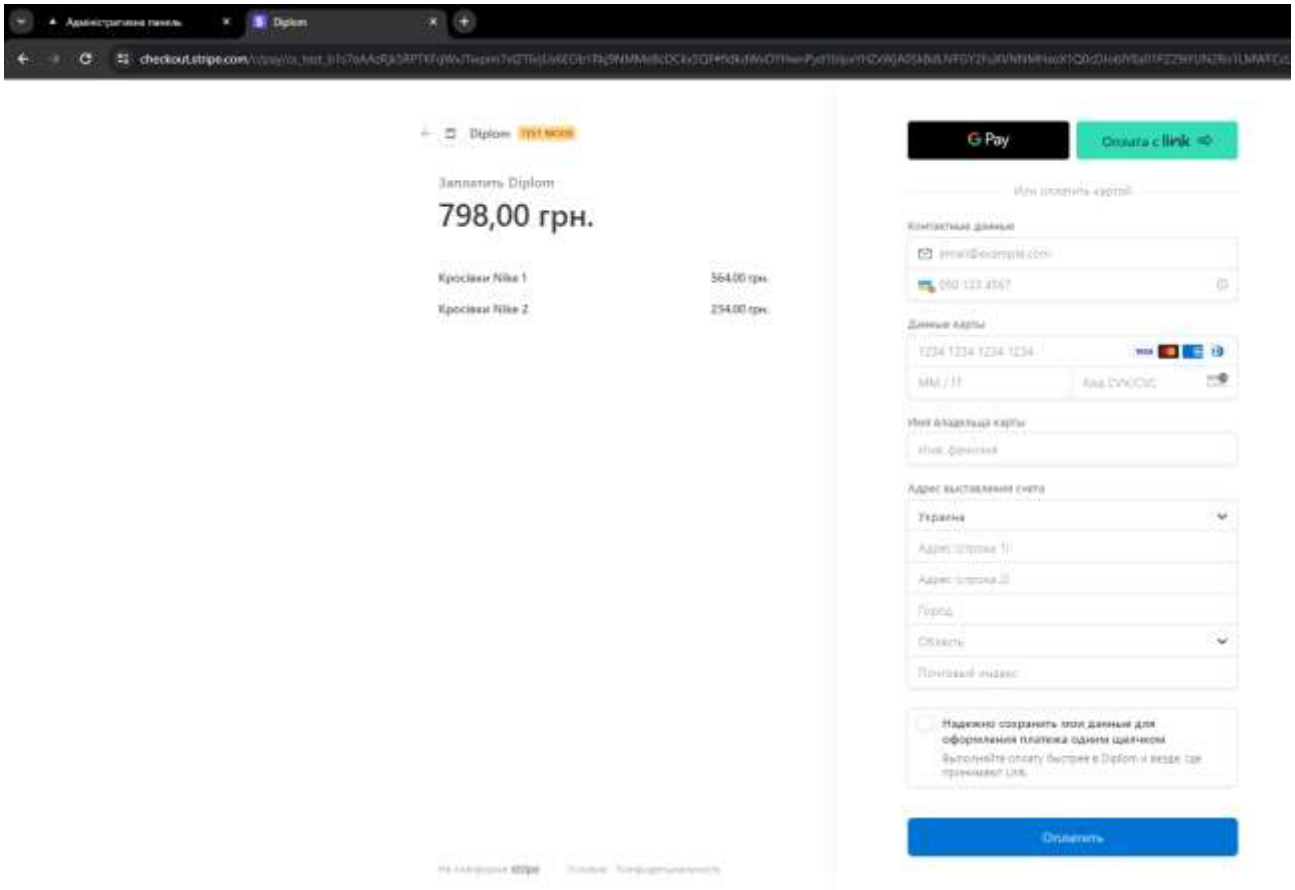


Рисунок 3.62 – Сервіс оплати “Stripe”

Після натискання кнопки “Оплатить” переходимо в наш термінал (рис 3.63) і можемо побачити POST коди зеленого кольору, що означає про успішне підключення і успішну оплату товару;

