МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЖИТОМИРСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Кафедра інженерії програмного забезпечення

**КУРСОВИЙ ПРОЕКТ**

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни: «Інтернет-програмування»

на тему:

**«ІНТЕРНЕТ-МАГАЗИН САДІВНИЧИХ ТОВАРІВ»**

студента II курсу групи ПІ-60

спеціальності 121 «Інженерія програмного забезпечення»

Хіміча Владислава Олеговича

(прізвище, ім’я та по-батькові)

Керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ,

,

Дата захисту: " \_\_\_ " \_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ р.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оцінка: ECTS \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Болотіна В.В.\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чижмотря О.Г.\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кузьменко О.В.\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Житомир – 2020

ЖИТОМИРСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет інформаційно-комп’ютерних технологій

Кафедра комп’ютерної інженерії та кібербезпеки

Освітній рівень: магістр

Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Зав. кафедри

\_\_\_\_\_\_\_\_\_\_\_ Пулеко І.В.

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_р.

ЗАВДАННЯ

НА КУРСОВИЙ ПРОЕКТ СТУДЕНТУ

Громському Олександру Олександровичу

1. Тема роботи: Розробка інтернет-магазину садівничих товарів

керівник роботи: к.т.н., доцент Морозов Андрій Васильович.

1. Строк подання студентом: “13”\_травня\_\_2020р.
2. Вихідні дані до роботи: Розробити інтернет-магазин садівничих товарів
3. Зміст розрахунково-пояснювальної записки(перелік питань. Які підлягають розробці)
   * + 1. Аналіз проблематики, методів та засобів вирішення задач
       2. Проектування та розробка програмного забезпечення
       3. Опис роботи з програмним додатком та його тестування
4. Перелік графічного матеріалу(з точним зазначенням обов’язкових креслень)

1)Презентація PowerPoint,

2)Репозиторій з повним кодом проекту:

<https://gitlab.com/pi60_hvo/course-project-sadivnyk>

1. Консультанти розділів проекту (роботи)

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Прізвище, ініціали та посади консультанта | Підпис, дата | |
| завдання  видав | завдання прийняв |
| 1, 2, 3 | Болотіна В.В., асистент кафедри КІ та КБ |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Дата видачі завдання “\_03\_” лютого 2019 р.

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів курсової роботи | Строк виконання етапів роботи | Примітки |
| 1 | Постановка задачі | 10.05.20 |  |
| 2 | Пошук, огляд та аналіз аналогічних розробок | 14.05.20 |  |
| 3 | Формулювання технічного завдання | 16.05.20 |  |
| 4 | Опрацювання літературних джерел | 18.05.20 |  |
| 5 | Проектування структури | 20.05.20 |  |
| 6 | Написання програмного коду | 06.06.20 |  |
| 7 | Відлагодження | 07.06.20 |  |
| 8 | Написання пояснювальної записки | 08.06.20 |  |
| 9 | Захист | 10.06.20 |  |

**КАЛЕНДАРНИЙ ПЛАН**

**Студент** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Хіміч В.О.

(підпис) (прізвище та ініціали)

**Керівник роботи** \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Болотіна В. В.

(підпис) (прізвище та ініціали)

# РЕФЕРАТ

Пояснювальна записка до курсового проекта на тему «Інтернет-магазин садівничих товарів» складається з трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 46 сторінках друкованого тексту.

Пояснювальна записка має 28 сторінок додатків. Список використаних джерел містить 6 найменувань і займає 1 сторінку. В роботі наведено 43 рисунка. Загальний обсяг роботи – 83 сторінок.

У першому розділі було проаналізовано задачі, наведено приклади сайтів-аналогів, досліджено технології розробки та поставлені задачі, які має виконувати веб-сайт.

У другому розділі було спроектовано загальний алгоритм роботи веб-сайту, розроблено функціональні алгоритми його роботи.

У третьому розділі проведено тестування програмного продукту.

Висновок містить в собі результати виконаної роботи з створення веб-сайту.

У додатку представлений лістинг розробленого програмного продукту.

Ключові слова: Model-View-Controller(MVC), Back-end, Front-end, Сервер, Система керування вмістом

# ЗМІСТ

[РЕФЕРАТ 4](#_Toc42678970)

[ЗМІСТ 5](#_Toc42678971)

[Перелік умовних скорочень 7](#_Toc42678972)

[ВСТУП 8](#_Toc42678973)

[РОЗДІЛ 1 АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ 9](#_Toc42678974)

[1.1 Аналіз задачі, засобів та методів її вирішення 9](#_Toc42678975)

[1.2 Аналіз існуючих веб-сайтів за тематикою курсового проекту (роботи) 10](#_Toc42678976)

[1.2.1 Vist 11](#_Toc42678977)

[1.2.2 Leto 11](#_Toc42678978)

[1.3 Технічне завдання на курсову роботу 13](#_Toc42678979)

[1.3.1 Загальне положення 13](#_Toc42678980)

[1.3.2 Найменування розробника та замовника. 13](#_Toc42678981)

[1.3.3 Підстава для розробки 13](#_Toc42678982)

[1.3.4 Вимоги до функціональних характеристик 13](#_Toc42678983)

[РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 16](#_Toc42678984)

[2.1 Проектування загального алгоритму роботи програми 16](#_Toc42678985)

[2.2 Розробка функціональних алгоритмів роботи програми 18](#_Toc42678986)

[2.3 Розробка програмного забезпечення 30](#_Toc42678987)

[РОЗДІЛ 3 ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ 39](#_Toc42678988)

[3.1 Опис роботи з програмним додатком 39](#_Toc42678989)

[3.2 Тестування роботи програмного забезпечення 53](#_Toc42678990)

[ВИСНОВКИ 54](#_Toc42678991)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55](#_Toc42678992)

[ДОДАТКИ 56](#_Toc42678993)

[Додаток А 57](#_Toc42678994)

[Додаток Б 58](#_Toc42678995)

[Додаток В 68](#_Toc42678996)

# Перелік умовних скорочень

CMS - Content Management System

API - Application Programming Interface

JSON – JavaScript Object Notation

XML – Extensible Markup Language

ORM – Object Relational Mapping

MVC – Model View Controller

JS – JavaScript

HTML – Hyper Text Markup Language

# ВСТУП

Актуальність теми: людство майже з самого початку свого існування займалося спочатку рослинництвом, а згодом садівництвом. Час йшов і ця справа, як і все навколо нас, розвивалася: спочатку повільно, а потім, з настанням цифрової ери, дуже стрімко. І оскільки людям завжди буде потрібна їжа, тема садівництва та рослинництва буде завжди актуальною. Нині швидко розвивається як і велике фермерство, так і мале, домашнє, і для всього цього необхідні відповідні інструменти. Позаяк ми живемо в високотехнологічній ері, людям буде значно комфортніше замовити потрібне приладдя через інтернет та не витрачати купу часу для пошуку інструментарію в офлайн магазинах. З іншого боку, офлайн магазини можуть значно підвищити свій прибуток завдяки частковому переходу до продажу у всесвітній павутині. Підсумовуючи все, що було сказано вище, можна сказати, що тема цього курсового проекту буде вічно актуальною.

Головна мета: дослідження особливостей проектування та реалізації системи адміністрування онлайн-магазину садівничих товарів.

*Об’єкт дослідження:* методи та засоби розробки CMS для інтернет-магазину садівничих товарів

*Предмет дослідження:* використання веб-технологій для реалізації системи керування контентом.

# АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

## Аналіз задачі, засобів та методів її вирішення

Метою цього курсового проекту є написання інтернет-магазину садівничих товарів. Загалом це завдання є досі складним та комплексним, а отже буде доцільним розділити його на три основні частини:

*Серверна частина (Back-end)* – частина веб-сайту, де буде відбуватися основна робота, непомітна для користувача.

*Зовнішній інтерфейс (Front-end)* – частина веб-сайту, з якою працює користувач.

* *Система керування вмістом (CMS)* - частина веб-сайту, де адміністратори сайту можуть керувати його вмістом.

Після визначення основних частин веб-сайту буде хорошим рішенням поділити розробку на менші частини, щоб ясно розуміти, скільки роботи залишилось та над чим працювати далі. Розробка інтернет-магазину садівничих товарів буде поділена на такі частини:

* Визначення основного функціоналу, який буде присутній у веб-сайті
* Розподілення основного функціоналу на менші частини задля структурованого процесу розробки
* Створення схеми бази даних та визначення сутностей, присутніх у ній
* Написання системи керування вмістом
* Написання серверної частини веб-сайту
* Написання зовнішнього вигляду веб-сайту
* Відлагодження кожної з частин проекту

Далі буде доцільним визначення з основним функціоналом веб-сайту і його можна розподілити на дві частини: для звичайного користувача та для адміністратора. Функціонал для звичайного користувача:

* Створення акаунту
* Можливість ввійти в свій акаунт
* Зміна персональних даних в особистому кабінеті
* Можливість пошуку товару за категорією
* Можливість пошуку товару за назвою
* Фільтрація товарів
* Додання товарів до корзини
* Зміна кількості товарів у корзині або повне видалення
* Замовлення товару
* Перегляд здійснених замовлень

Функціонал для адміністратора веб-сайту:

* Керування власним акаунтом адміністратора
* Створення нових записів, необхідних для відображення товарів інтернет-магазину
* Редагування вже існуючих записів
* Видалення записів

Після визначення з функціоналом буде доречним подумати про технології, які використовуватимуться при розробці інтернет-магазину. Для бек-енду було обрано платформу Node.JS, яка дозволяє виконувати Javascript код на сервері. Задля спрощення розробки серверної частини на Node.JS було обрано, мабуть, найпопулярніший фреймворк для цієї технології – Express. Системою керування даними буде MySQL у зв’язці з ORM для платформи Node.JS Sequelize, тому що це значно полегшить роботу з базою даних. Для фронт-енду буде використано класичний стек HTML, CSS, Javascript, окрім того, що замість CSS буде використано препроцесор Sass.

## Аналіз існуючих веб-сайтів за тематикою курсового проекту (роботи)

На жаль, у простого користувача немає можливості отримати доступ до системи керування вмістом будь-якого інтернет-магазину, тому у цьому підпункті буде порівнюватись зовнішній вигляд веб-сайту та функціонал доступний користувачу. Для порівняння було обрано два онлайн-магазини схожої з цим курсовим проектом тематики: Vist та Leto

### Vist

Vist – один з найбільших гравців ринку не лише садівничих товарів, але й будівельних. Їх сайт досить складний та великий: має обширний каталог, безліч товарів, переведений на декілька мов та багато іншого. Проте зовнішній вигляд досить застарілий, також при використанні сайтом постійно можна знайти недоробки, які псують враження про сайт. Ще мінусами є те, що на товар не вказується ціна, а зареєструватися на сайті можуть лише приватні підприємці. Однак хочу відмітити, що структура сайту є досить вдалою та інтуїтивно зрозумілою.

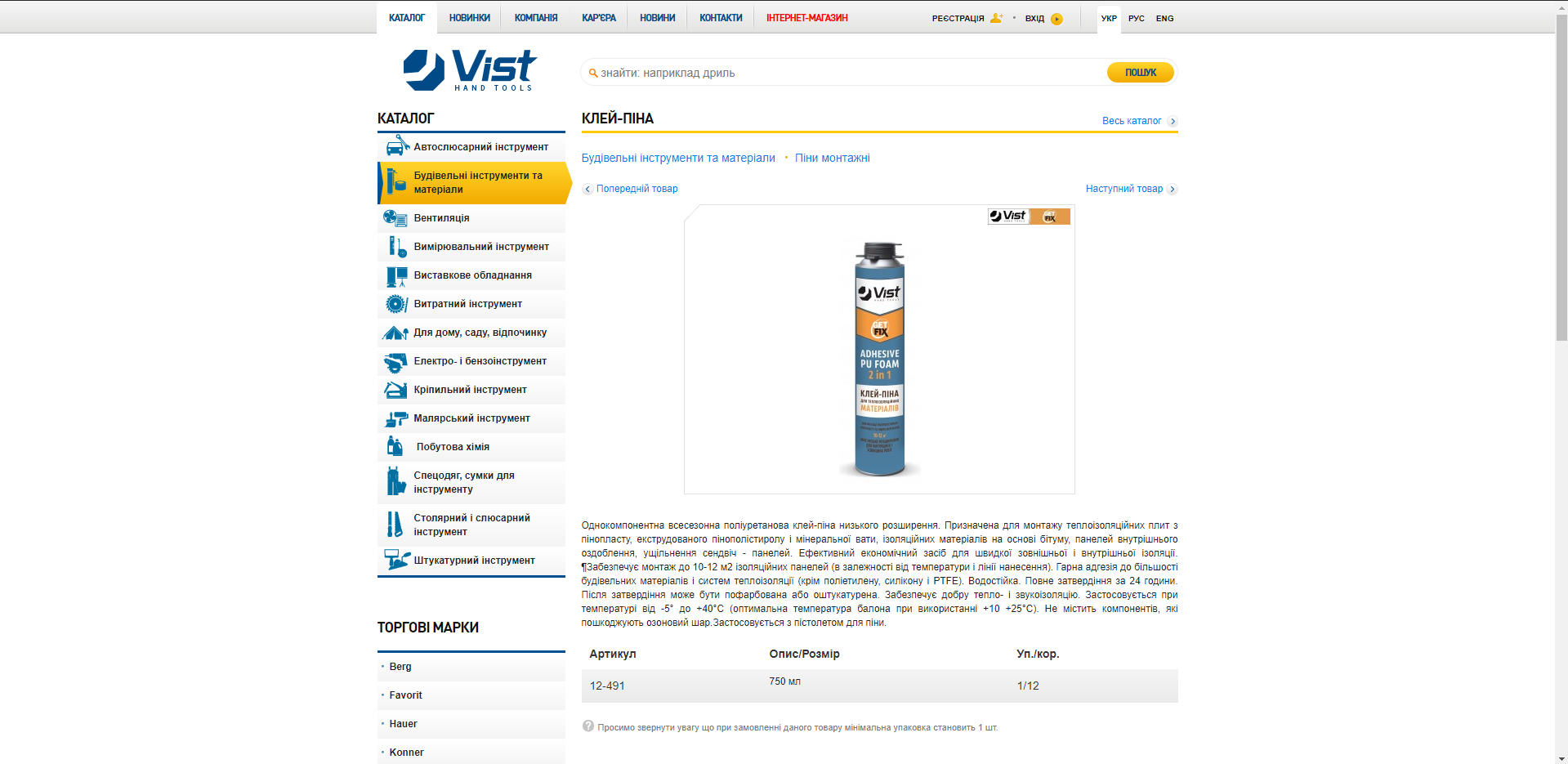


Рисунок 1.1. Сторінка продукту з Vist

### Leto

З інтернет-магазином Leto ситуація теж неоднозначна: якщо користуватися сайтом з десктопу, то все просто чудово, адже сайт має привабливий, сучасний дизайн та безліч функціоналу. Але при вході в цей інтернет-магазин з мобільного пристрою сайтом стає неможливо користуватись: меню та кнопки “залазять” одне на одного, блоки з текстом займають ширину більшу, аніж ширина дисплею смартфону та багато іншого. На відміну від сайту Vist, Leto показує ціну товарів та надає змогу зареєструватися звичайному користувачу.

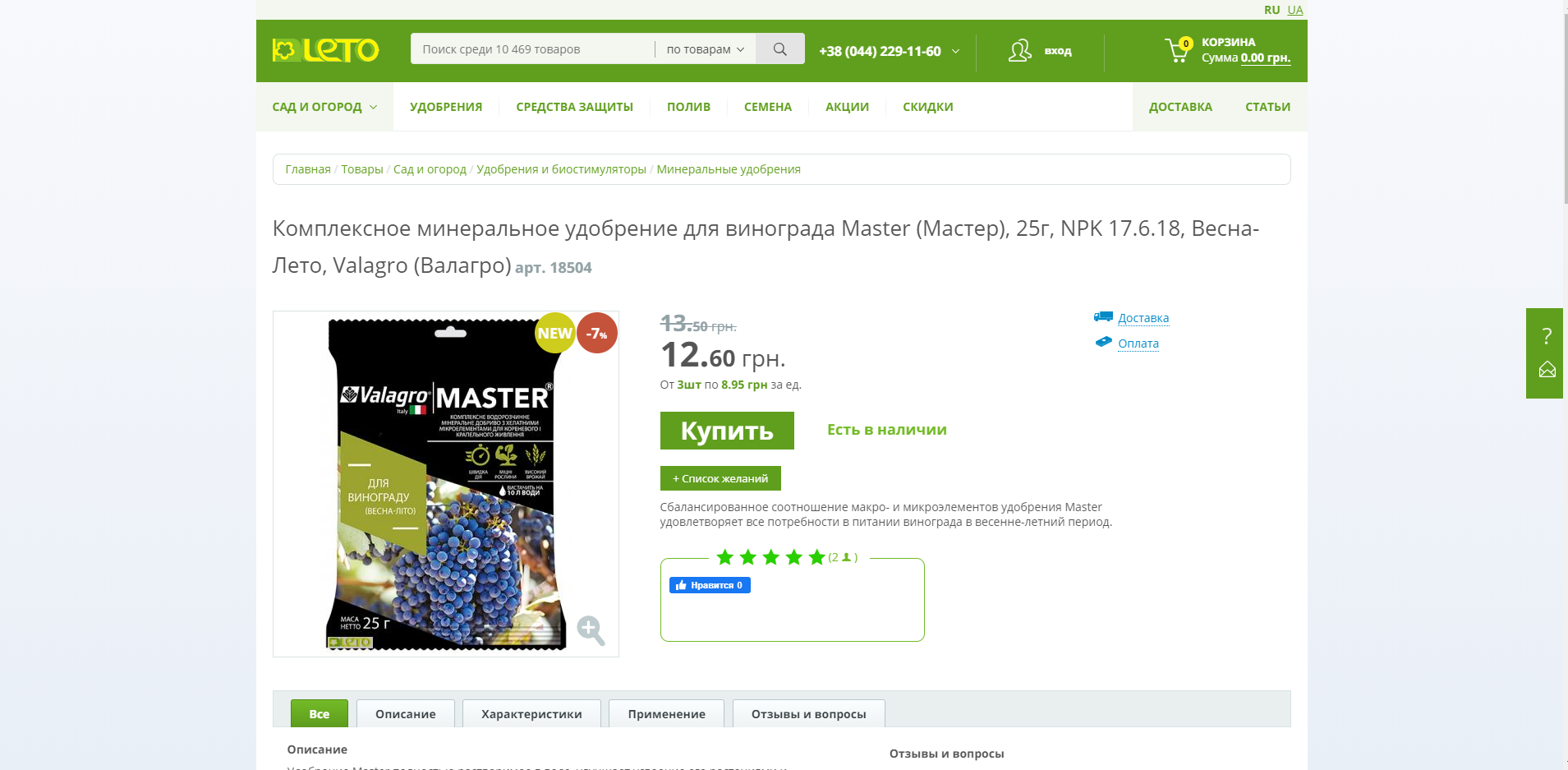


Рисунок 1.2. Сторінка продукту з Leto

Таблиця 1.1 Порівняння

|  |  |  |  |
| --- | --- | --- | --- |
| Назва / Критерій | Функціональність | Доступність | Вигляд |
| Vist | + | - | +- |
| Leto | + | + | +- |

*Висновки до порівняльної таблиці:*  обидва веб-сайти мають хорошу функціональність, якій позаздрять будь-який малий інтернет-магазин. Але обидва інтернет-магазини мають проблеми з зовнішнім виглядом. Якщо говорити про індивідуальні проблеми, то Vist орієнтований на приватних підприємців та не дає змоги замовити товар звичайному користувачу.

## Технічне завдання на курсову роботу

### Загальне положення

*Найменування програмного засобу:* повне найменування програмної системи: "Інтернет-магазин садівничих товарів та система керування вмістом для нього"(надалі “веб-сайт”). Коротка назва програмної системи - "Інтернет-магазин"

*Призначення розробки та область застосування:* веб-сайт призначений для пошуку товарів та їх замовлення. Також присутня система контролю контенту веб-сайту для адміністраторів.

### Найменування розробника та замовника.

*Розробник даного продукту****:*** студент групи ПІ-60 Хіміч Владислав Олегович (надалі "розробник").

*Замовник програмного продукту****:*** кафедра інженерія програмного забезпечення Державного університету «Житомирська політехніка» в межах виконання курсової з дисципліни «Інтернет програмування» Болотіна Вікторівна Василівна, Морозов Андрій Василійович

### Підстава для розробки

*Документ на підставі якого ведеться розробка:* Робота ведеться на підставі навчального плану за напрямом 121 «Інженерія програмного забезпечення».

### Вимоги до функціональних характеристик

Загальні вимоги:

* Можливість знайти товар
* Можливість додати товар у корзину
* Можливість змінювати кількість замовленого товару
* Можливість замовити товар
* Змінювати особисті дані
* Редагувати записи, пов’язані з товарами (для адміністратора)
* Додавати записи, пов’язані з товарами (для адміністратора)
* Видаляти записи, пов’язані з товарами (для адміністратора)

*Склад виконуваних функцій:* Розробити інтернет-магазин, що підтримує виконання наступних операцій:

* Пошук товарів за назвою
* Відображення товарів за категоріями
* Фільтрація знайдених товарів
* Створення облікового запису, можливість зміни імені, прізвища, по-батькові, контактного номеру, дати народження, статі та паролю
* Додання товару до корзини
* Операції з кількістю замовлених товарів такі, як: додавання, зменшення, видалення
* Надання адміністраторам інтерфейсу для зручного додавання, редагування та видалення записів, пов’язаних з товарами.

*Організація вхідних і вихідних даних:* вхідними даними є інформація про садівничі товари (назва, опис, варіації товарів, ціна)

Організація вхідних і вихідних даних повинна відповідати інформаційній структурі виконуваних з операцій, вхідним та вихідним паперовим документами.

Введення оперативних даних повинно виконуватися з використанням діалогових екранних форм, побудованих на основі візуальних компонентів. Введення даних виконується на основі затверджених форм документів: анкета, заява, інформаційна довідка та в режимі online оператором зі слів користувача.

Часові характеристики і розмір пам'яті, необхідної для роботи програми: час реакції програми на дії користувача (маніпуляції з пристроями введення даних) не повинен перевищувати 0,25 с.

Час виконання команд меню не більше 0.1 с.

Відображення масивів даних за запитами не більше 2 с.

Доступність БД – 90% цілодобово.

Операції з’єднання з БД не більше 1 хвилини.

Обсяг оперативної пам'яті, необхідний для роботи програми не менше 1Гб.

Дисковій простір, необхідний для збереження програми і файлів даних не більше 300 Мбат для робочої станції та 20 ГБайт..

Інсталяційний пакет програми, що містить у складі БД не повинні перевищувати 100 Мбайт.

Вимоги до методів рішення і мов програмування

Вибір методів рішення здійснюється розробникам без узгодження з замовником.

# ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Проектування загального алгоритму роботи програми

Розпочати проектування варто з найвищого рівня абстракції – структури веб-сайту. Загалом у світі веб-розробки є дві найбільш поширені архітектури – монолітна та мікросервісна. Мікросервісна архітектура є досить складною, але має великі переваги. Її суть полягає в тому, що веб-сайт розподілений на частини, які слабко зв’язані між собою, з цього слідує, що при помилці в якомусь з мікросервісів, це не вплине загалом на роботу всього серверу. Ще одним значним плюсом є те, що оскільки компоненти не зв’язані і взаємодіють між собою за допомогою API, можна використовувати саме ті технології, які найбільше підходять для певної задачі. Хоч мікросервісна архітектура має значні переваги, які було наведено вище, для цього проекту вона не дуже підходить по таким причинам: час розробки та складність. Якщо над веб-сайтом працює команда, то мікросервіси будуть чудовим вибором, але при недостачі часу та людських ресурсів краще обрати інший підхід – монолітна архітектура. При моноліті все розміщується на одному сервері: і база даних, і бізнес-логіка, і генерація інтерфейсу користувача. Звісно, це має певні мінуси. Так, наприклад, при помилці в якомусь компоненті весь веб-сервер може припинити роботу. Проте, саме для цього проекту вибір монолітної архітектури буде більш доцільним, адже цей підхід дозволить швидко розробляти та масштабувати веб-сайт.

Далі буде важливим визначитись з нижчим, але не менш важливим рівнем абстракції. На цьому рівні буде застосовуватись патерн, який є безальтернативним у веб-розробці,- MVC (Model – View – Controller).

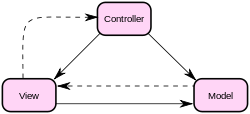


Рис. 2.1. Діаграма взаємодії між компонентами патерну

Для кращого розуміння шаблону, розберімо три його основні компоненти:

* Model: відображає поведінку застосунку, яка не залежить від інтерфейсу користувача. Відповідає за бізнес-логіку веб-сайту.
* View: будь-яке представлення інформації для користувача, частіше за все користувацький інтерфейс у вигляді кнопок, текстових полів та іншого.
* Controller: компонент, який зв’язує модель та представлення. Якщо говорити в загальному, то контролер реагує на запити користувача та перетворює їх на команди для моделі.

Цей шаблон дозволить розподілити веб-сайт на компоненти, що зменшить його зв’язність та зробить весь проект більш структурованим.

Тепер буде необхідним перейти до проектування структури БД, а отже і основної частини Моделі з патерну MVC.

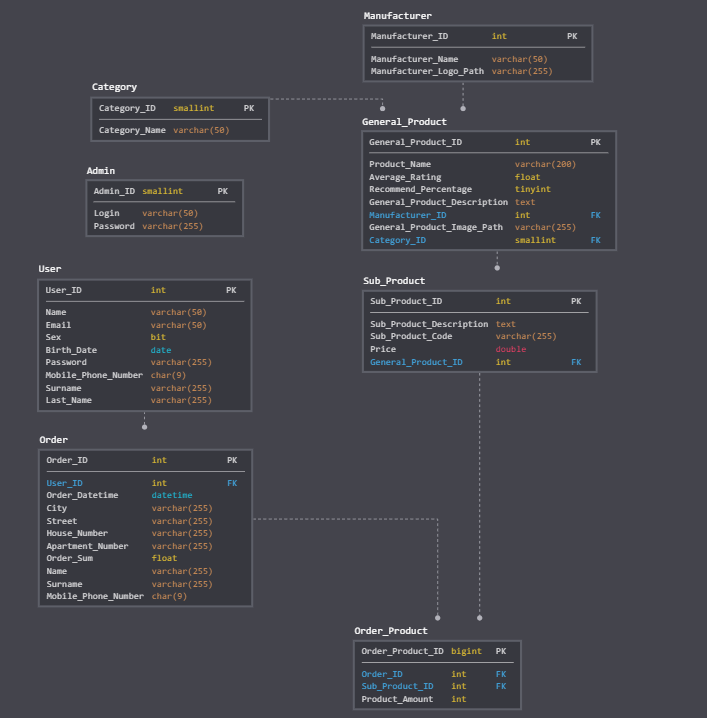


Рис. 2.2. Структура БД проекту

## Розробка функціональних алгоритмів роботи програми

Маючи структуру БД майбутнього веб-сайту, можна приступити до створення необхідних сутностей в рамках ORM для Node.JS Sequelize. Було прийнято рішення розпочати розробку проекту з системи керування вмістом, а отже потрібно створити сутність адміністратора в базі даних.

Лістинг admin.js:

*const* bcrypt = require('bcrypt-nodejs');  
  
module.exports = (sequelize, DataTypes) => {  
 *let* Admin = sequelize.define('Admin', {  
 Admin\_ID: {  
 type: DataTypes.INTEGER,  
 primaryKey: *true*,  
 autoIncrement: *true* },  
 Login: DataTypes.STRING,  
 Password: DataTypes.STRING  
 });  
 Admin.*beforeCreate*((admin, options) => {  
 admin.Password = *this*.generatePassword(admin.Password);  
 });  
 Admin.generatePassword = *function*(password) {  
 *return* bcrypt.hashSync(password, bcrypt.genSaltSync(5), *null*, *function* (err, hash) {  
 console.log('Password creation process was done')  
 });  
 };  
 Admin.prototype.validatePassword = *function*(password) {  
 *return* bcrypt.compareSync(password, *this*.Password);  
 };  
 *return* Admin;  
};

Блок-схема сутності Admin:

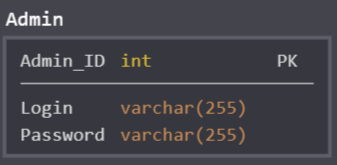


Рис. 2.3. Блок-схема сутності Admin

Ця сутність має всього три поля, одним з яких є ідентифікатор, необхідний у кожній таблиці в будь-якій базі даних. Також у базі даних зберігатиметься логін адміністратора та пароль. Але ж зберігати пароль у вигляді простого тексту є дуже поганою та не безпечною практикою, тому для генерації хешу паролю використовуватиметься пакунок bcrypt, який буде встановлено окремо для проекту. Він дозволить генерувати хеш перед створенням нової сутності Admin за допомогою методу generatePassword та робити перевірку паролю при спробі входу методом validatePassword. Ніяких зв’язків з іншими таблицями таблиця Admin не має.

Користувач для системи керування вмістом був створений, а тому потрібно надати змогу адміністратору створювати необхідні для відображення товарів записи. У товару може бути багато характеристик, але ключовими є категорія та виробник, тому пропоную перейти до них.

Лістинг category.js:

module.exports = (sequelize, DataTypes) => {  
 *let* category = sequelize.define('Category', {  
 Category\_ID: {  
 type: DataTypes.INTEGER,  
 primaryKey: *true*,  
 autoIncrement: *true* },  
 Category\_Name: DataTypes.STRING  
 });  
 *return* category;  
};

Блок-схема сутності Category:

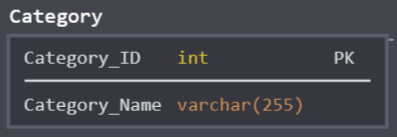


Рис. 2.4. Блок-схема сутності Category

Все доволі просто: сутність Category містить інформації лише про назву категорії та власне ідентифікатор категорії.

Лістинг manufacturer.js:

module.exports = (sequelize, DataTypes) => {  
 *let* manufacturer = sequelize.define('Manufacturer', {  
 Manufacturer\_ID: {  
 type: DataTypes.INTEGER,  
 primaryKey: *true*,  
 autoIncrement: *true* },  
 Manufacturer\_Name: DataTypes.STRING,  
 Manufacturer\_Logo\_Path: DataTypes.STRING  
 });  
 *return* manufacturer;  
};

Блок-схема сутності Manufacturer:

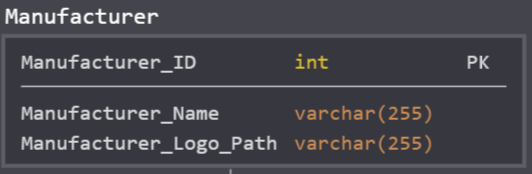


Рис. 2.5. Блок-схема сутності Manufacturer

Таблиця Manufacturer має, окрім, звісно, назви виробника, на одне поле більше: логотип виробника, для збереження якого використовується поле типу String, тому що зберігатиметься саме URL його лого.

Наступна таблиця найважливіша – таблиця загального продукту. Саме загального, тому що було передбачено, що один товар може мати більше однієї варіації.

Лістинг generalproduct.js:

module.exports = (sequelize, DataTypes) => {  
 *let* generalProduct = sequelize.define('General\_Product', {  
 General\_Product\_ID: {  
 type: DataTypes.INTEGER,  
 autoIncrement: *true*,  
 primaryKey: *true* },  
 Product\_Name: DataTypes.STRING,  
 Average\_Rating: DataTypes.FLOAT,  
 Recommend\_Percentage: DataTypes.INTEGER,  
 General\_Product\_Description: DataTypes.TEXT,  
 General\_Product\_Image\_Path: DataTypes.STRING,  
 Category\_ID\_FK: {  
 type: DataTypes.INTEGER,  
 references: {  
 model: 'categories',  
 key: 'Category\_ID'  
 }  
 },  
 Manufacturer\_ID\_FK: {  
 type: DataTypes.INTEGER,  
 references: {  
 model: 'manufacturers',  
 key: 'Manufacturer\_ID'  
 }  
 }  
 });  
 *return* generalProduct;  
};

Блок-схема сутності General\_Product:



Рис. 2.6. Блок-схема сутності General\_Product

Загальний продукт зберігатиме назву продукту, його опис та посилання на зображення продукту. Також ця таблиця матиме зовнішні ключі, які відсилаються на сутності, які було переглянуто раніше: Category і Manufacturer.

Кожен загальний продукт матиме, як мінімум, один підпродукт, який міститиме більш конкретну інформацію про загальний продукт.

Лістинг subproduct.js:

module.exports = (sequelize, DataTypes) => {  
 *return* sequelize.define('Sub\_Product', {  
 Sub\_Product\_ID: {  
 type: DataTypes.INTEGER,  
 autoIncrement: *true*,  
 primaryKey: *true* },  
 Sub\_Product\_Description: DataTypes.TEXT,  
 Sub\_Product\_Code: DataTypes.STRING,  
 Price: DataTypes.DOUBLE,  
 General\_Product\_ID\_FK: {  
 type: DataTypes.INTEGER,  
 references: {  
 model: 'General\_Products',  
 key: 'General\_Product\_ID'  
 },  
 field: 'General\_Product\_ID\_FK'  
 }  
 })  
};

Блок-схема сутності Sub\_Product:

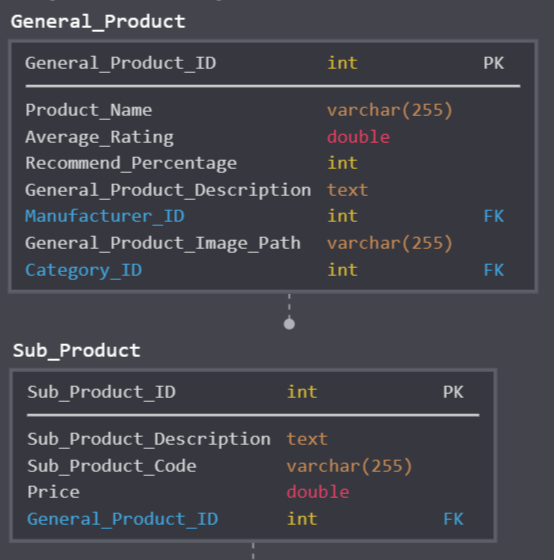


Рис. 2.7. Блок-схема сутності Sub\_Product

Таблиця Sub\_Product теж не є дуже великою, тому що містить опис, код товару та, що найголовніше, ціну. Очевидно, що є присутнім зовнішній ключ, який відсилається на сутність General\_Product.

Це були всі таблиці, пов’язані з системою контролю вмістом, але сам веб-сайт матиме ще й сутності, пов’язані з користувачем. Розпочнімо з User.

Лістинг user.js:

*const* bcrypt = require('bcrypt-nodejs');  
  
module.exports = (sequelize, DataTypes) => {  
 *let* User = sequelize.define('User', {  
 User\_ID: {  
 type: DataTypes.INTEGER,  
 primaryKey: *true*,  
 autoIncrement: *true* },  
 Name: DataTypes.STRING,  
 Last\_Name: DataTypes.STRING,  
 Surname: DataTypes.STRING,  
 Email: DataTypes.STRING,  
 Sex: DataTypes.BOOLEAN,  
 Birth\_Date: DataTypes.DATEONLY,  
 Mobile\_Phone\_Number: DataTypes.STRING(9),  
 Password: DataTypes.STRING  
 });  
 User.*beforeCreate*((user, options) => {  
 user.Password = User.generatePassword(user.Password);  
 });  
 User.generatePassword = *function*(password) {  
 *return* bcrypt.hashSync(password, bcrypt.genSaltSync(5), *null*, *function* (err, hash) {  
 console.log('Password creation process was done')  
 });  
 };  
 User.prototype.validatePassword = *function*(password) {  
 *return* bcrypt.compareSync(password, *this*.Password);  
 };  
 *return* User;  
};

Блок-схема сутності User:



Рис. 2.8. Блок-схема сутності User

Ця сутність містить таку інформацію про користувача: ім’я, прізвище, по-батькові електронна пошта, стать, дата народження та номер мобільного телефону. Для роботи з паролем користувача так само було використано пакунок bcrypt, за допомогою якого відбувається хешування паролю перед створенням користувача та перевірка паролю при вході.