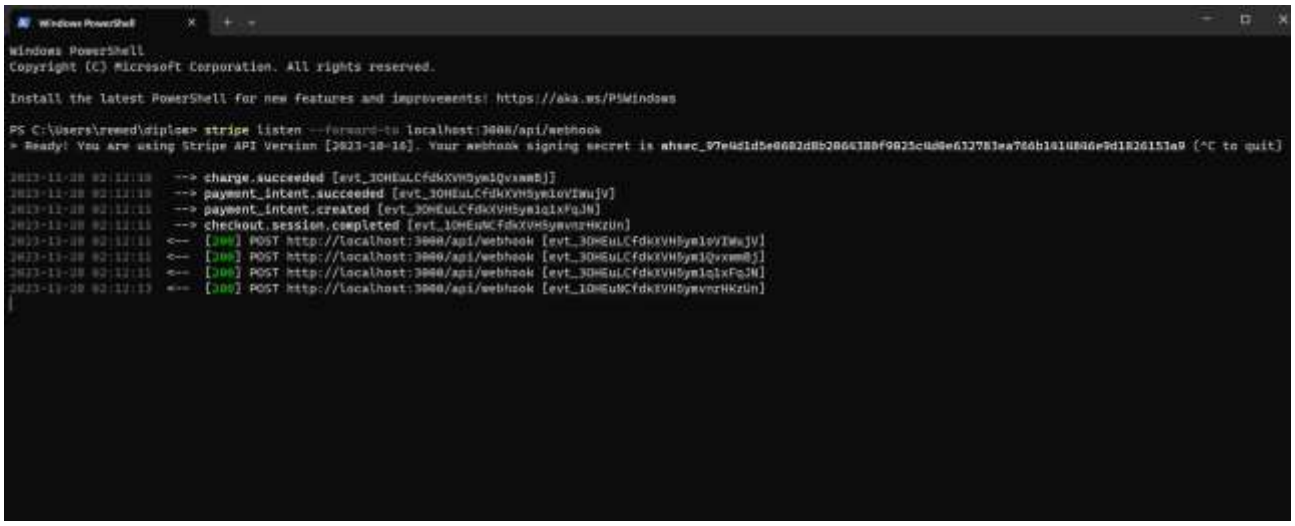


Рисунок 3.63 – Термінал “Stripe” з успішною оплатою товару

Також перейдемо до адміністративної панелі та функції “Замовлення” (рис 3.64) де ми можемо побачити в таблиці наше, успішно оплачене замовлення, на що вказує статус: True, під значенням “Оплата”;