Після створення користувача можна перейти до ініціалізації того, без чого інтернет-магазин не має сенсу: замовлення. Тут ситуація схожа на ту, що була присутня у таблицях з товарами. А саме: замовлення може мати багато товарів. Тому було прийнято рішення розбити одну сутність (у розумінні звичайного користувача) на дві таблиці: Order та Order\_Product. Пропоную перейти до розгляду сутності Order.

Лістинг order.js:

module.exports = (sequelize, DataTypes) => {  
 *return* sequelize.define('Order', {  
 Order\_ID: {  
 type: DataTypes.INTEGER,  
 primaryKey: *true*,  
 autoIncrement: *true* },  
 Order\_Datetime: {  
 type: DataTypes.DATE,  
 defaultValue: DataTypes.NOW  
 },  
 City: DataTypes.STRING,  
 Area: DataTypes.STRING,  
 Street: DataTypes.STRING,  
 House\_Number: DataTypes.STRING,  
 Apartment\_Number: DataTypes.STRING,  
 Order\_Sum: DataTypes.DOUBLE,  
 Name: DataTypes.STRING,  
 Last\_Name: DataTypes.STRING,  
 Mobile\_Phone\_Number: DataTypes.STRING(9),  
 User\_ID\_FK: {  
 type: DataTypes.INTEGER,  
 references: {  
 model: 'users',  
 key: 'User\_ID'  
 }  
 }  
 })  
};

Блок-схема сутності Order:

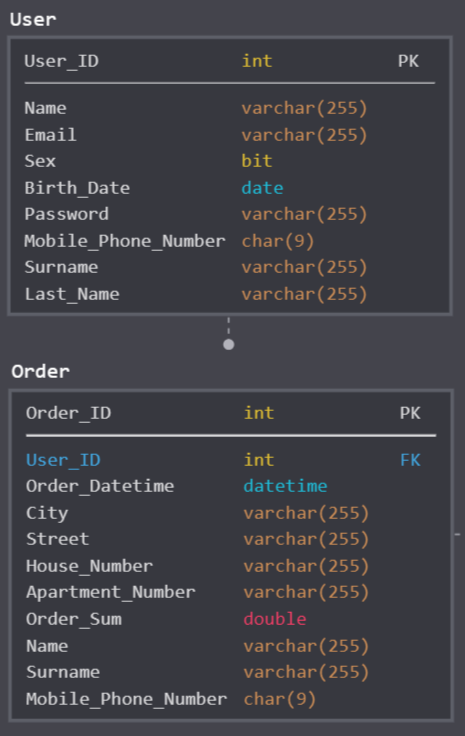


Рис. 2.9. Блок-схема сутності Order

Оскільки замовлення здійснює користувач, то ця таблиця має зовнішній ключ, який з’єднаний з унікальним ідентифікатором користувача з таблиці User. Також ця сутність має наступні поля: час і дата замовлення, місто для доставки, вулиця для доставки, будинок та номер квартири для доставки, сума замовлення, ім’я, прізвище та мобільний телефон отримувача. Для збереження номеру будинку та квартири використовується тип з ORM Sequelize STRING, адже ці значення можуть містити букви і/або дроби.

Сформувавши основні критерії замовлення, можна перейти до створення таблиці, де буде зберігатися інформація про замовлені товари. За це відповідає таблиця Order\_Product.

Лістинг orderproduct.js:

*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* Order = require('../../models/user/order')(Sequelize, sequelize);  
*const* SubProduct = require('../../models/product/subproduct')(Sequelize, sequelize);  
  
module.exports = (sequelize, DataTypes) => {  
 *let* OrderProduct = sequelize.define('Order\_Product', {  
 Order\_Product\_ID: {  
 type: DataTypes.BIGINT,  
 primaryKey: *true*,  
 autoIncrement: *true* },  
 Product\_Amount: DataTypes.INTEGER,  
 Order\_ID\_FK: {  
 type: DataTypes.INTEGER,  
 references: {  
 model: 'orders',  
 key: 'Order\_ID'  
 }  
 },  
 Sub\_Product\_ID\_FK: {  
 type: DataTypes.INTEGER,  
 references: {  
 model: 'sub\_products',  
 key: 'Sub\_Product\_ID'  
 }  
 }  
 });  
 OrderProduct.*beforeCreate*((orderProduct, options) => {  
 Order.findByPk(orderProduct.Order\_ID\_FK).then(order => {  
 *let* orderSum;  
 *if* (!order.Order\_Sum)  
 orderSum = 0;  
 *else* orderSum = order.Order\_Sum;  
 SubProduct.findByPk(orderProduct.Sub\_Product\_ID\_FK).then(subProduct => {  
 orderSum += subProduct.Price \* orderProduct.Product\_Amount;  
 Order.update({Order\_Sum: orderSum}, {where: {Order\_ID: orderProduct.Order\_ID\_FK}}).then(result => {  
 *if* (result)  
 console.log('Update successful');  
 *else* console.log('Update failed');  
 })  
 });  
  
 });  
 });  
 *return* OrderProduct;  
};

Блок-схема сутності Order\_Product:

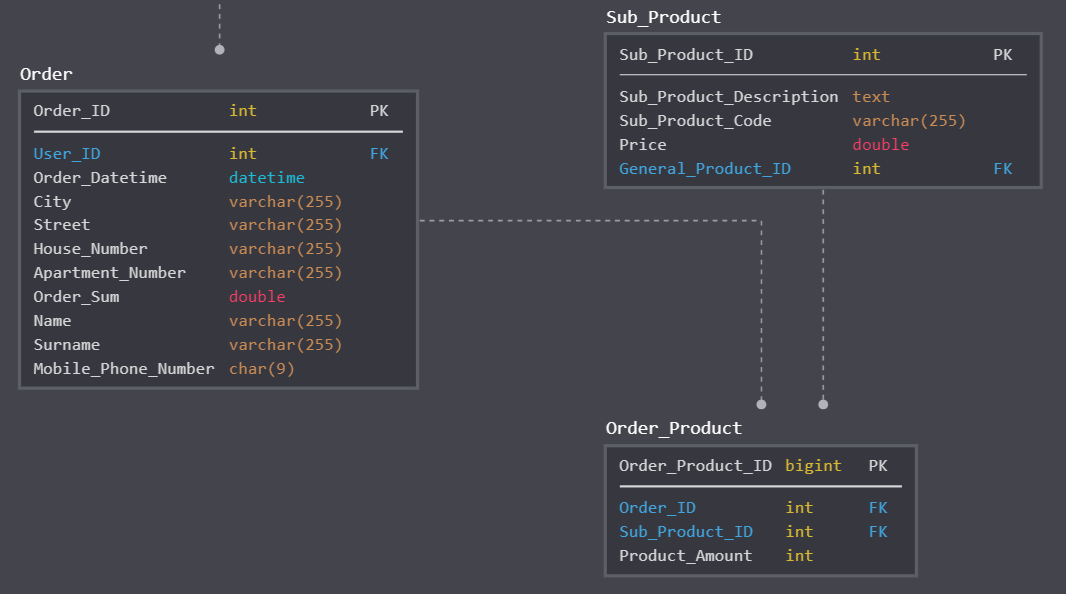


Рис. 2.10. Блок-схема сутності Order\_Product

Ця таблиця досить мала і половина її складу це зовнішні ключі, в цьому випадку на ідентифікатори таблиць Order і Sub\_Product. Зрозуміло, що ця таблиця також має свій ідентифікатор, але він не типу INTEGER, як всі попередні, а типу BIGINT, адже замовлених товарів за весь час існування інтернет-магазину може бути дуже велика кількість. Ще одне поле – це кількість певної одиниці товару. Також до цієї таблиці було додано таку собі процедуру, яка відбувається перед створенням запису у таблиці. А саме: знаходиться запис з таблиці Order за зовнішнім ключем, потім знаходиться відповідний товар з таблиці Sub\_Product, після чого до поля Order\_Sum таблиці Order додається загальна ціна замовленого товару шляхом множення ціни товару на його кількість.

Розібравшись як і з загальною структурою бази даних проекту, так і окремо з її сутностями, можна перейти до огляду файлів, які знаходяться поза патерном MVC та є допоміжними.

Було визначено сутності бази даних, але далі необхідно зробити можливим використовувати їх у самому сервері. За це відповідають файли sequelize.js та databasesequelize.js.

Лістинг sequelize.js:

*const* sequelize = require('sequelize');  
  
*const* Sequelize = *new* sequelize('sadivnyk\_database', 'Vladyslav', 'mypassword123', {  
 dialect: 'mysql',  
 host: 'localhost'  
});  
  
module.exports = {  
 sequelize, Sequelize  
};

Лістинг databasesequelize.js:

*const* {sequelize, Sequelize} = require('./sequelize');  
  
*// models  
const* categoryFile = require('./product/category');  
*const* manufacturerFile = require('./product/manufacturer');  
*const* generalProductFile = require('./product/generalproduct');  
*const* subProductFile = require('./product/subproduct');  
*const* adminFile = require('./admin/admin');  
*const* userFile = require('./user/user');  
*const* orderFile = require('./user/order');  
*const* orderProductFile = require('./user/orderproduct');  
  
*// models instances  
const* category = categoryFile(Sequelize, sequelize);  
*const* manufacturer = manufacturerFile(Sequelize, sequelize);  
*const* generalProduct = generalProductFile(Sequelize, sequelize);  
*const* subProduct = subProductFile(Sequelize, sequelize);  
*const* admin = adminFile(Sequelize, sequelize);  
*const* user = userFile(Sequelize, sequelize);  
*const* order = orderFile(Sequelize, sequelize);  
*const* orderProduct = orderProductFile(Sequelize, sequelize);  
  
*// relationships*manufacturer.hasMany(generalProduct, {foreignKey: 'Manufacturer\_ID\_FK', onDelete: 'set null'});  
generalProduct.*belongsTo*(manufacturer, {foreignKey: 'Manufacturer\_ID\_FK', as: 'Manufacturers'});  
category.*hasMany*(generalProduct, {foreignKey: 'Category\_ID\_FK', onDelete: 'set null'});  
generalProduct.*belongsTo*(category, {foreignKey: 'Category\_ID\_FK', as: 'Categories'});  
generalProduct.*hasMany*(subProduct, {foreignKey: 'General\_Product\_ID\_FK', sourceKey: 'General\_Product\_ID', onDelete: 'cascade'});  
subProduct.belongsTo(generalProduct, {foreignKey: 'General\_Product\_ID\_FK', targetKey: 'General\_Product\_ID'});  
order.hasMany(orderProduct, {foreignKey: 'Order\_ID\_FK', sourceKey: 'Order\_ID', onDelete: 'cascade'});  
orderProduct.*belongsTo*(order, {foreignKey: 'Order\_ID\_FK', targetKey: 'Order\_ID'});  
subProduct.hasMany(orderProduct, {foreignKey: 'Sub\_Product\_ID\_FK', sourceKey: 'Sub\_Product\_ID', onDelete: 'set null'});  
orderProduct.*belongsTo*(subProduct, {foreignKey: 'Sub\_Product\_ID\_FK', targetKey: 'Sub\_Product\_ID'});  
Sequelize.sync().then(() => {  
 console.log('Database and tables created!')  
});

У файлі sequelize.js відбувається підключення пакунку sequelize, тобто ORM, яка є досить популярною серед розробників на платформі Node.JS. Цей файл викликатиметься у всіх місцях, де буде необхідним працювати з сутностями бази даних. В свою чергу файл databasesequelize підключається у головному файлі всього серверу app.js при його запуску, що дозволяє відразу створити таблиці, якщо вони відсутні, та створити необхідні зв’язки між ними.

Останнім допоміжним файлом є passport.js, який налаштовує пакунок passport, головна задача якого спростити розробнику вхід користувачів у свої акаунти.

Лістинг passport.js:

*function* UserInfo(userId, userType) {  
 *this*.userId = userId;  
 *this*.userType = userType;  
}  
  
*const* passport = require('passport');  
*const* LocalStrategy = require('passport-local').Strategy;  
  
*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* Admin = require('../models/admin/admin')(Sequelize, sequelize);  
*const* User = require('../models/user/user')(Sequelize, sequelize);  
  
passport.serializeUser(*function*(someUser, done) {  
 *let* userType = 'user';  
 *let* userId;  
 *let* userPrototype = Object.getPrototypeOf(someUser);  
 *if* (userPrototype === Admin.prototype) {  
 userType = 'admin';  
 userId = someUser.Admin\_ID;  
 }  
 *else if* (userPrototype === User.prototype) {  
 userType = 'user';  
 userId = someUser.User\_ID;  
 }  
 *let* userInfo = *new* UserInfo(userId, userType);  
 done(*null*, userInfo);  
});  
  
passport.deserializeUser(*function* (UserInfo, done) {  
 *if* (UserInfo.userType === 'admin') {  
 Admin.*findByPk*(UserInfo.userId).then(admin => {  
 *if* (admin)  
 done(*null*, admin);  
 })  
 }  
 *else if* (UserInfo.userType === 'user') {  
 User.*findByPk*(UserInfo.userId).then(user => {  
 *if* (user)  
 done(*null*, user);  
 })  
 }  
});  
  
passport.use('local.admin', *new* LocalStrategy({  
 usernameField: 'Login',  
 passwordField: 'Password',  
 passReqToCallback: *true*}, *function*(req, Login, Password, done) {  
 Admin.*findOne*({where: {Login: Login}}).then(admin => {  
 *if* (!admin)  
 *return* done(*null*, *false*, {message: 'No such login'});  
 *if* (!admin.validatePassword(Password)) {  
 console.log('Password doesn\'t match');  
 *return* done(*null*, *false*, {message: 'Password doesn\'t match'});  
 }  
 *else  
 return* done(*null*, admin);  
 })  
}));  
  
passport.use('local.user', *new* LocalStrategy({  
 usernameField: 'email',  
 passwordField: 'password',  
 passReqToCallback: *true*}, *function* (req, email, password, done) {  
 User.*findOne*({where: {Email: email}}).then(user => {  
 *if* (!user)  
 *return* done(*null*, *false*, {message: 'No such email'});  
 *if* (!user.validatePassword(password)) {  
 console.log('User password does not match');  
 *return* done(*null*, *false*, {message: 'Password doesn\'t match'});  
 }  
 *else  
 return* done(*null*, user);  
 });  
}));

Passport використовує аутентифікаційний механізм, який розробник цього пакунку називає стратегією. Їх насправді є сотні і можна знайти стратегії, де відбувається аутентифікація за допомогою, мабуть, усіх можливих сайтів або соціальних мереж. Але для цього проекту буде використовуватись власні стратегії, яку можливо створити за допомогою пакунку від цього ж розробника – passport-local. Далі процес створення стратегії стає легким, досить лише вказати назву відповідних полів з атрибуту name форми, яка надсилатиметься з сайту на сервер, після чого у callback можна валідувати передані дані та надсилати відповідь на запит. Але це не все, що потрібно пакунку passport, адже ще необхідно створити методи для serializeUser та deserializeUser, що, як можна зрозуміти з назви, серіалізують та десеріалізують користувачів. Оскільки у веб-сайті використовуються два типи користувачів, було створено допоміжний клас UserInfo, у якому всього два поля: userId та userType і саме за допомогою останнього і визначається тип користувача.

## Розробка програмного забезпечення

В попередньому підпункті було детально розібрано Model - частину з патерну MVC, у цій частині звіту будуть розібрані інші складові цього шаблону – View та Controller. Пропоную розпочати з View.

Задля того, щоб спростити процес відображення та генерації сторінок, у фреймворці Express, який використовують у зв’язці з Node.JS, є така річ, як view engine або, як іноді його називають, template engine. Його перевагою є те, що ми можемо визначати деякі шаблони у файлах представлень, які потім наповнюватимуться необхідними даними. Взагалі view engine для Express є велика кількість, але для цього проекту було обрано Handlebars, тому що він пропонує широкий функціонал, зрозумілий синтаксис та є найпопулярнішим серед Node.JS + Express розробників.

Ще однією важливою можливістю у Handlebars є те, що ми можемо визначати загальні шаблони, які використовуватимуться для всіх сторінок. І через те, що цей веб-проект має користувацьку частину та систему керування вмістом, буде визначено 2 загальних шаблони сторінок окремо для кожної частини. Доречним буде розглянути спершу, як і до цього, шаблон для панелі адміністратора.

Лістинг adminlayout.hbs:

<!DOCTYPE *html*>  
<html>  
<head>  
 <title>{{*title*}}</title>  
 <meta *charset*="utf-8">  
 <meta *name*="viewport" *content*="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link *rel*='stylesheet' *href*='/stylesheets/adminstyle.css' />  
 <link *rel*="stylesheet" *href*="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" *integrity*="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" *crossorigin*="anonymous">  
 <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">  
</head>  
<body>  
{{{*body*}}}  
<script *src*="https://code.jquery.com/jquery-3.5.1.slim.min.js" *integrity*="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" *crossorigin*="anonymous"></script><script *src*="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" *integrity*="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI" *crossorigin*="anonymous"></script>  
{{#*each scripts*}}  
 <script *src*="{{*script*}}"></script>  
{{/*each*}}  
</body>  
</html>

У цьому файлі з розширенням .hbs підключаються всі необхідні скрипти та стилі, а динамічним контентом є заголовок сторінки, тіло – тобто те, що і буде відображатись адміністратору та скрипти, назви яких будуть передані у масиві.