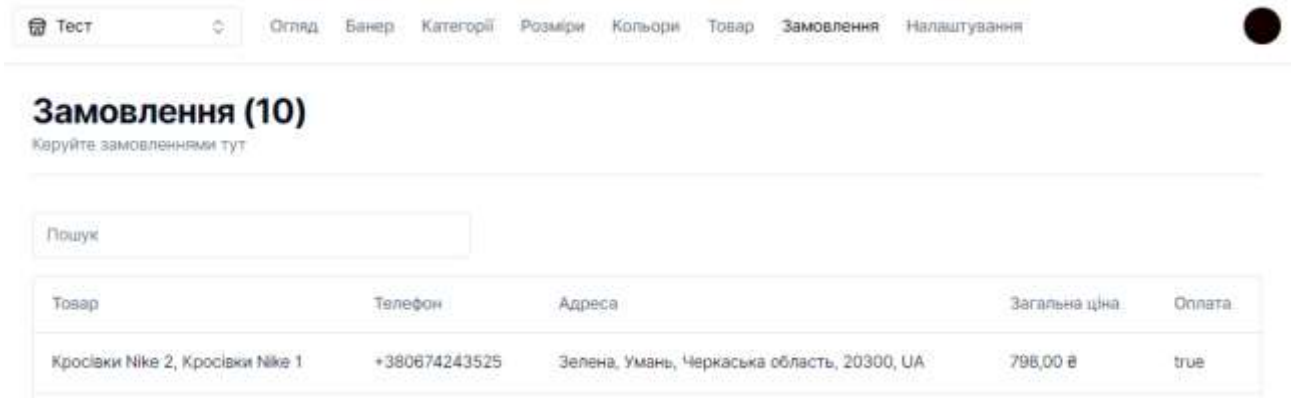
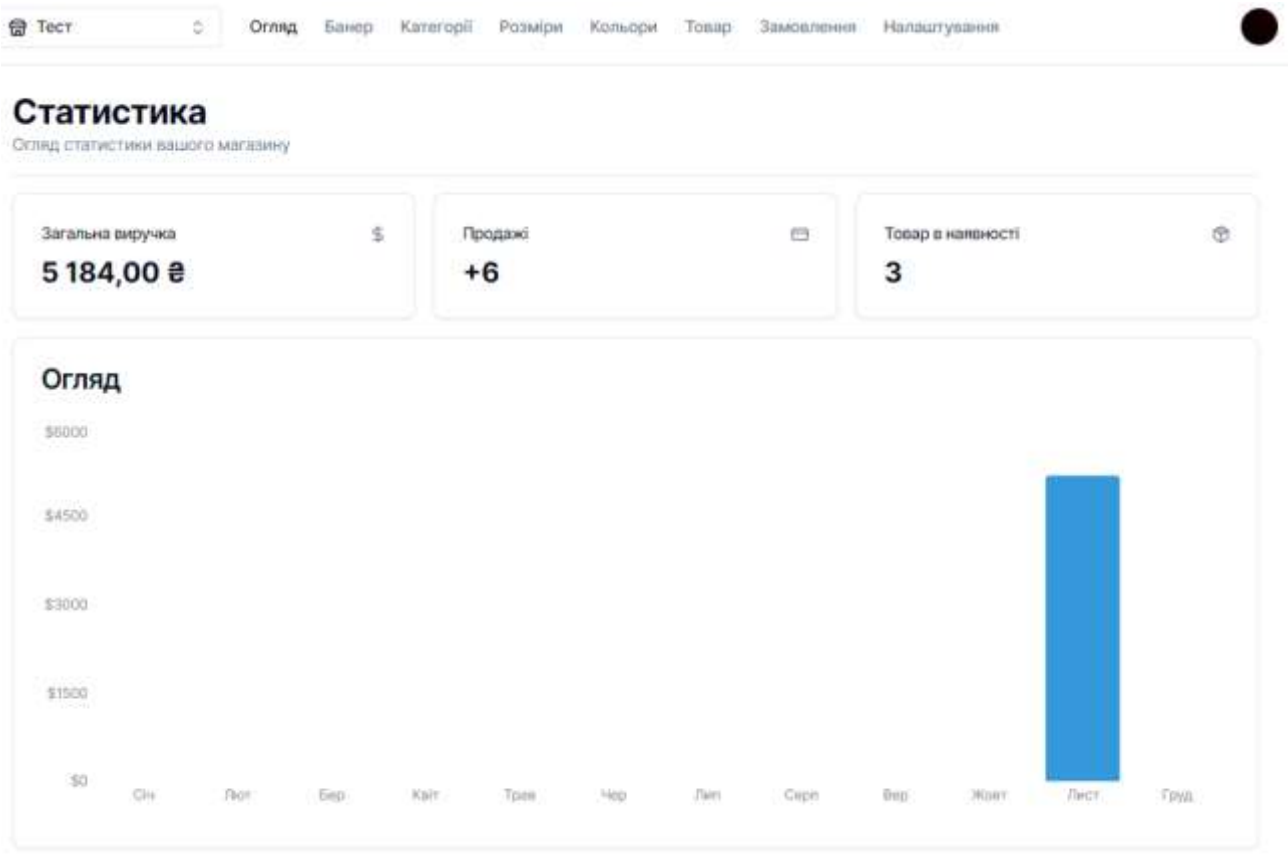


Рисунок 3.64 – Вкладка “Замовлення” з успішно оплаченим товаром

Наступний і фінальний крок в створені цієї інформаційної системи – створення візуалізації даних, в нашому випадку це – загальна сума продаж, кількість продаж всього, кількість товару в магазині та графік прибутку по місяцям. Ця функція знаходиться самою першою в адміністративній панелі

(рис 3.65), тому при заході в систему можна зразу переглянути всю статистику магазину



3.65 – Вкладка “Огляд” з статистикою магазина

Збір даних проводиться таким чином, наприклад блок “Товар в наявності” – тут відображається весь товар в якому не ввімкнена функція архівації, “Продажі” та “Загальна виручка” працюють однаково, якщо оплата замовлення пройшла успішно, то сума цього замовлення йде в “Загальну виручку” і добавляється +1 до “Продаж”.

ВИСНОВОК

Під час виконання магістерської роботи було проведено дослідження і аналіз головних аспектів інформаційної системи для адміністрування, була визначена актуальність і потреба в адмініструвані веб-додатків. На основі цих даних було розроблено інформаційну систему для легкого адміністрування та наявністю функції візуалізації даних.

Створена інформаційна система має основні функції адміністрування для обраної тематики веб-додатку, а саме:

- Додання товару, та редагування його;

- Керування банерами і категоріями в магазині;

- Перегляд активних замовлень та контактну інформацію покупців;

- Моніторинг статистики магазину;

Для написання даної інфорційної системи було використано такі інструменти як: мова програмування - JavaScript, чудовий та легкий фреймворк – Next.js, фреймовр для CSS – Tailwind, для роботи з базою даних було обрано – Prisma. Також було задіяно не мало сервісів як полегшили роботу над цим проектом, а саме: Clerk – сервіс який допоміг підєднати чудову функцію авторизації, PlanetScale – сервіс через який працювала база даних цього проекту, сервіс Cloudinary – через який завантажувались зображення в веб-додаток та сервіс який відповідає за оплату в веб-магазині – Stripe.

В результаті було створено робочу адміністративну панель та веб-додаток в вигляді інтернет магазину одягу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Learn Prisma In 60 Minutes [Електронний ресурс] – 2022 – Режим доступу до ресурсу: https://www.youtube.com/watch?v=RebA5Jrlwg&ab_channel=WebDevSimplified
2. Next.js Authentication with Clerk [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=UYUa8FRq2k8&ab_channel=ForThoseWhoCode
3. How I setup stripe in my Next.js side project [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=6qk7Jjpxs1M&ab_channel=WebDevCody
4. Uploading Images in Node.js with the Cloudinary Node SDK - Dev Hints [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=hGzVY88q8I0&ab_channel=Cloudinary
5. shadcn/ui [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://ui.shadcn.com/
6. github [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://github.com/
7. Next js 14 - что нового? [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=n1Y2VfJC6Ec&ab_channel=PurpleSchool%7CAntonLarichev
8. React Tutorial for Beginners [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=SqcY0GlETPk&ab_channel=ProgrammingwithMosh
9. ChatGPT [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://chat.openai.com/
10. Is a MANAGEMENT INFORMATION SYSTEMS degree worth it? [Електронний ресурс] – 2020 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=EMGvP6hav_U&ab_channel=ShaneHummus
11. Why Prisma Might Not Be a Good Choice for Your Project [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=Xk43fwSQ8No&ab_channel=Joshtriedcoding

12. Shadcn UI Crash Course # - Introduction & Setup [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=wcTzlJi2Oz4&ab_channel=NetNinja
13. Shadcn + Next.js 13 is HEAVEN | A QUICK GUIDE [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=ABbww4CFQSo&ab_channel=developedbyed
14. These 2 UI Libraries Are The Perfect Combo [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=48qgz7utECQ&ab_channel=Joshtriedcoding
15. Tailwind CSS React Tutorial [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=L2KHCWJ3gjs&ab_channel=AdrianTwarog
16. Authentication Doesn't Get Easier Than This [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=aQN4bJ1yc-k&ab_channel=Joshtriedcoding
17. Stripe & Firebase Tutorial • Add Payments To Your NextJS App [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=xi3F2Zv91UE&ab_channel=pixegami
18. Next.js 14 Full Course 2023 | Build and Deploy a Full Stack App Using the Official React Framework [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=wm5gMKuwSYk&ab_channel=JavaScriptMastery
19. Routing in Next.js – How to Use App Router in your Next Apps [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.freecodecamp.org/news/routing-in-nextjs/
20. E-Commerce with NextJS & Stripe [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=bG88qVHGGic&ab_channel=Icodestuff
21. Information Systems What is it? What does it mean? [Електронний ресурс] – 2017 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=dLfpZtIacbc&ab_channel=ITCareerQuestions
22. Information Technology In 4 Minutes [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=XZrckLYqdys&ab_channel=ShaneHummus