Лістинг layout.hbs:

<!DOCTYPE *html*>  
<html>  
 <head>  
 <title>{{*title*}}</title>  
 <meta *charset*="utf-8">  
 <meta *name*="viewport" *content*="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link *rel*='stylesheet' *href*='/stylesheets/style.css' />  
 <link *rel*="stylesheet" *href*="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" *integrity*="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk" *crossorigin*="anonymous">  
 <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">  
 <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/ion-rangeslider/2.3.1/css/ion.rangeSlider.min.css"/>  
 <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/malihu-custom-scrollbar-plugin/3.1.5/jquery.mCustomScrollbar.min.css">  
 <link *rel*="stylesheet" *href*="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/css/bootstrap-datepicker.min.css" />  
 </head>  
 <body>  
 {{> *header this*}}  
 {{{*body*}}}  
 {{> *footer*}}  
 <script *src*="https://code.jquery.com/jquery-3.5.1.slim.min.js" *integrity*="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" *crossorigin*="anonymous"></script>  
 <script *src*="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" *integrity*="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" *crossorigin*="anonymous"></script>  
 <script *src*="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" *integrity*="sha384-OgVRvuATP1z7JjHLkuOU7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JKI" *crossorigin*="anonymous"></script>  
 <script *src*="https://cdnjs.cloudflare.com/ajax/libs/malihu-custom-scrollbar-plugin/3.1.5/jquery.mCustomScrollbar.concat.min.js"></script>  
 <script *src*="https://cdnjs.cloudflare.com/ajax/libs/ion-rangeslider/2.3.1/js/ion.rangeSlider.min.js"></script>  
 <script *src*="https://cdnjs.cloudflare.com/ajax/libs/bootstrap-datepicker/1.8.0/js/bootstrap-datepicker.min.js"></script>  
 <script *src*="/javascripts/helpers/alertContainer.js"></script>  
 <script *src*="https://cdnjs.cloudflare.com/ajax/libs/jquery.mask/1.14.16/jquery.mask.min.js"></script>  
 {{#*each scripts*}}  
 <script *src*="{{*script*}}"></script>  
 {{/*each*}}  
 <script *src*="/javascripts/signupjs.js"></script>  
 </body>  
</html>

Шаблон для самого інтернет-магазину має схожу структуру з попереднім шаблоном: так само динамічно відображається заголовок, тіло та скрипти, окрім однієї деталі. У цьому темплейті вже застосовуються часткові представлення (англійською partials), а саме для хедера та футера, які на всіх сторінках будуть однакові.

Далі буде доцільним подивитися на роботу view engine не в шаблонах, а на прикладі самих сторінок.

Так, наприклад, у adminpanel.hbs (додаток А) було використано всю силу двигуна представлень і з лістингу, який знаходиться у додатках, можна побачити, як використовуються цикли, вкладені цикли, умови та навіть власні допоміжні функції для Handlebars. Проте, цей view engine не дозволяє використовувати динамічні представлення у самому представлені, тому у контролері, який буде розглянутий пізніше, довелося власноруч зчитувати необхідні дані з .hbs файлів та вставляти у представлення adminpanel.hbs, якщо це потрібно.

Після розгляду сторінки для адмін-панелі було б добре подивитися на генерацію сторінки для інтернет-магазину і якнайкраще це можна зробити у файлі product.hbs.

Лістинг product.hbs:

<div *class*="main-content d-flex justify-content-center">  
 <div *class*="col-lg-6 col-11 shadow p-lg-5 p-md-3 p-0 py-3">  
 <div *class*="px-1 mb-3">  
 <button *class*="default-link link-line-height" *onclick*="window.history.back()">  
 <i *class*="fa fa-arrow-left fa-2x"></i>  
 <div *class*="icon-span-text">Назад</div>  
 </button>  
 </div>  
 <h1 *class*="header-font px-1">{{*product*.*Product\_Name*}}</h1>  
 <div *class*="text-center">  
 <img *src*="{{*product*.*General\_Product\_Image\_Path*}}" *alt*="Зображення товару" *class*="my-4 product-image">  
 </div>  
 <div *class*="pt-3 px-3 main-font">  
 Виробник: {{*manufacturer*.*Manufacturer\_Name*}}  
 <img *src*="{{*manufacturer*.*Manufacturer\_Logo\_Path*}}" *alt*="Лого виробника" *class*="ml-3" *height*="48px">  
 </div>  
 <hr/>  
 <div *class*="px-3 main-font">  
 Опис: {{*product*.*General\_Product\_Description*}}  
 </div>  
 <hr/>  
 {{> *alertcontainer*}}  
 <div *class*="mt-3 p-1 main-font">  
 <table *class*="sub-products-table">  
 <thead>  
 <tr>  
 <td>Артикул</td>  
 <td>Опис/Розмір</td>  
 <td>Ціна</td>  
 <td>Купити</td>  
 </tr>  
 </thead>  
 {{#*each subProducts*}}  
 <tr>  
 <td>{{*this*.*Sub\_Product\_Code*}}</td>  
 <td>{{*this*.*Sub\_Product\_Description*}}</td>  
 <td>{{*this*.*Price*}}</td>  
 <td><button *role*="button" *onclick*="addToCart('/addtocart/{{*this*.*Sub\_Product\_ID*}}')" *class*="btn main-button main-font">Купити</button></td>  
 </tr>  
 {{/*each*}}  
 </table>  
 </div>  
 </div>  
</div>

Взагалі тут використовується та сама логіка, що і у попередньому представлені: передаються певні об’єкти і колекції, властивості яких можна використовувати для відображення необхідної інформації, але при розробці було помічено, що ці властивості можна навіть передавати у функції зі скриптів, які завантажуються разом зі сторінкою. Інші представлення немає сенсу переглядати, адже вони всі використовують ту ж логіку, яка була описана у цих двох прикладах.

Нарешті можна перейти до останнього і, мабуть, найважливішого компоненту MVC – контролери. Перед тим, як перейти до розбору самих контролерів, хотілось би розказати про роутинг запитів. Деякі запити мають одну і ту ж початкову частину в URL і тому, щоб не прописувати її для кожного запиту з цією частиною, у фреймворці Express є можливість створювати роутери. Все просто: вказуєш спільну частину у методі app.use та передаєш створений роутер. І ще чудово те, що в роутерах теж можна вкладати роутери, що дозволить зменшити розмір файлів та збільшити читабельність коду. Найкращим прикладом цього є файл adminrouter.js.

Лістинг adminrouter.js:

*const* express = require('express');  
*const* passport = require('passport');  
*const* csrf = require('csurf');  
  
*var* csrfProtection = *new* csrf();  
  
*//controllers  
const* adminSignInController = require('../../controllers/admin/adminsignincontroller.js');  
*const* adminPanelController = require('../../controllers/admin/adminpanelcontroller');  
  
*//routers  
const* resourcesRouter = require('./resourcesrouter');  
*const* createRouter = require('./createRouter');  
*const* deleteRouter = require('./deleteRouter');  
*const* editRouter = require('./editrouter');  
  
*const* databaseSequelize = require('../../models/databasesequelize');  
  
*const* adminRouter = express.Router();  
adminRouter.use(csrfProtection);  
adminRouter.post('/signin', passport.authenticate('local.admin', {  
 successRedirect: '/admin/panel',  
 failureRedirect: '/admin'  
}, )); *// Right order of paths*adminRouter.get('/panel/\*', isAdminLogged);  
adminRouter.get('/panel/logout', adminPanelController.logoutAdmin);  
adminRouter.use('/panel/create', createRouter);  
adminRouter.use('/panel/resources', resourcesRouter);  
adminRouter.use('/panel/delete', deleteRouter);  
adminRouter.use('/panel/edit', editRouter);  
adminRouter.get('/panel', isAdminLogged, adminPanelController.returnAdminPanel);  
adminRouter.get('/', adminSignInController.returnAdminSignInPage); *function* isAdminLogged(request, response, next) {  
 *if* (request.user && 'Admin\_ID' *in* request.user)  
 next();  
 *else* response.sendStatus(403);  
}  
  
module.exports = adminRouter;

Цей роутер використовує необхідні йому контролери, інші роутери, які будуть використовуватись для певних шляхів та деякі сторонні пакунки. Також хочу відмітити, що тут було використано middleware для перевірки того, чи адміністратор є залогіненим, в іншому випадку йому не надається доступ до системи керування вмістом. Роутери було використано і для інших частин проекту, але їх склад схожий або простіший за приклад, наведений вище, тому не бачу сенсу їх переглядати.

Нарешті можна перейти до розгляду самих контролерів. Як завжди, будуть наведені приклад як і з системи керування вмістом, так і з користувацької частини.

Головним контролером для системи керування вмістом є контролер з файлу adminpanelcontroller.js (додаток Б). На жаль, не вийшло його розподілити на менші контролери, адже логіка всіх методів є дуже схожою і використовує одні і ті ж алгоритми та сутності для цих алгоритмів. Якщо пояснити суть самого контролеру, то вона полягає в тому, щоб створювати необхідне відображення для системи керування вмісту, а саме повертати необхідні записи з таблиць та форми для редагування існуючих та створення нових записів.

Ще одним контролером з системи керування вмістом, який розглядатиметься, буде контролер з admindeletecontroller.js.

Лістинг admindeletecontroller.js:

*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* Admin = require('../../models/admin/admin')(Sequelize, sequelize);  
*const* Category = require('../../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../../models/product/subproduct')(Sequelize, sequelize);  
  
exports.deleteAdmin = *function* (request, response, adminId) {  
 Admin.destroy({where: {Admin\_ID: adminId}}). then(result => {  
 *if* (result)  
 response.send('Record deleted');  
 *else* response.send('Deletion failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Deletion failed');  
 })  
};  
  
exports.deleteCategory = *function* (request, response, categoryId) {  
 Category.destroy({where: {Category\_ID: categoryId}}). then(result => {  
 *if* (result)  
 response.send('Record deleted');  
 *else* response.send('Deletion failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Deletion failed');  
 })  
};  
  
exports.deleteManufacturer = *function* (request, response, manufacturerId) {  
 Manufacturer.destroy({where: {Manufacturer\_ID: manufacturerId}}). then(result => {  
 *if* (result)  
 response.send('Record deleted');  
 *else* response.send('Deletion failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Deletion failed');  
 })  
};  
  
exports.deleteGeneralProduct = *function*(request, response, generalProductId) {  
 GeneralProduct.destroy({where: {General\_Product\_ID: generalProductId}}).then(result => {  
 *if* (result)  
 response.send('Record deleted');  
 *else* response.send('Deletion failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Deletion failed');  
 })  
};  
  
exports.deleteSubProduct = *function*(request, response, subProductId) {  
 SubProduct.destroy({where: {Sub\_Product\_ID: subProductId}}).then(result => {  
 *if* (result)  
 response.send('Record deleted');  
 *else* response.send('Deletion failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Deletion failed');  
 })  
};

Цей контролер було обрано для демонстрації, тому що він вже не впливає напряму на відображення сторінки, а відповідає на запит, який надсилається з фронт-енду за допомогою fetch та javascript. У цьому випадку метод контролеру просто надсилає текст про результат видалення запису у відповідь.

Далі будуть розглянуті контролери користувацього інтерфейсу. І перший з них контролер з файлу productcontroller.js, який відповідає за відображення сторінки товару.

Лістинг productcontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
  
*const* Manufacturer = require('../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
  
  
exports.returnProductPage = *function* (request, response, productId) {  
 *let* product;  
 *let* subProducts;  
 *let* productManufacturer;  
 GeneralProduct.findByPk(productId).then(generalProduct => {  
 product = generalProduct;  
 Manufacturer.findByPk(generalProduct.Manufacturer\_ID\_FK).then(manufacturer => {  
 productManufacturer = manufacturer;  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: product.General\_Product\_ID}}).then(subProductsFromDatabase => {  
 subProducts = subProductsFromDatabase;  
 response.render('product', {  
 title: 'Садівник - ' + product.Product\_Name,  
 product: product,  
 manufacturer: productManufacturer,  
 subProducts: subProducts,  
 csrfToken: request.csrfToken(),  
 scripts: [{script: '/javascripts/productjs.js'}]  
 })  
 })  
 })  
  
 })  
};

Цей контролер має лише один метод і в ньому він спочатку знаходить запис з таблиці General\_Product за ідентифікатором, який передається у цей метод, після чого знаходить виробника товару та записи з таблиці Sub\_Product, які мають у зовнішньому ключі ідентифікатор продукту, знайденого раніше. Всі запити до бази даних відбуваються поступово, один за одним за допомогою callback. Тільки після виконання всіх запитів генерується сторінка, яка надсилається користувачу.

Останнім контролером для аналізу буде той, який відповідає за створення замовлення.

Лістинг ordercontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
*const* Order = require('../models/user/order')(Sequelize, sequelize);  
*const* OrderProduct = require('../models/user/orderproduct')(Sequelize, sequelize);

*var* Cart = require('../models/helpers/cart');

exports.returnOrderPage = *function* (request, response) {  
 *if* (!request.session.cart)  
 *return* response.redirect('/cart');  
 *let* cart = *new* Cart(request.session.cart);  
 response.render('order', {  
 title: 'Садівник - Оформлення Замовлення',  
 csrfToken: request.csrfToken,  
 user: request.user,  
 products: cart.generateArray(),  
 totalPrice: cart.totalPrice,  
 scripts: [{script: '/javascripts/orderjs.js'}]  
 })  
};

exports.tryToRedirectToOrderPage = *function* (request, response) {  
 *if* (request.user) {  
 *let* isUser = 'User\_ID' *in* request.user;  
 *if* (request.isAuthenticated() && isUser) {  
 response.send('Redirect');  
 }  
 *else* response.send('Not logged');  
 }  
 *else* {  
 request.session.orderUrl = '/order';  
 request.session.save();  
 response.send('Not logged');  
 }  
};  
exports.placeOrder = *function* (request, response) {  
 *if* (!request.session.cart)  
 *return* response.send('Redirect cart');  
 *if* (!request.user) {  
 *return* response.send('Log in');  
 }  
 *if* (request.user) {  
 *let* isUser = 'User\_ID' *in* request.user;  
 *if* (request.isAuthenticated() && !isUser) {  
 *return* response.send('Admin');  
 }  
 }  
 *let* cart = *new* Cart(request.session.cart);  
 Order.create({Name: request.body.name, Last\_Name: request.body.lastName, City: request.body.city, Area: request.body.area, Street: request.body.street, House\_Number: request.body.building, Apartment\_Number: request.body.apartment, Mobile\_Phone\_Number: request.body.phone, User\_ID\_FK: request.user.User\_ID }).then(result => {  
 *if* (result) {  
 *let* count = 0;  
 *for* (*let* item *in* cart.items) {  
 OrderProduct.create({Product\_Amount: cart.items[item].qty, Sub\_Product\_ID\_FK: cart.items[item].item.Sub\_Product\_ID, Order\_ID\_FK: result.Order\_ID}).then(result => {  
 *if* (result) {  
 count++;  
 *if* (count === Object.keys(cart.items).length) {  
 request.session.cart = {};  
 request.session.save();  
 response.send('Order placed');  
 }  
 }  
 });  
 }  
 }  
 *else* response.send('Creation failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Creation failed');  
 })  
};

Цей контролер поєднує в собі методи, які відповідають за рендеринг сторінок з відповідною інформацією, та методи, які надсилають у відповідь результат перевірки певного стану. Так, перший метод з лістингу просто рендерить сторінку замовлення, в той час як другий метод перевіряє, чи користувач залогінений і надсилає необхідну відповідь. Останній та найбільш громіздкий метод створює записи у таблицях Order і Order\_Product і надсилає назад повідомлення про успішність роботи з базою даних.