23. #11 | Починаємо роботи з Cloudinary | Розробка сайту піцерії | React, Typescript, Hasura [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=mVcCFl5eGg8&ab_channel=%D0%9D%D0%BE%D1%82%D0%B0%D1%82%D0%BA%D0%B8%D0%A5%D0%B5%D0%B4%D0%BE%D1%84%D0%B0
24. React JS Full Course 2023 | Build an App and Master React in 1 Hour [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=b9eMGE7QtTk&ab_channel=JavaScriptMastery
25. Learn Web Development and Actually Get a Job [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=glgmavKWg3Y&ab_channel=JamesCross
26. What is PlanetScale? [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=e63OWkVmawY&ab_channel=RyanChenkie
27. Connecting to your PlanetScale database [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=HDOfFC_Bl1E&ab_channel=PlanetScale
28. How to Build a REST API with Next.js 13 [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=-MFiza7ZRzs&ab_channel=DaveGray
29. Next.js 13 API Routes Tutorial] [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=varePWkGi8Y&ab_channel=NativeNotify
30. Clerk | Authentication and User Management [Електронний ресурс] Режим доступу до ресурсу: https://clerk.com/
31. shadcn/ui [Електронний ресурс] Режим доступу до ресурсу: https://ui.shadcn.com/
32. Node.js Ultimate Beginner's Guide in 7 Easy Steps [Електронний ресурс] – 2020 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=ENrzD9HAZK4&ab_channel=Fireship
33. Complete E-Commerce App with React, Strapi, Stripe | Shopping App Tutorial for Beginners [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=BCkWFblNLKU&t=40s&ab_channel=LamaDev
34. 7 better ways to create a React app [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=2OTq15A5s0Y&ab_channel=Fireship

35. Tailwind CSS Tutorial for Beginners (2023) – What YOU need to know [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=DenUCuq4G04&ab_channel=Lukas%7CWebDevelopment%26Design
36. 7 ways to deal with CSS [Електронний ресурс] – 2022 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=ouncVBiye_M&ab_channel=Fireship
37. MySQL - The Basics // Learn SQL in 23 Easy Steps [Електронний ресурс] – 2021 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=Cz3WcZLRaWc&ab_channel=Fireship
38. You need to learn SQL RIGHT NOW!! (SQL Tutorial for Beginners) [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=xiUTqnI6xk8&ab_channel=NetworkChuck
39. Next.js App Router: Routing, Data Fetching, Caching [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=gSSsZReIFRk&ab_channel=Vercel
40. NextJS Tutorial - All 12 Concepts You Need to Know [Електронний ресурс] – 2023 - Режим доступу до ресурсу: https://www.youtube.com/watch?v=5miHyP6lExg&t=27796s&ab_channel=CodeWithAntonio

Додаток А
“База даних”

Код файлу shema.prisma, де знаходяться всі моделі бази даних:

```
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url = env("DATABASE_URL")
  relationMode = "prisma"
}

// Модель магазину

model Store {
  id        String     @id @default(uuid())
  name      String
  userId    String
  billboards      Billboard[] @relation("StoreToBillboard")
  categories      Category[] @relation("StoreToCategory")
  products        Product[]  @relation("StoreToProduct")
  sizes           Size[]     @relation("StoreToSize")
  colors          Color[]     @relation("StoreToColor")
  orders          Order[]     @relation("StoreToOrder")
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}

// Модель банерів

model Billboard {
  id          String @id @default(uuid())
  storeId     String
  store       Store @relation("StoreToBillboard", fields: [storeId], references:
[id])
  label       String
  imageUrl    String
  categories  Category[]
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@index([storeId])
  }
```

```
// Модель категорій

model Category{
  id          String      @id @default(uuid())
  storeId     String
  store       Store     @relation("StoreToCategory", fields: [storeId], references:
[id])
  billboardId String
  billboard   Billboard   @relation(fields: [billboardId], references: [id])
  name        String
  products    Product[]   @relation("CategoryToProduct")
  createdAt   DateTime    @default(now())
  updatedAt   DateTime    @updatedAt

  @@index([storeId])
  @@index([billboardId])
}

// Модель розмірів для товарів

model Size {
  id          String    @id @default(uuid())
  storeId     String    // Зовнішній ключ для зберігання
  store       Store     @relation("StoreToSize", fields: [storeId], references:
[id])
  name        String
  value       String
  products    Product[] // Відношення до моделі продукту
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  @@index([storeId])
}

// Модель кольорів для товарів

model Color {
  id          String   @id @default(uuid())
  storeId     String   // Зовнішній ключ для зберігання
  store       Store    @relation("StoreToColor", fields: [storeId], references:
[id])
  name        String
  value       String
  products    Product[] // Відношення до моделі продукту
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  @@index([storeId])
}
```

```
// Модель товару

model Product {
  id          String    @id @default(uuid())
  storeId     String    // Зовнішній ключ для зберігання
  store       Store     @relation("StoreToProduct", fields: [storeId], references:
[id])
  categoryId  String    // Відношення до моделі продукту
  category    Category  @relation("CategoryToProduct", fields: [categoryId],
references: [id])
  name        String
  price       Decimal
  isFeatured  Boolean   @default(false)
  isArchived  Boolean   @default(false)
  sizeId      String
  size        Size      @relation(fields: [sizeId], references: [id])
  colorId     String
  color       Color     @relation(fields: [colorId], references: [id])
  images      Image[]
  orderItems  OrderItem[]   // Відношення до моделі замовлення
  createdAt   DateTime  @default(now())
  updatedAt   DateTime  @updatedAt

  @@index([storeId])
  @@index([categoryId])
  @@index([sizeId])
  @@index([colorId])
}

// Модель Замовлення

model Order {
  id        String    @id @default(uuid())
  storeId     String    // Foreign Key to Store
  store       Store     @relation("StoreToOrder", fields: [storeId], references:
[id])
  orderItems OrderItem[] // Відношення до моделі OrderItem
  isPaid     Boolean   @default(false)
  phone      String    @default("")
  address    String    @default("")
  createdAt DateTime  @default(now())
  updatedAt DateTime  @updatedAt

  @@index([storeId])
}

// Модель замовлення товару

model OrderItem {
  id        String  @id @default(uuid())
```

```
  orderId   String  // Зовнішній ключ на замовлення
  order     Order   @relation(fields: [orderId], references: [id])
  productId String  // Зовнішній ключ до продукту
  product   Product @relation(fields: [productId], references: [id])

  @@index([orderId])
  @@index([productId])
}

// Модель зображень

model Image {
  id          String   @id @default(uuid())
  productId   String   // Зовнішній ключ до продукту
  product     Product  @relation(fields: [productId], references: [id], onDelete:
Cascade)
  url         String   // URL-адреса зображення
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  @@index([productId])
}
```

Додаток Б
“Статистика магазину”

Код файлу page.tsx, де відображається статистика магазину:

```
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/card";
import { Heading } from "@/components/ui/heading";
import { Separator } from "@/components/ui/separator";
import { formatter } from "@/lib/utils";
import { CreditCard, DollarSign, Package } from "lucide-react";
import { getTotalRevenue } from "@/actions/get-total-revenue";
import { getSalesCount } from "@/actions/get-sales-count";
import { getGraphRevenue } from "@/actions/get-graph-revenue";
import { getStockCount } from "@/actions/get-stock-count";
import { Overview } from "@/components/overview";


interface DashboardPageProps {
    params: { storeId: string };
};


const DashboardPage: React.FC<DashboardPageProps> = async ({
    params
}) => {
    const totalRevenue = await getTotalRevenue(params.storeId);
  const graphRevenue = await getGraphRevenue(params.storeId);
  const salesCount = await getSalesCount(params.storeId);
  const stockCount = await getStockCount(params.storeId);

    return (
        <div className="flex-col">
            <div className="flex-1 space-y-4 p-8 pt-6">
             <Heading title="Статистика" description="Огляд статистики вашого
магазину" />
             <Separator />
             <div className="grid gap-4 grid-cols-3">
             <Card>
               <CardHeader className="flex flex-row items-center justify-between
space-y-0 pb-2">
                 <CardTitle className="text-sm font-medium">
                 Загальна виручка
                 </CardTitle>
                 <DollarSign className="h-4 w-4 text-muted-foreground" />
              </CardHeader>
              <CardContent>
              <div className="text-2xl font-bold">
                {formatter.format(totalRevenue)}
                </div>
            </CardContent>
             </Card>
```

```
            <Card>
            <CardHeader className="flex flex-row items-center justify-between
space-y-0 pb-2">
              <CardTitle className="text-sm font-medium">
                Продажі
              </CardTitle>
              <CreditCard className="h-4 w-4 text-muted-foreground" />
            </CardHeader>
            <CardContent>
              <div className="text-2xl font-bold">
                +{salesCount}
                </div>
            </CardContent>
          </Card>
             <Card>
            <CardHeader className="flex flex-row items-center justify-between
space-y-0 pb-2">
              <CardTitle className="text-sm font-medium">Товар в
наявності</CardTitle>
              <Package className="h-4 w-4 text-muted-foreground" />
            </CardHeader>
            <CardContent>
              <div className="text-2xl font-bold">{stockCount}</div>
            </CardContent>
          </Card>
            </div>
            <Card className="col-span-4">
          <CardHeader>
            <CardTitle>Огляд</CardTitle>
          </CardHeader>
          <CardContent className="pl-2">
            <Overview data={graphRevenue} />
          </CardContent>
        </Card>
        </div>
     </div>
    );
}

export default DashboardPage;
```