# ОПИС РОБОТИ З ПРОГРАМНИМ ДОДАТКОМ ТА ЙОГО ТЕСТУВАННЯ

## Опис роботи з програмним додатком

Розпочнімо тестування з CMS інтернет-магазину. Для доступу до неї потрібно перейти за адресою /admin:

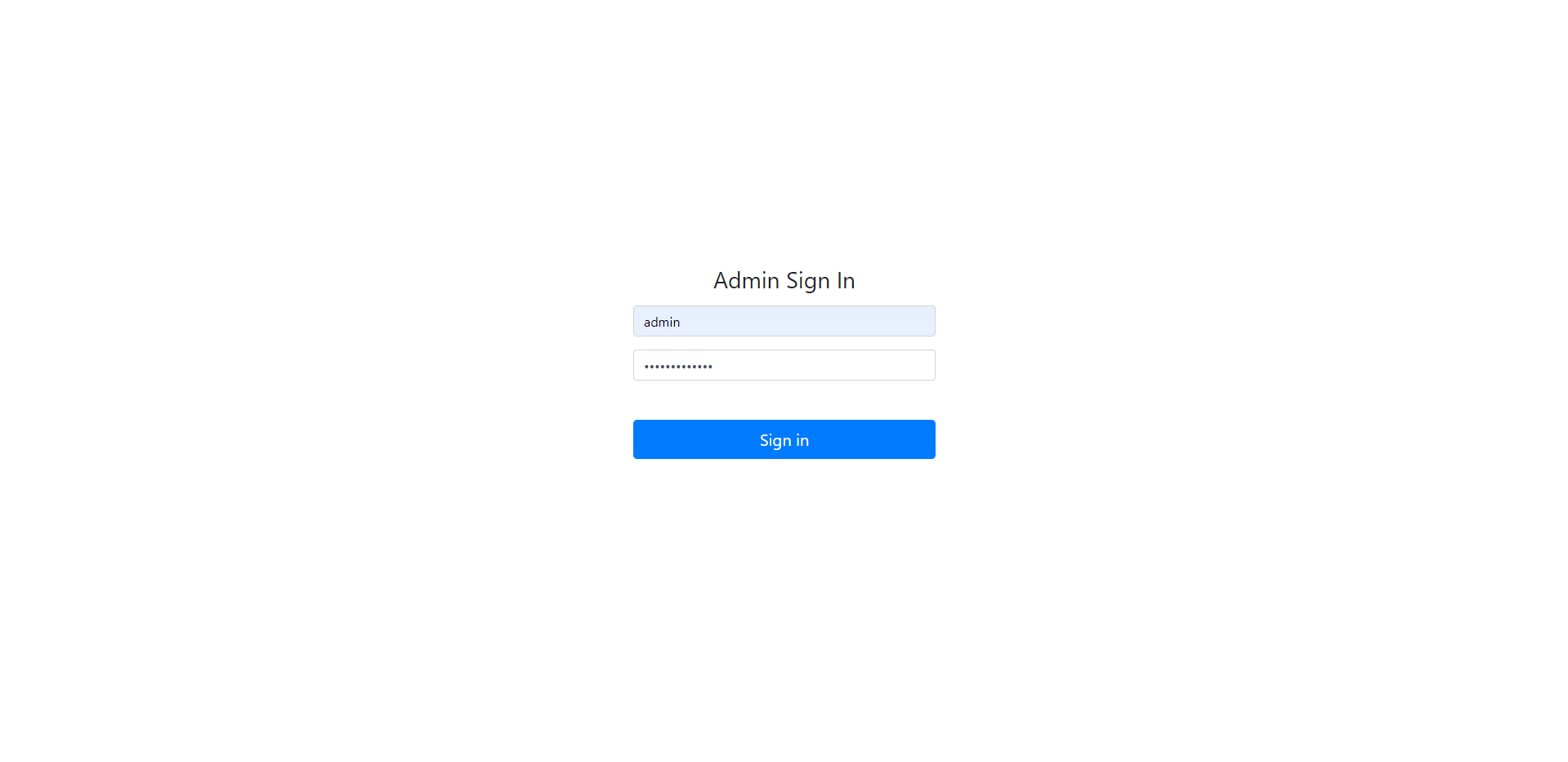


Рис. 3.1. Вхід в адмін-панель

Після вдалого входу відобразиться пуста адмін-панель зі списком сутностей, з якими можна проводити різні операції

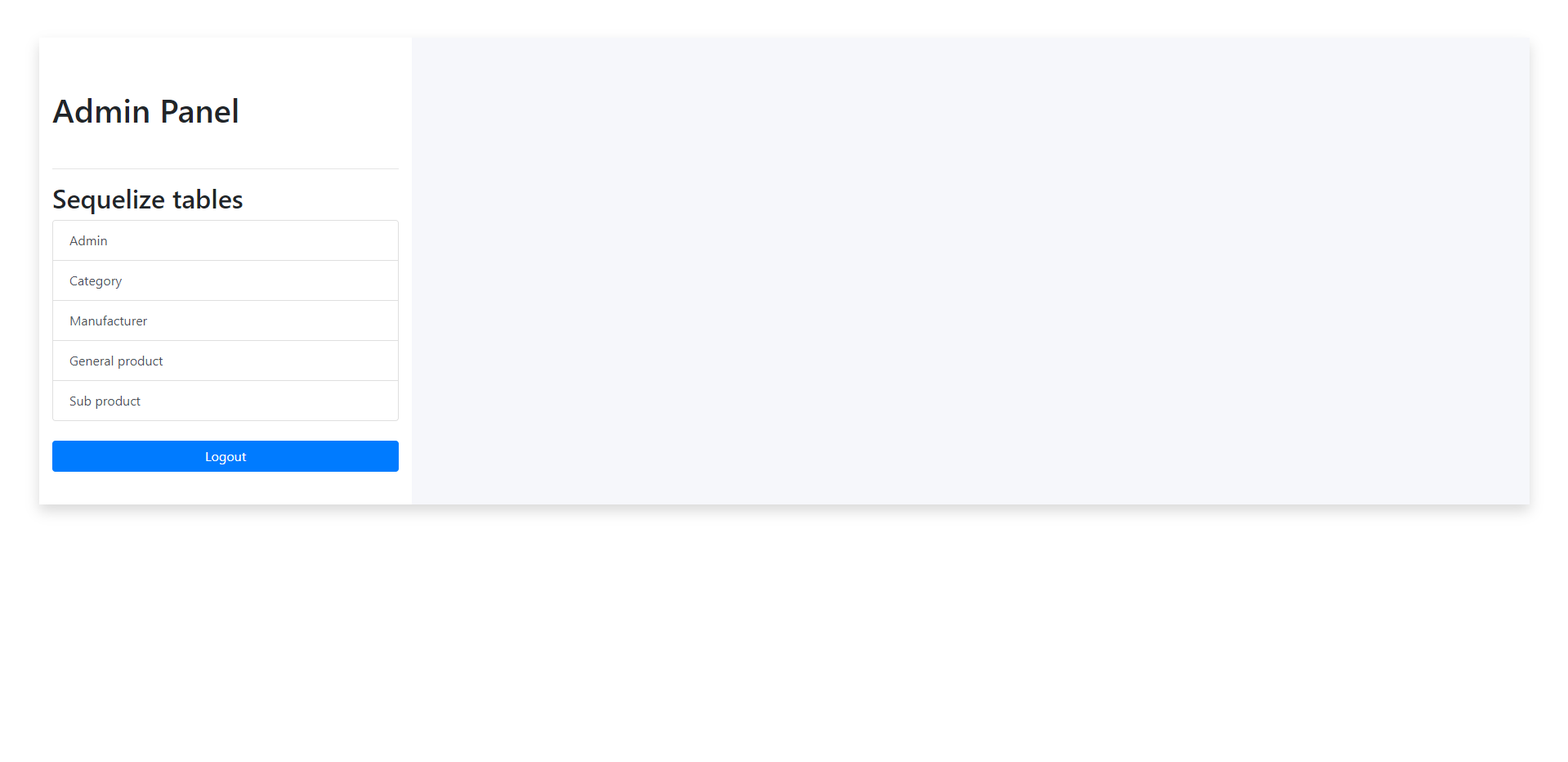


Рис. 3.2. Початковий вигляд адмін-панелі

Для того, щоб почати працювати з певною таблицею з бд, потрібно обрати її у списку зліва, після чого відобразяться перші 10 записів цієї таблиці, знизу будуть доступні посилання на сторінки з записами.

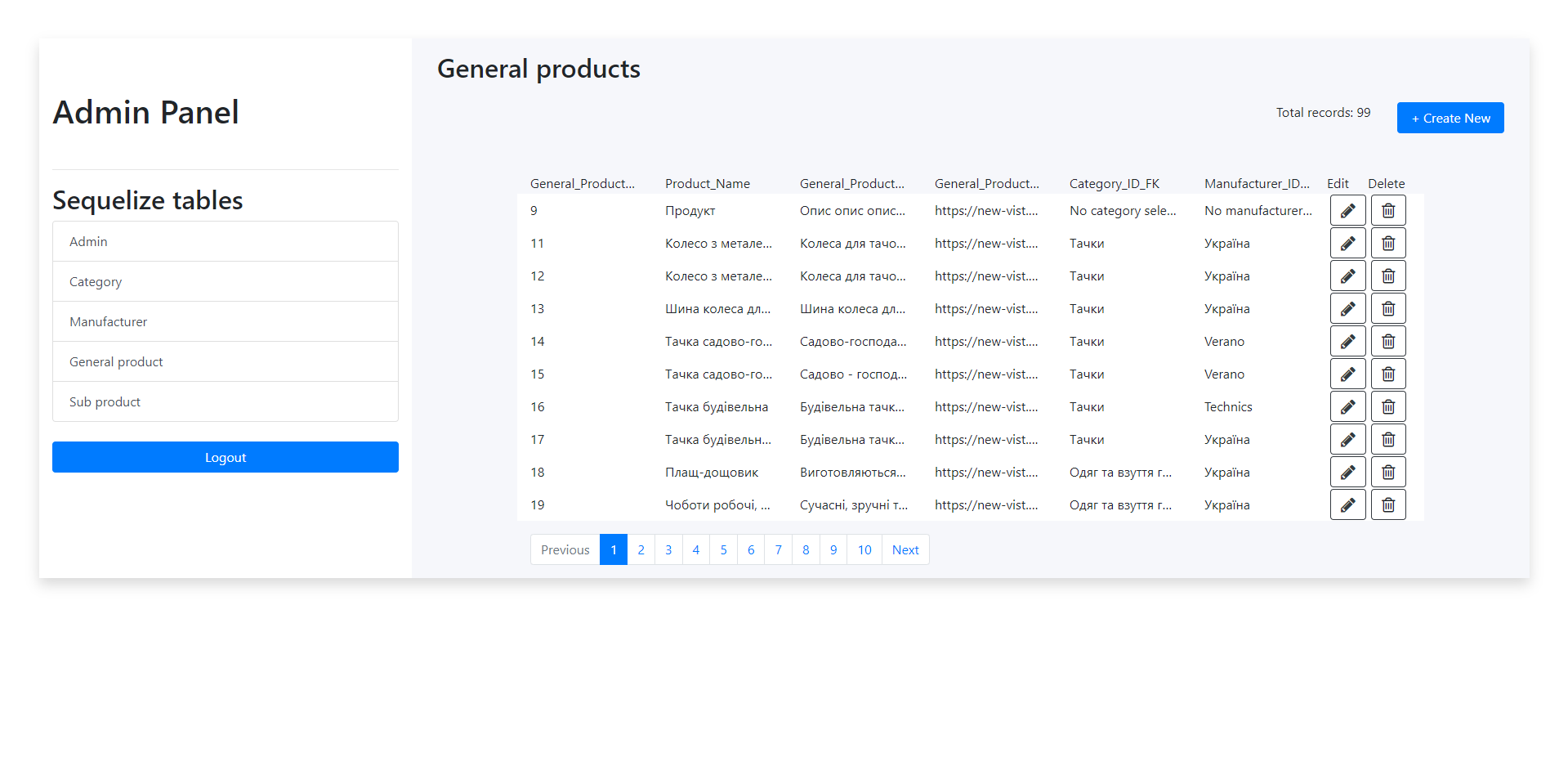


Рис. 3.3. Вигляд адмін-панелі з обраною таблицею

Спробуймо створити новий запис у таблиці General\_Products, для цього потрібно натиснути на кнопку Create New, яка присутня у всіх відображеннях таблиць. З’явиться наступна форма, дані в якій ми і заповнимо.

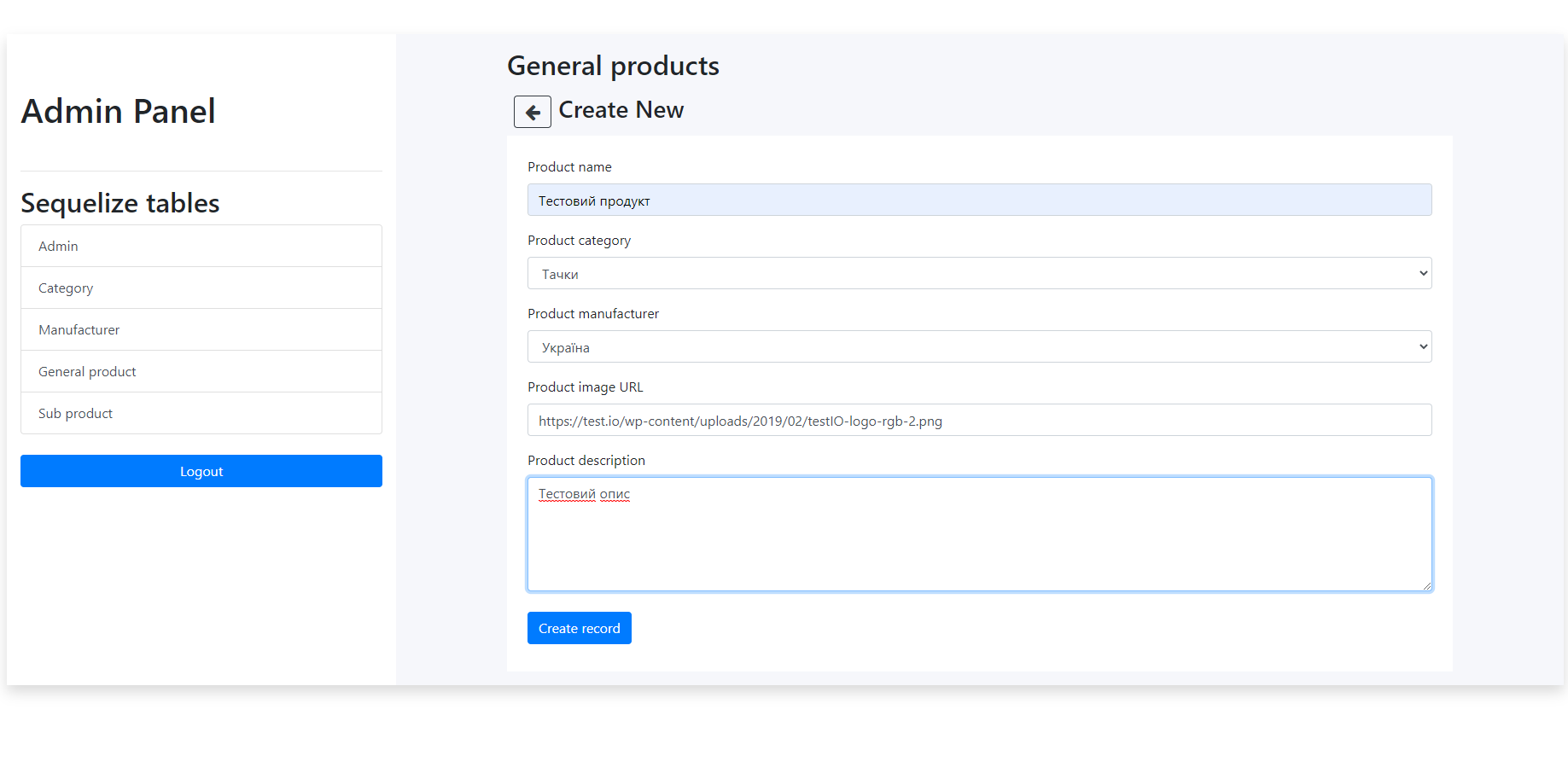


Рис. 3.4. Створення нового запису у таблиці General\_Products

Після вдалого заповнення полів на формі виведеться повідомлення про успішне створення запису у таблиці. Всі дані, які містилися у полях, стираються.

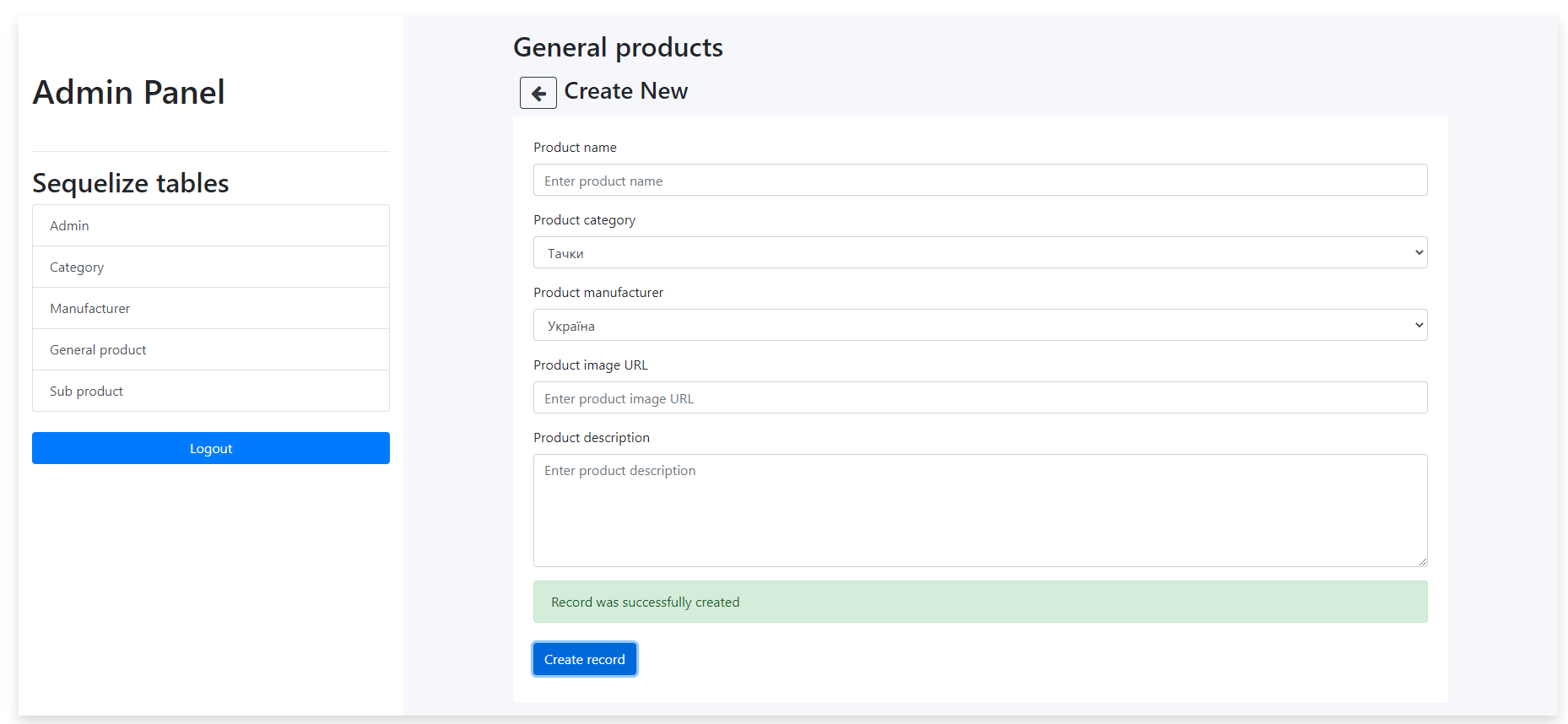


Рис. 3.5. Форма після вдалого створення запису у таблиці General\_Products

Тепер перевірмо, чи справді цей запис створився, повернувшись назад та перейшовши на останню сторінку записів таблиці General\_Products.

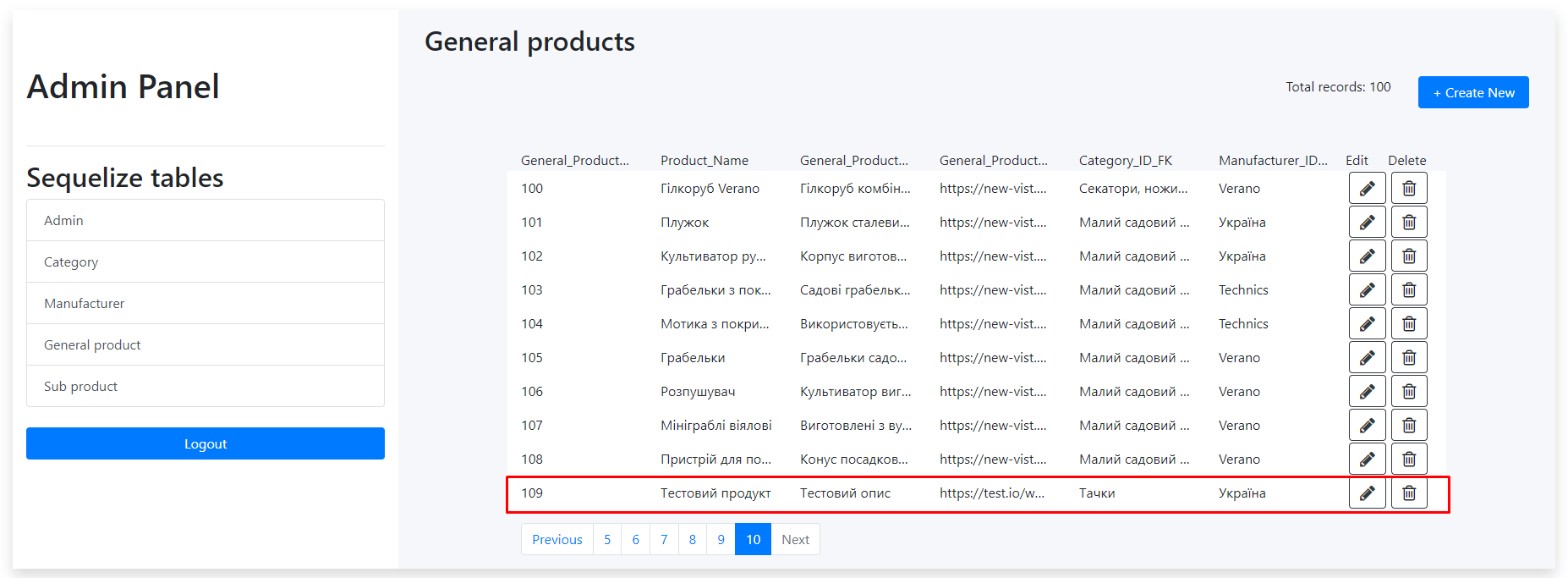


Рис. 3.6. Список всіх записів після вдалого створення запису у таблиці General\_Products

Тепер спробуємо відредагувати створений перед цим запис, натиснувши на кнопку з олівцем. З’явиться наступна форма (рис 3.7, наступна сторінка).

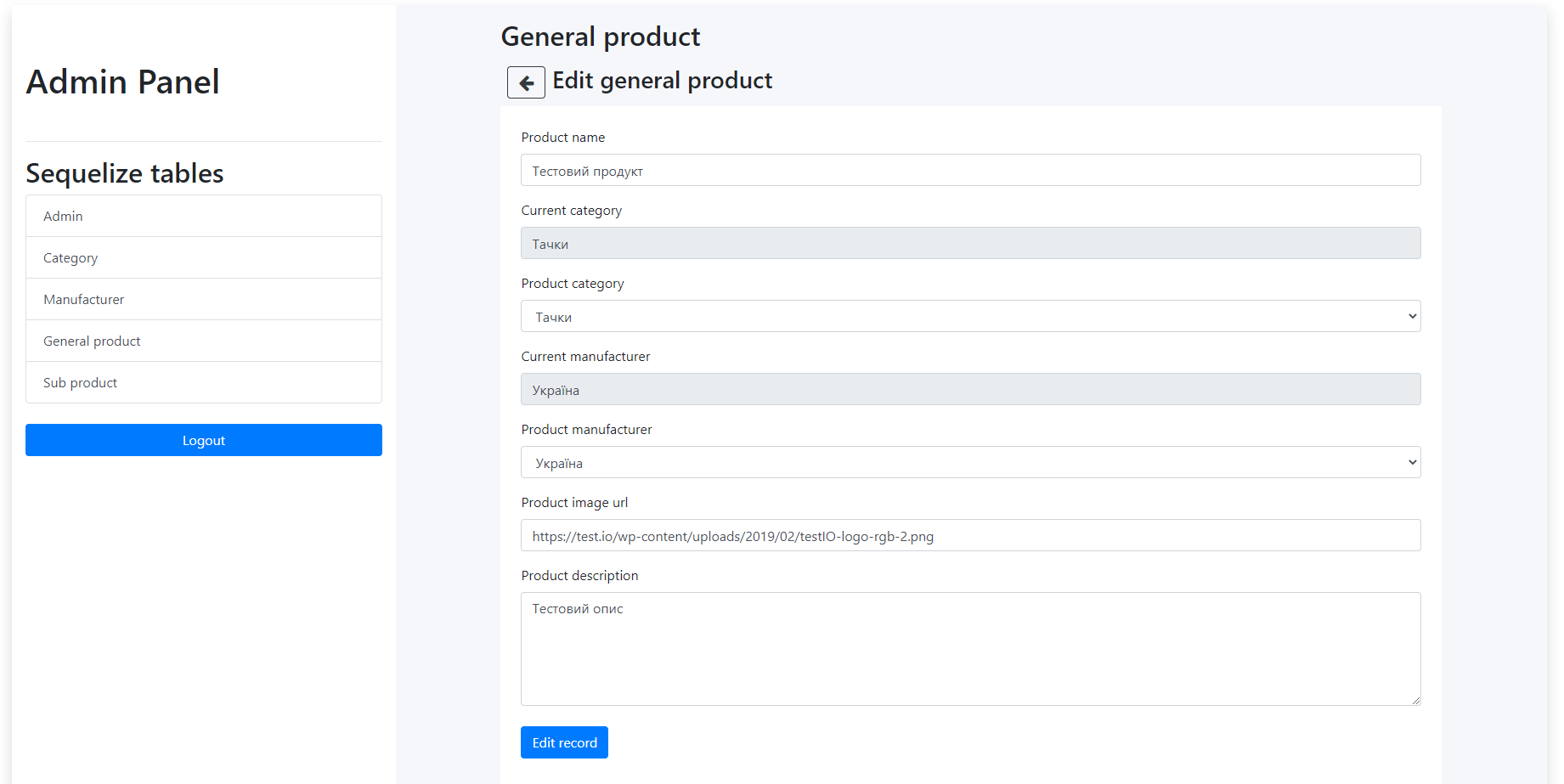


Рис. 3.7. Форма редагування запису

Відразу ж можна редагувати запис. Змінімо деякі поля.

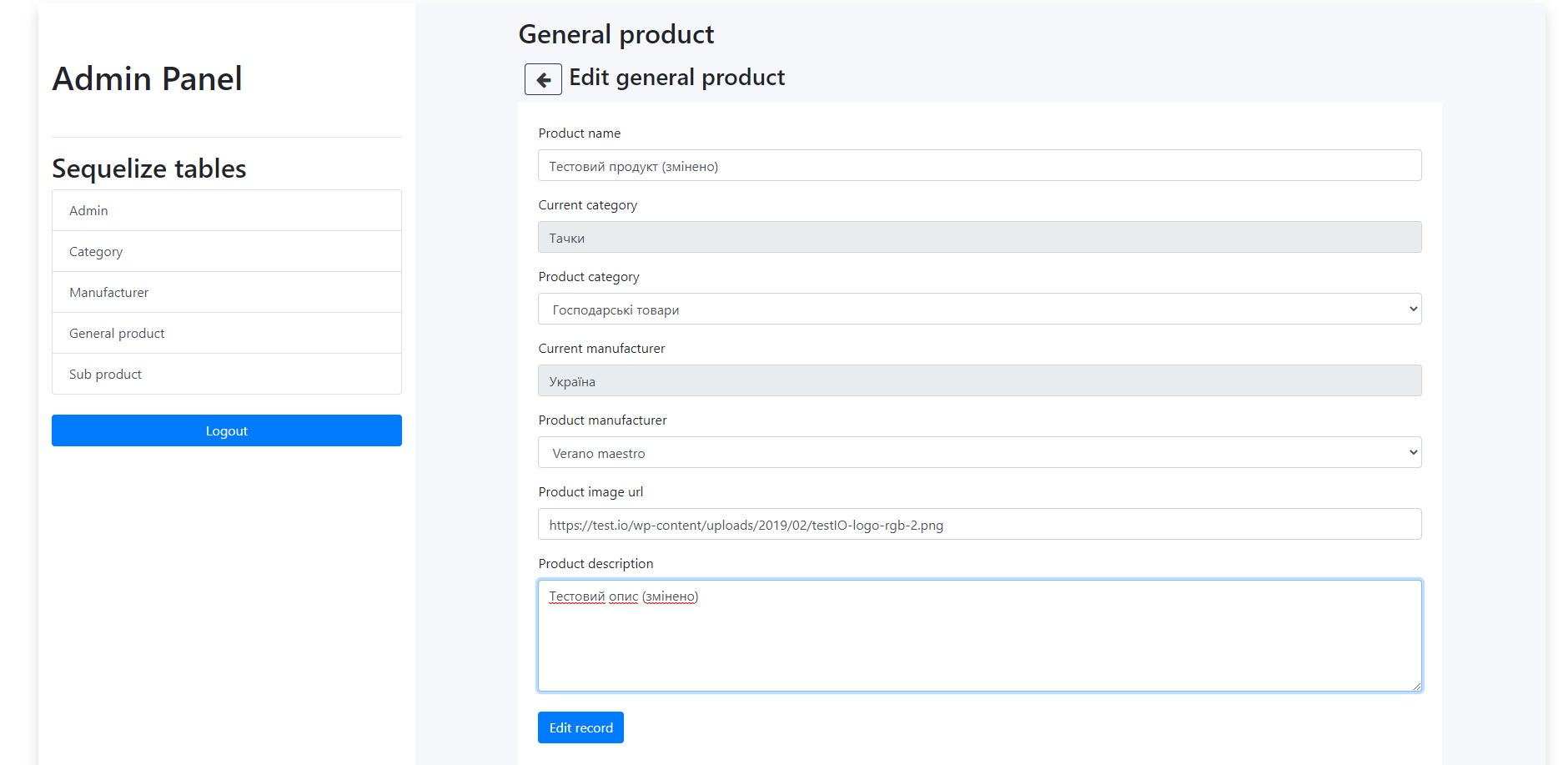


Рис. 3.8. Форма з відредагованим вмістом запису

Після вводу правильних даних до полів, форма матиме наступний вигляд (рис. 3.9, наступна сторінка):

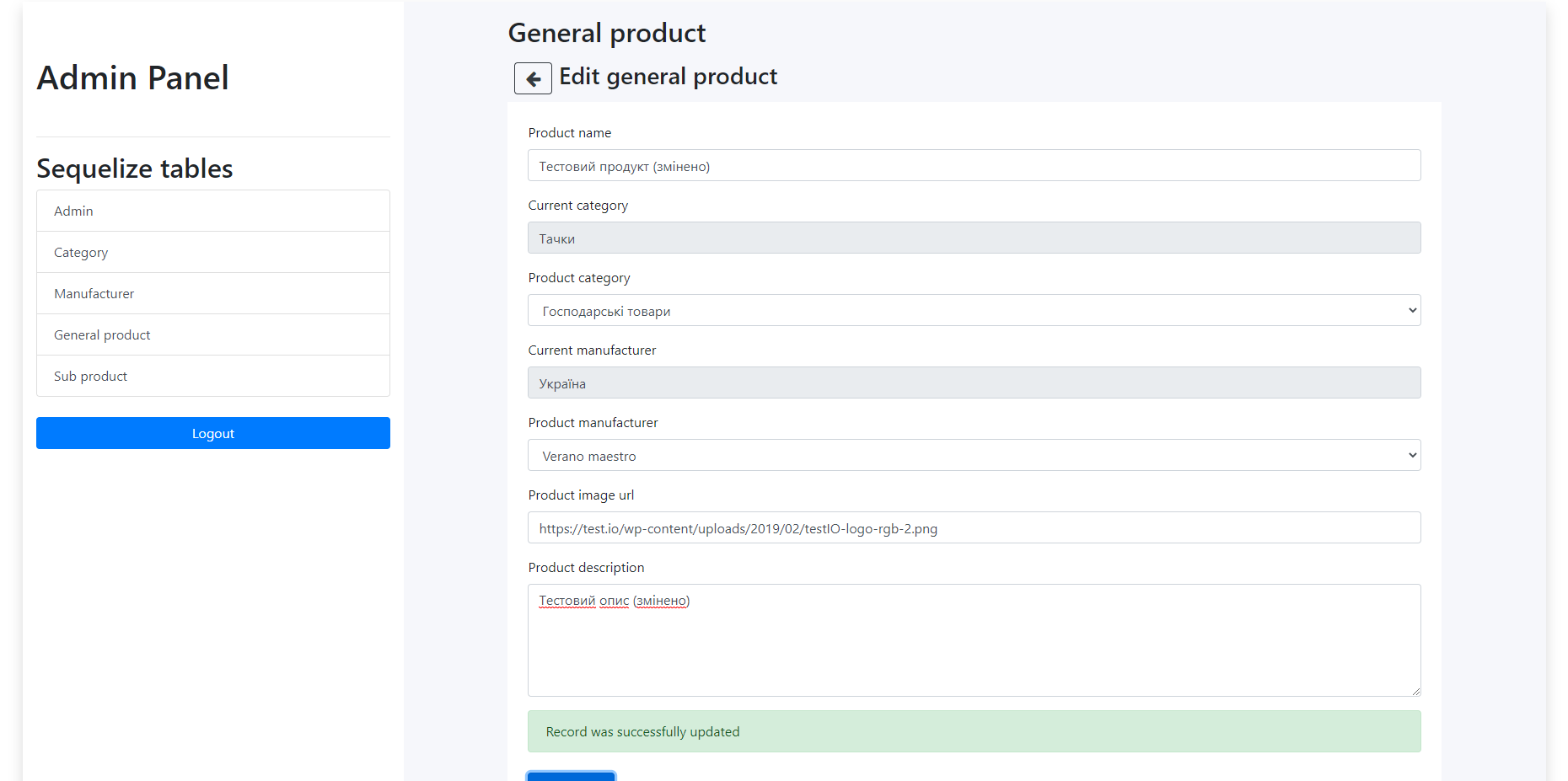


Рис. 3.9. Форма після вдалого редагування запису

Тепер потрібно перевірити, чи справді змінилися дані запису, для цього повернемось на сторінку зі всіма записами таблиці General\_Products.

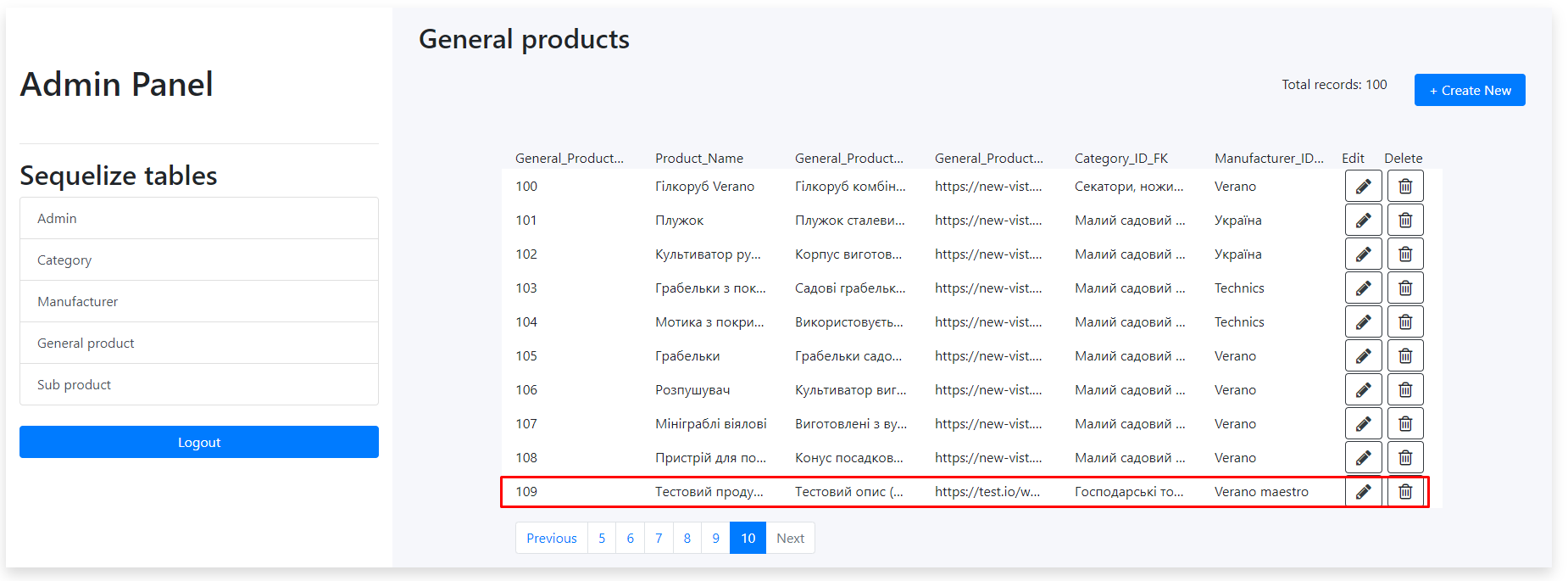


Рис. 3.10. Список всіх записів після редагування

Звісно, може виникнути потреба видаляти записи, для цього можна скористатись іконкою з баком, яка присутня біля кожного запису:

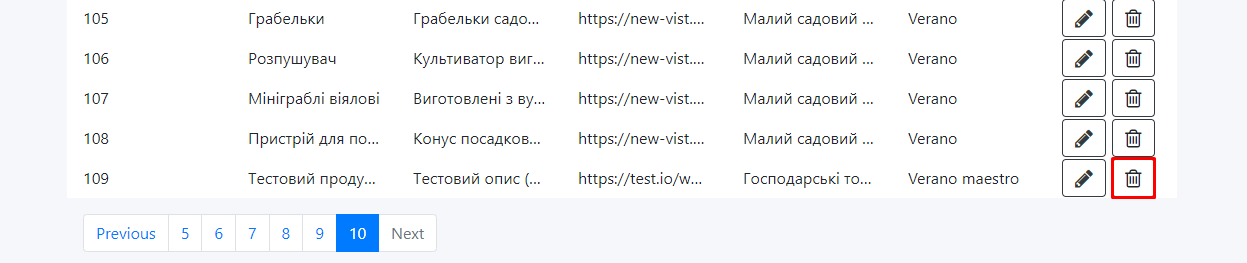


Рис. 3.11. Кнопка видалення обраного запису

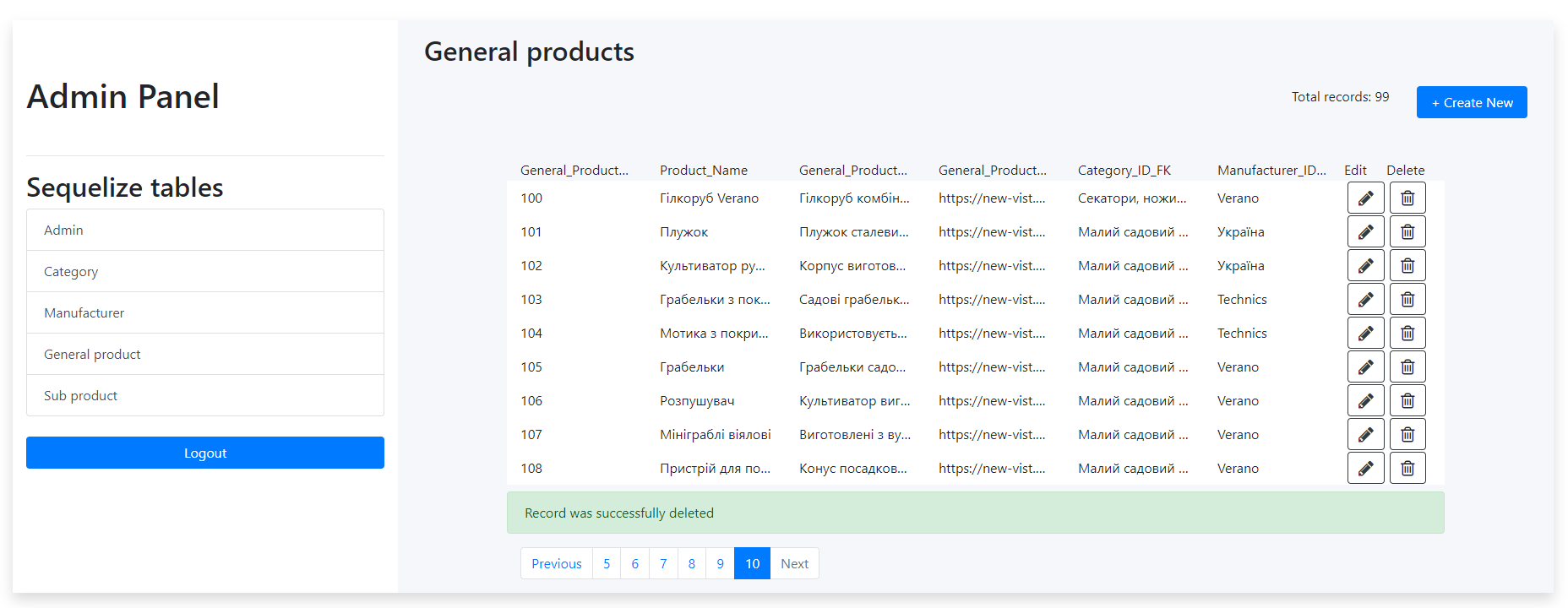


Рис. 3.12. Вигляд всіх записів після успішного видалення запису

Переглядати роботу з іншими таблицями немає сенсу, адже там все працює таким же чином. Після закінчення роботи в системі контролю вмістом, бажано натиснути кнопку Logout, щоб не було проблем зі входом до акаунту користувача в самому інтернет-магазині.

Перейдімо до огляду роботи з інтернет-магазином для звичайного користувача.

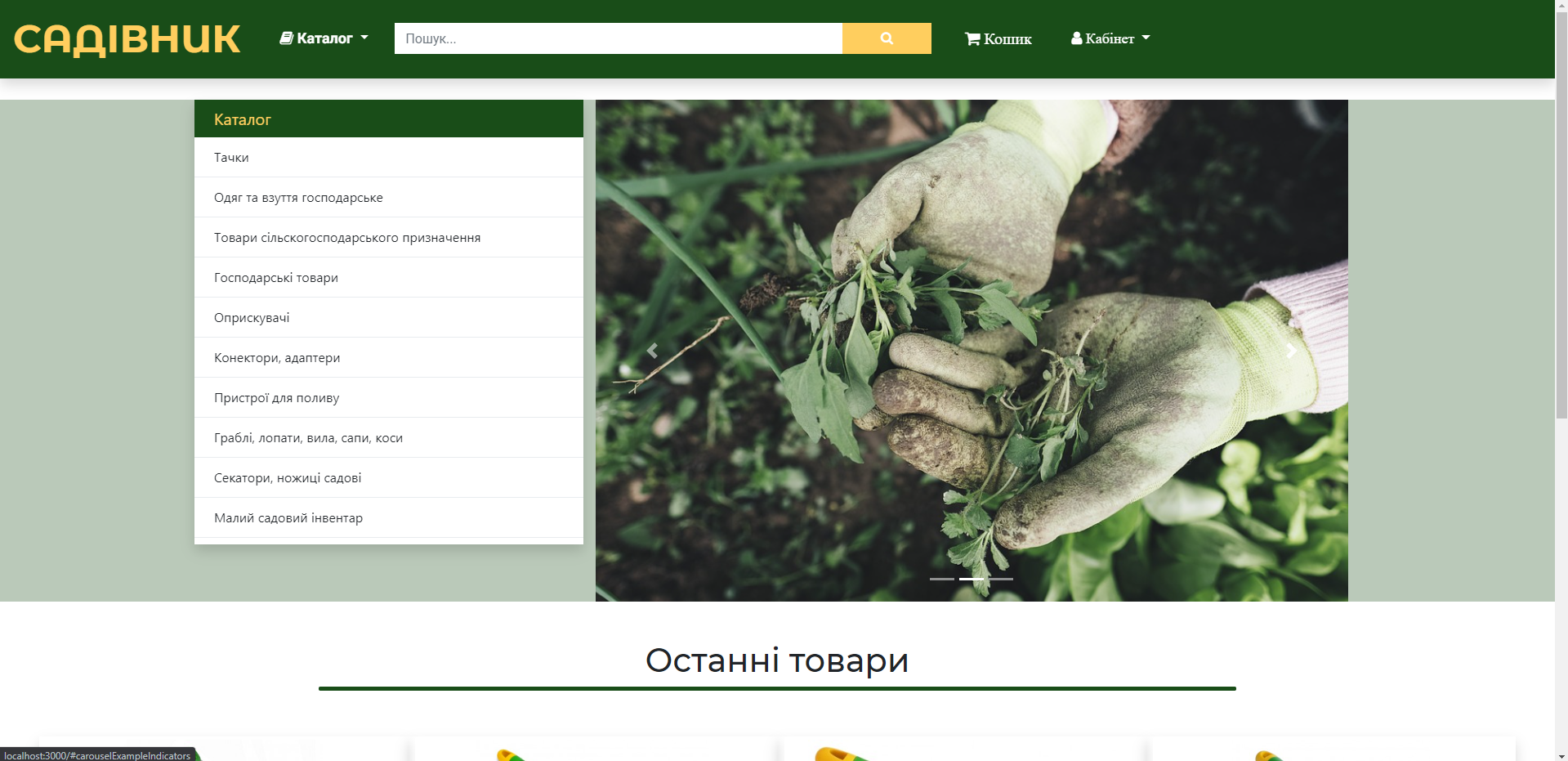


Рис. 3.13. Головна сторінка інтернет-магазину

Нижче знаходяться останні додані товари, де можна перейти до самого товару без обрання категорії або пошуку. Хедер при цьому завжди залишається вгорі, тобто він sticky.

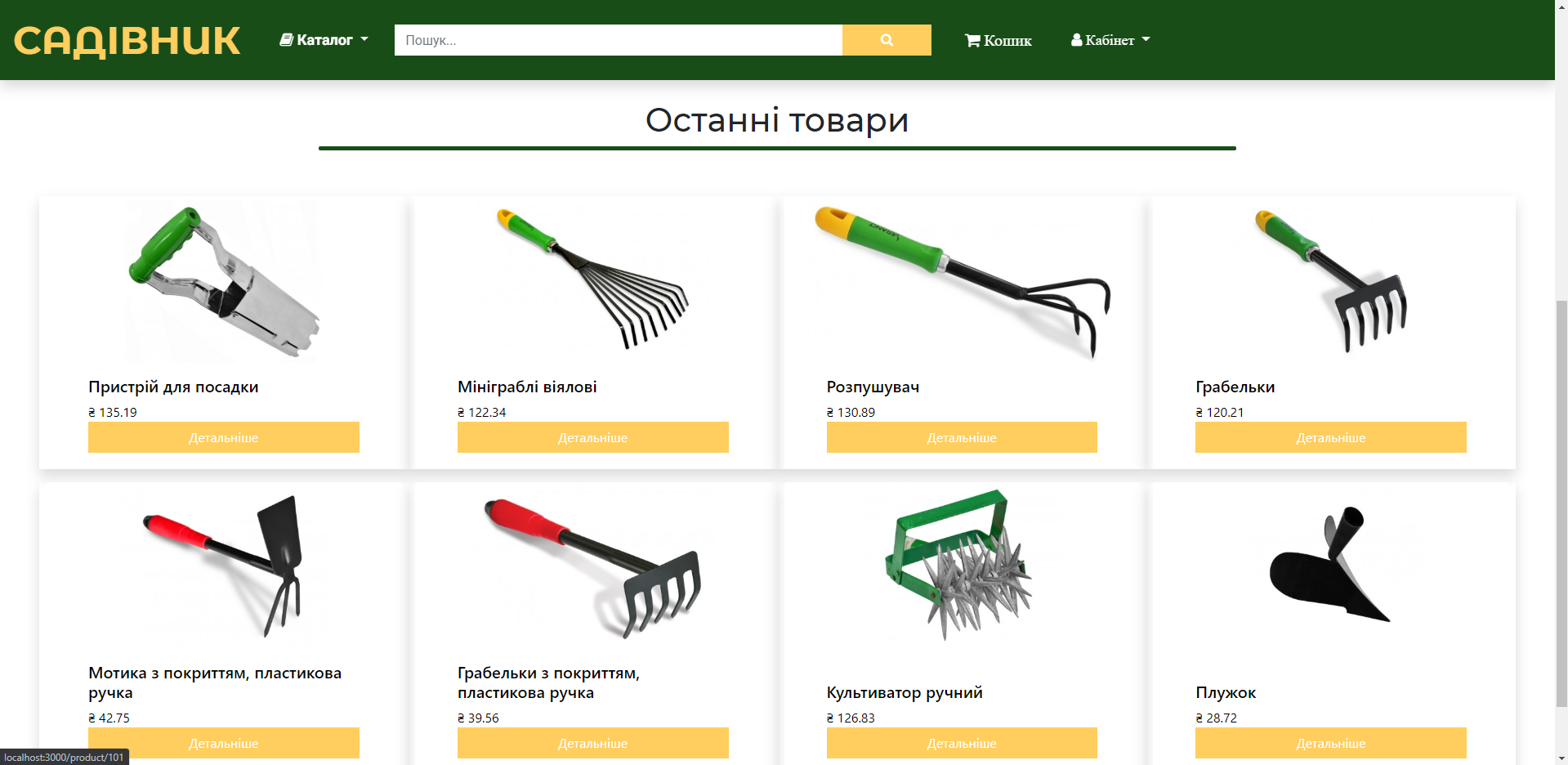


Рис. 3.14. Продовження головної сторінки веб-сайту.

Спробуймо перейти до сторінки якоїсь з категорій. Там буде відображено товари саме цієї категорії:



Рис. 3.15. Сторінка категорії з товарами

Є можливість відфільтрувати товари за такими показниками, як виробник та ціна:

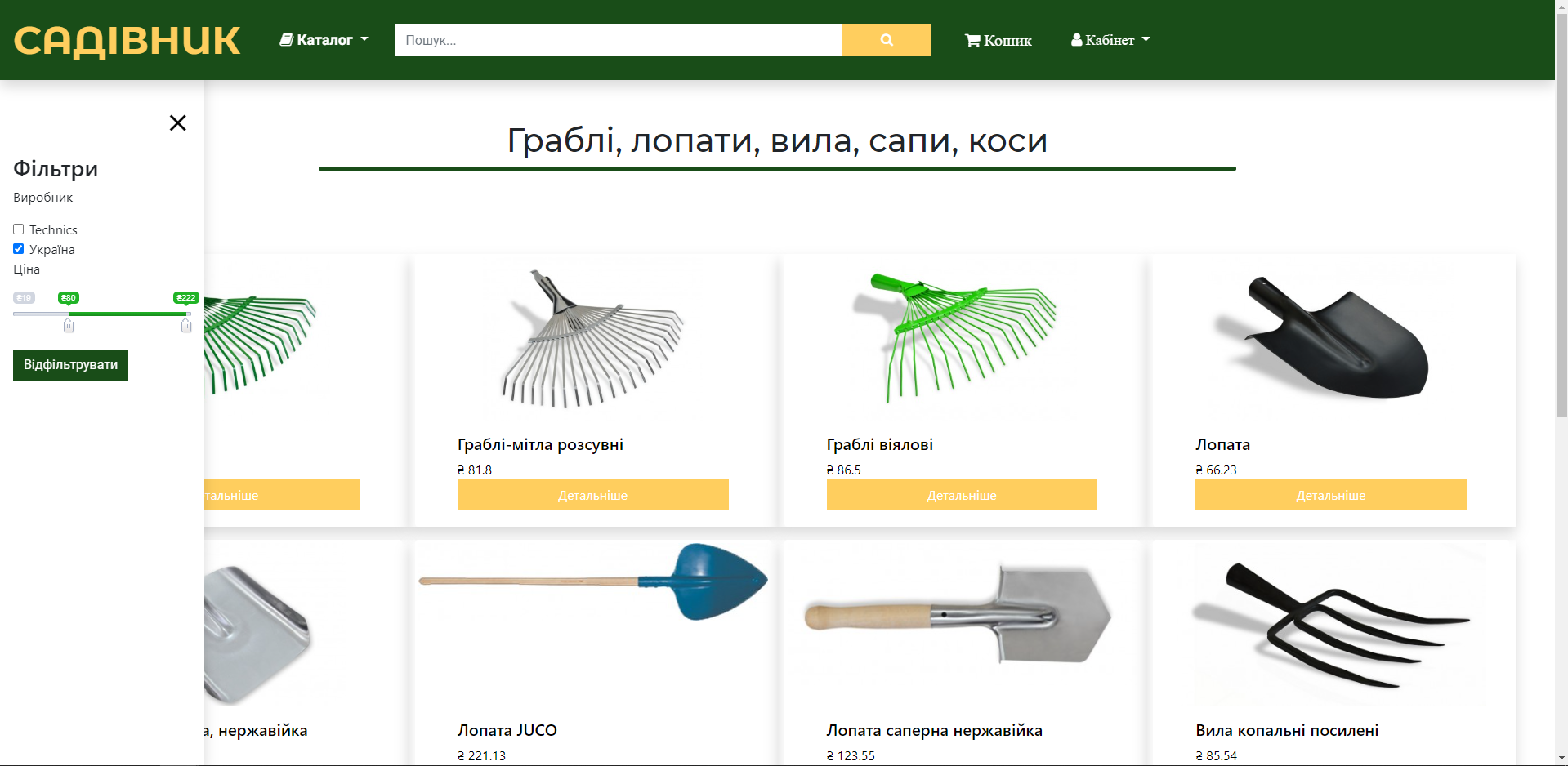


Рис. 3.16. Встановлення фільтрів для продуктів

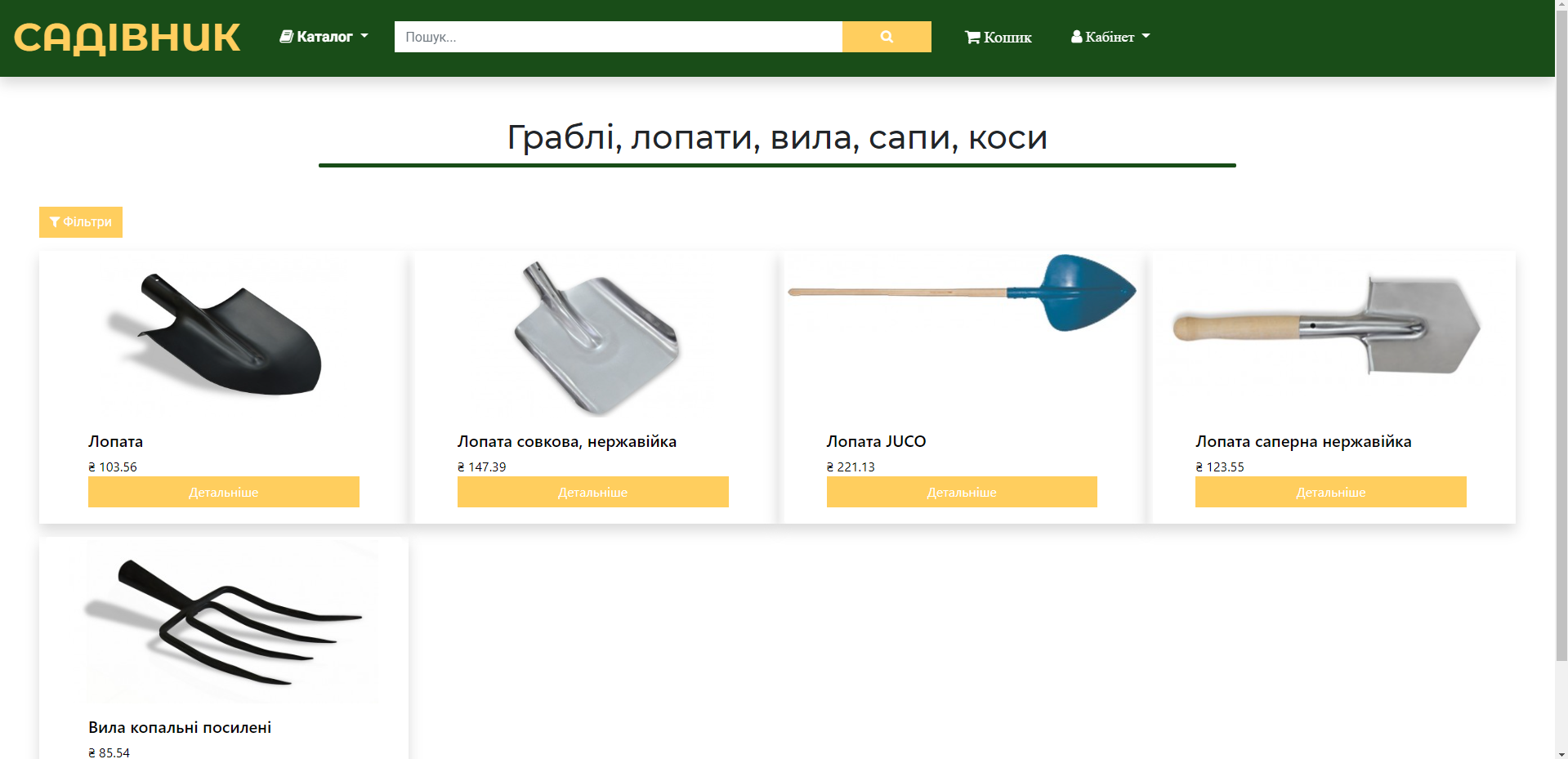


Рис. 3.17. Список відфільтрованих товарів

Також можна скористатись пошуком для знаходження необхідних товарів.