Додаток В
“Створення категорій”

Код файлу categorty-form.tsx, який відповідає за створення категорій:

```
"use client";
import * as z from "zod";
import { useState } from "react";
import { Category, Billboard } from "@prisma/client";
import { Trash } from "lucide-react";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";


import { Heading } from "@/components/ui/heading";
import { Button } from "@/components/ui/button";
import { Separator } from "@/components/ui/separator";
import { Form, FormControl, FormField, FormItem, FormLabel, FormMessage } from
"@/components/ui/form";
import { Input } from "@/components/ui/input";
import toast from "react-hot-toast";
import axios from "axios";
import { useParams, useRouter } from "next/navigation";
import { AlertModal } from "@/components/modals/alert-modal";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
"@/components/ui/select"

const formSchema = z.object({
    name: z.string().min(1),
    billboardId: z.string().min(1),
  });


type CategoryFormValues = z.infer<typeof formSchema>;


interface CategoryFormProps {
    initialData: Category | null;
    billboards: Billboard[];
}

export const CategoryForm: React.FC<CategoryFormProps> = ({
    initialData,
    billboards
}) => {
    const params = useParams();
    const router = useRouter();

    const [open, setOpen] = useState(false);
    const [loading, setLoading] = useState(false);
```

```
    const title = initialData ? 'Редагувати Категорію' : 'Створити Категорію';
    const description = initialData ? 'Редагувати Категорію' : 'Додати нову
категорію';
    const toastMessage = initialData ? 'Категорія зміненна' : 'Категорія створена';
    const action = initialData ? 'Зберегти змінни' : 'Створити';


    const form = useForm<CategoryFormValues>({
        resolver: zodResolver(formSchema),
        defaultValues: initialData || {
          name: '',
          billboardId: '',
        }
      });

      const onSubmit = async (data: CategoryFormValues) => {
        try{
            setLoading(true);
            if (initialData) {
              await
axios.patch(`/api/${params.storeId}/categories/${params.categoryId}`, data);
      } else {
              await axios.post(`/api/${params.storeId}/categories`, data);
            }
            router.refresh();
            router.push(`/${params.storeId}/categories`);
            toast.success(toastMessage);
        }   catch (error) {
            toast.error('Щось пішло не так :(');
          } finally {
            setLoading(false);
          }
      };

      const onDelete = async () => {
        try {
          setLoading(true);
          await
axios.delete(`/api/${params.storeId}/categories/${params.categoryId}`);
          router.refresh();
          router.push('/');
          router.push(`/${params.storeId}/categories`);
          toast.success('Категорію видалено');
        } catch (error: any) {
          toast.error('Спочатку переконайтеся, що ви видалили всі продукти в
категорії.');
        } finally {
          setLoading(false);
          setOpen(false);
```

```
        }
      }

    return (
        <>
        <AlertModal
        isOpen={open}
        onClose={() => setOpen(false)}
        onConfirm={onDelete}
        loading={loading}
      />
        <div className="flex items-center justify-between">
           <Heading
           title={title}
           description={description}
           />

          {initialData && (
           <Button
           disabled={loading}
           variant="destructive"
           size="sm"
           onClick={() => setOpen(true)}
           >
            <Trash className="h-4 w-4" />
           </Button>
           )}

        </div>
        <Separator />
        <Form {...form}>
           <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-8 w-
full">
            <div className="md:grid md:grid-cols-3 gap-8">
                <FormField
                control={form.control}
                name="name"
                render={({ field }) => (
                    <FormItem>
                        <FormLabel>Ім'я</FormLabel>
                        <FormControl>
                    <Input disabled={loading} placeholder="Ім'я Категорії"
{...field} />
                  </FormControl>
                  <FormMessage />
                </FormItem>
                )}
                />
                <FormField
              control={form.control}
```

```
              name="billboardId"
              render={({ field }) => (
                <FormItem>
                  <FormLabel>Дошка</FormLabel>
                  <Select disabled={loading} onValueChange={field.onChange}
value={field.value} defaultValue={field.value}>
                    <FormControl>
                      <SelectTrigger>
                        <SelectValue defaultValue={field.value}
placeholder="Вибрати Дошку" />
                      </SelectTrigger>
                    </FormControl>
                    <SelectContent>
                      {billboards.map((billboard) => (
                        <SelectItem key={billboard.id}
value={billboard.id}>{billboard.label}</SelectItem>
                      ))}
                    </SelectContent>
                  </Select>
                  <FormMessage />
                </FormItem>
              )}
            />
            </div>
            <Button disabled={loading} className="ml-auto" type="submit">
                {action}
            </Button>
          </form>
        </Form>
        </>
    );
};
```

Додаток Г
“Загальна кількість продаж”

Код файлу get-sales-count.ts, який відповідає за збір інформації по загальній кількості продаж:

```
import prismadb from "@/lib/prismadb";

export const getSalesCount = async (storeId: string) => {
  const salesCount = await prismadb.order.count({
    where: {
      storeId,
      isPaid: true
    },
  });

  return salesCount;
};
```

Додаток Д
“ Загальна сума продаж”

Код файлу get-total-revenue.ts, який відповідає за збір інформації по загальній сумі продаж:

```
import prismadb from "@/lib/prismadb";

export const getTotalRevenue = async (storeId: string) => {
  const paidOrders = await prismadb.order.findMany({
    where: {
      storeId,
      isPaid: true
    },
    include: {
      orderItems: {
        include: {
          product: true
        }
      }
    }
  });

  const totalRevenue = paidOrders.reduce((total, order) => {
    const orderTotal = order.orderItems.reduce((orderSum, item) => {
      return orderSum + item.product.price.toNumber();
    }, 0);
    return total + orderTotal;
  }, 0);

  return totalRevenue;
};
```

Додаток Ж
“Графік продаж”

Код файлу get-graph-revenue.ts, який відповідає за графік продаж по місяцям:

```
import prismadb from "@/lib/prismadb";

interface GraphData {
  name: string;
  total: number;
}

export const getGraphRevenue = async (storeId: string): Promise<GraphData[]> => {
  const paidOrders = await prismadb.order.findMany({
    where: {
      storeId,
      isPaid: true,
    },
    include: {
      orderItems: {
        include: {
          product: true,
        },
      },
    },
  });

  const monthlyRevenue: { [key: number]: number } = {};

  // Групування замовлень по місяцях і підсумовування виручки

  for (const order of paidOrders) {
    const month = order.createdAt.getMonth(); // 0 для січня, 1 для лютого і так
далі ...
    let revenueForOrder = 0;

    for (const item of order.orderItems) {
      revenueForOrder += item.product.price.toNumber();
    }

    // Додавання доходу для цього замовлення до відповідного місяця
    monthlyRevenue[month] = (monthlyRevenue[month] || 0) + revenueForOrder;
  }

  // Перетворення згрупованих даних у формат, очікуваний графіком
  const graphData: GraphData[] = [
    { name: "Січ", total: 0 },
    { name: "Лют", total: 0 },
    { name: "Бер", total: 0 },
    { name: "Квіт", total: 0 },
```

```
    { name: "Трав", total: 0 },
    { name: "Чер", total: 0 },
    { name: "Лип", total: 0 },
    { name: "Серп", total: 0 },
    { name: "Вер", total: 0 },
    { name: "Жовт", total: 0 },
    { name: "Лист", total: 0 },
    { name: "Груд", total: 0 },
  ];

 // Заповнення відомостей про доходи
  for (const month in monthlyRevenue) {
    graphData[parseInt(month)].total = monthlyRevenue[parseInt(month)];
  }

  return graphData;
};
```

Додаток З

“Статистика кількості товару”

Код файлу get-stock-count.ts, який відповідає за статистику кількості товару в магазині:

```
import prismadb from "@/lib/prismadb";

export const getStockCount = async (storeId: string) => {
  const stockCount = await prismadb.product.count({
    where: {
      storeId,
      isArchived: false,
    }
  });

  return stockCount;
};
```