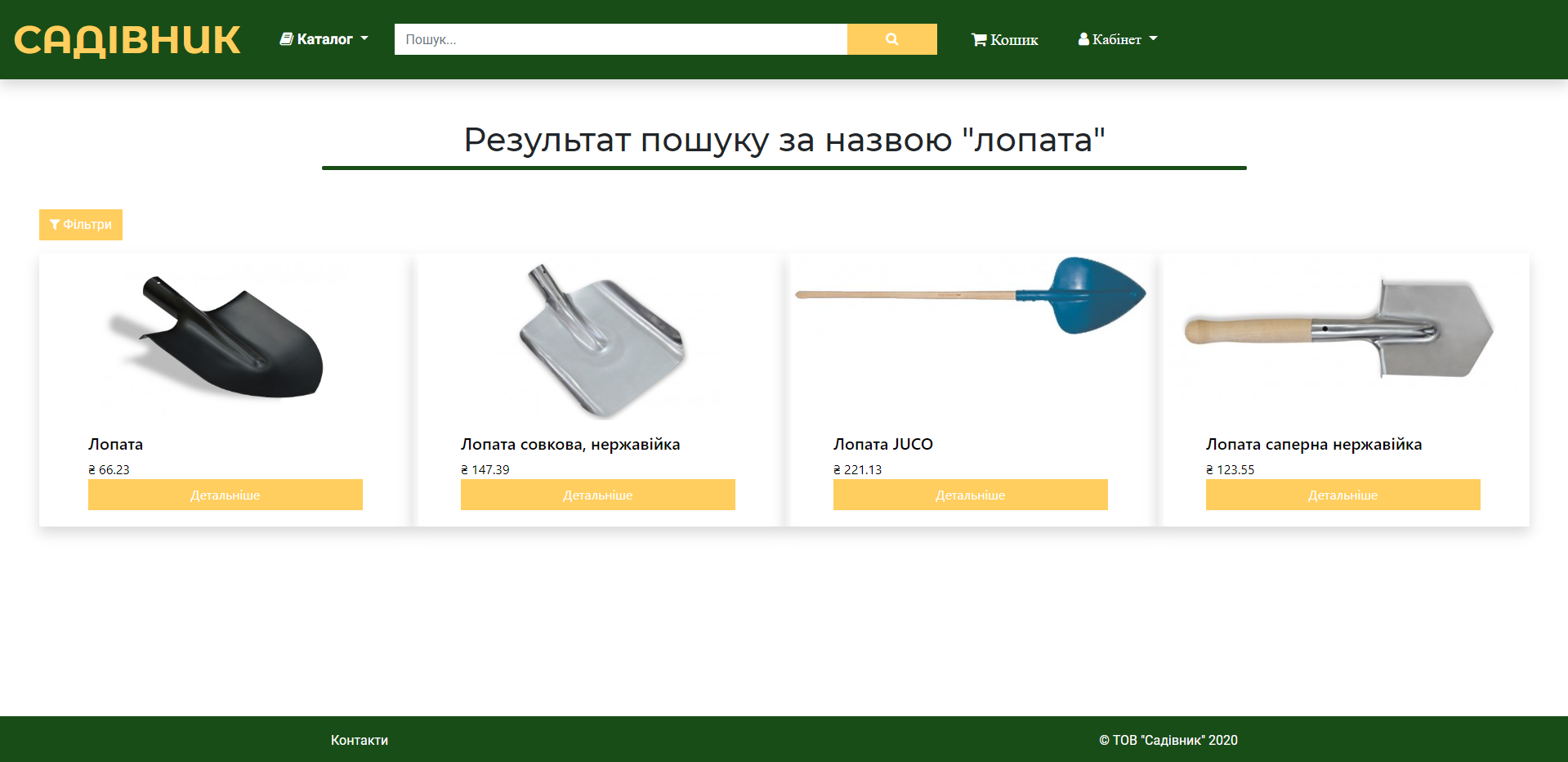


Рис. 3.18. Результат пошуку товарів

Для знайдених товарів також доступна фільтрація, яка працює так само, як і в товарах за категорією. Перейдімо до процесу придбання товару. До сторінки товару можна перейти або натиснувши на сам блок з товаром, або на кнопку Детальніше, яка присутня на кожному блоці товару.

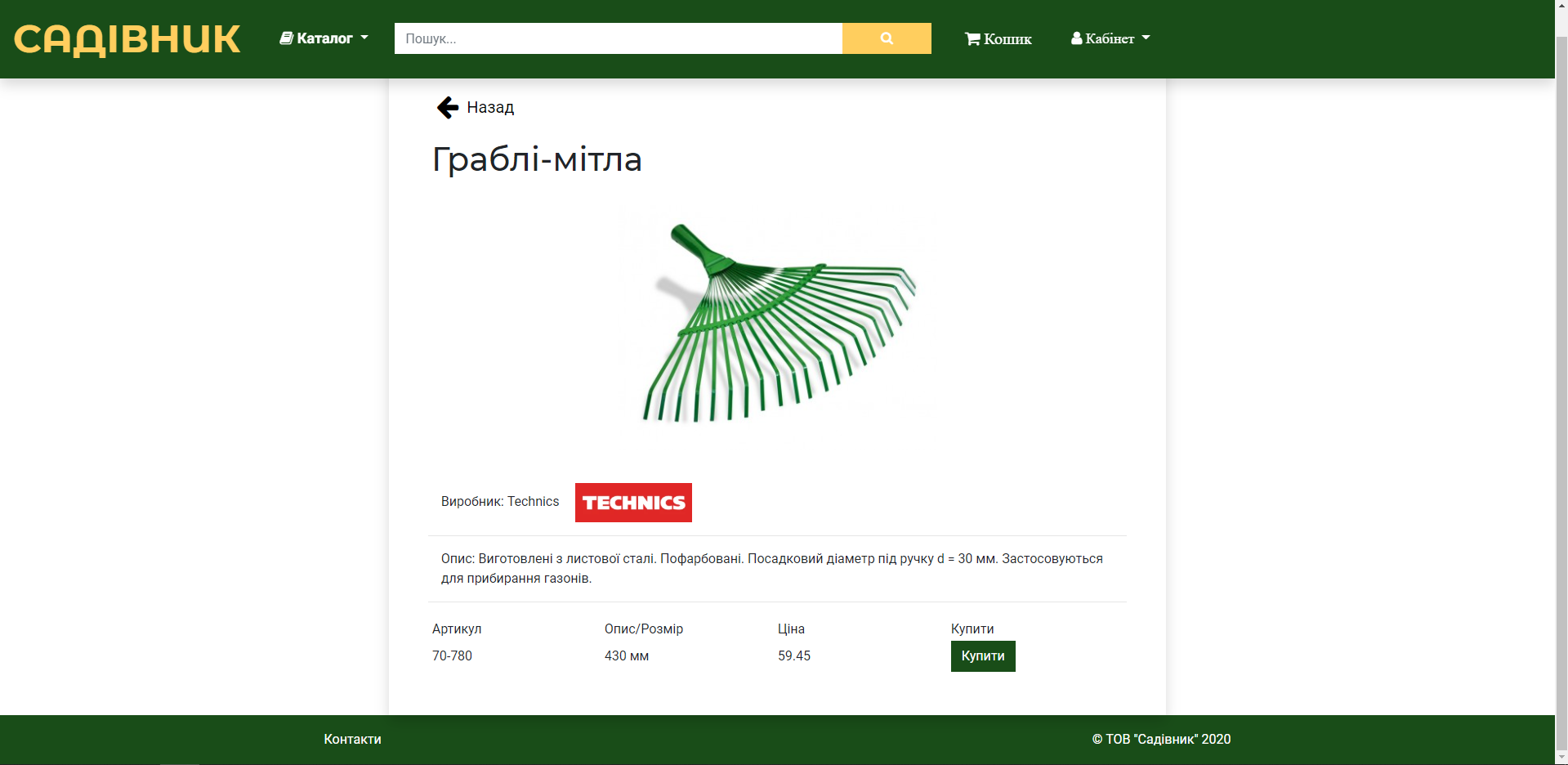


Рис. 3.19. Сторінка товару

Придбати товар можна обравши одну з його варіацій у таблиці, що знаходиться під описом, після цього товар додасться в кошик.

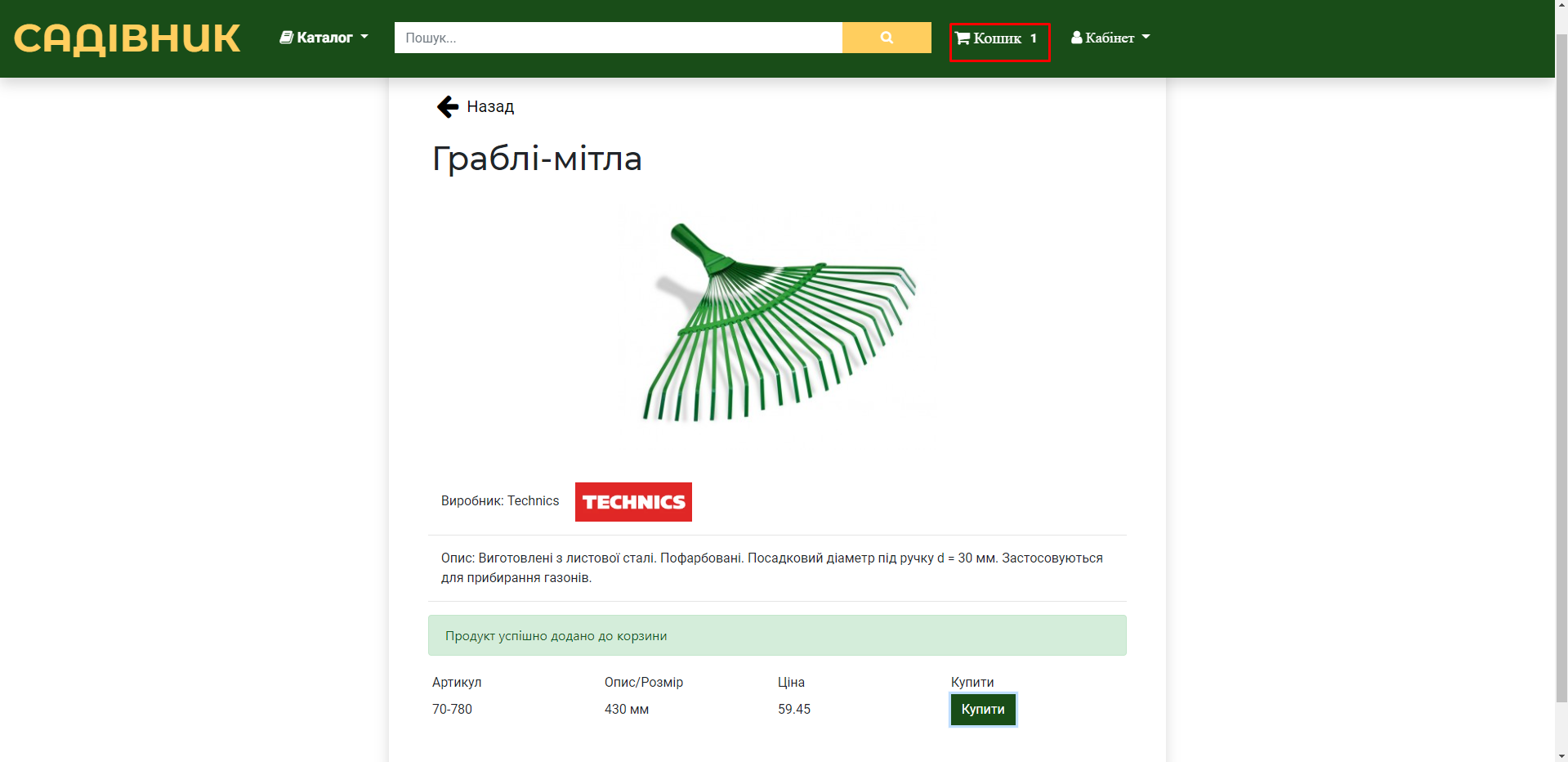


Рис. 3.20. Сторінка товару після успішного його додання до кошику

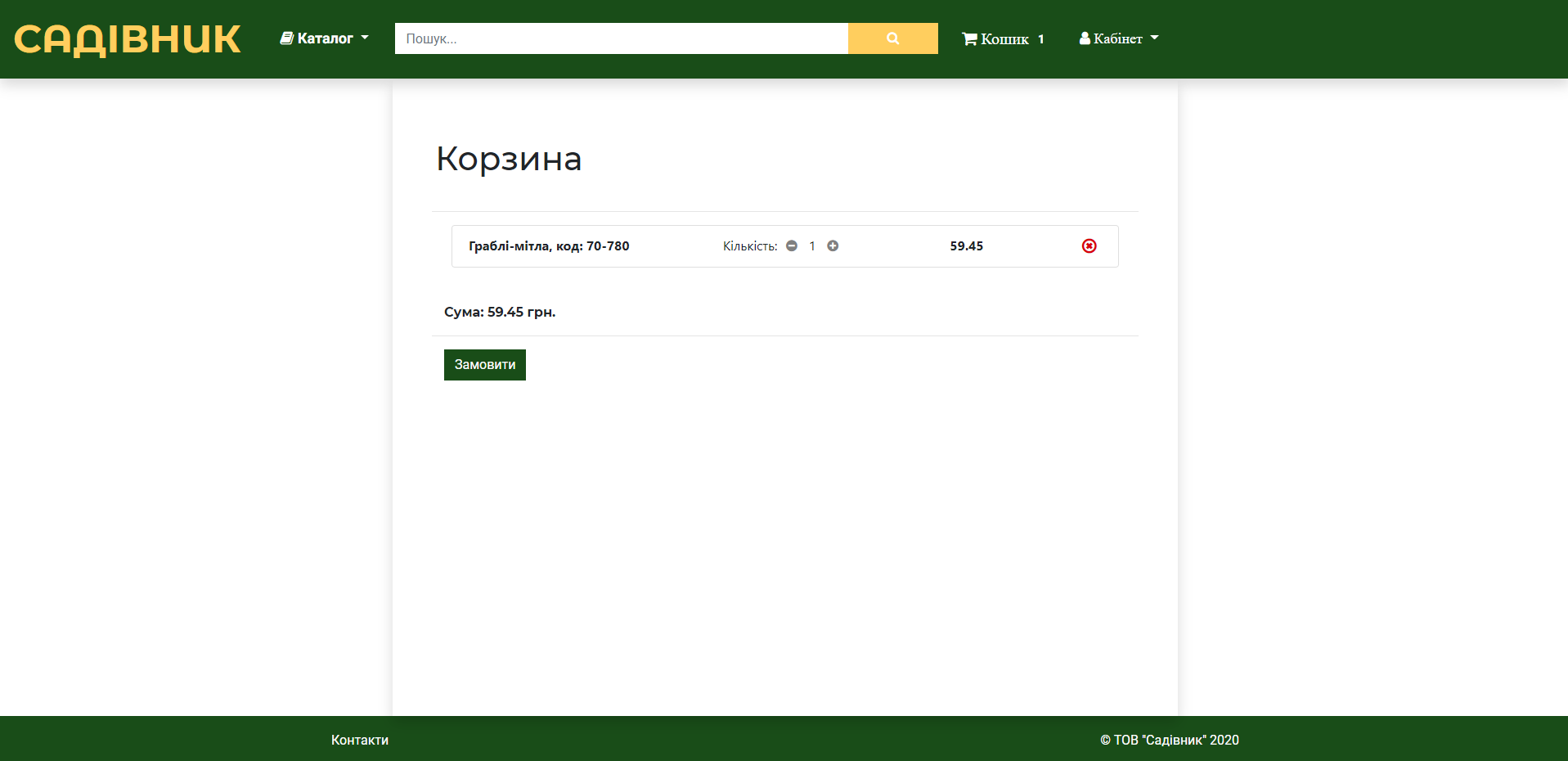


Рис. 3.21. Сторінка корзини з товарами

Перейшовши до корзини, можна змінювати кількість обраних товарів, або взагалі видалити його зі списку.

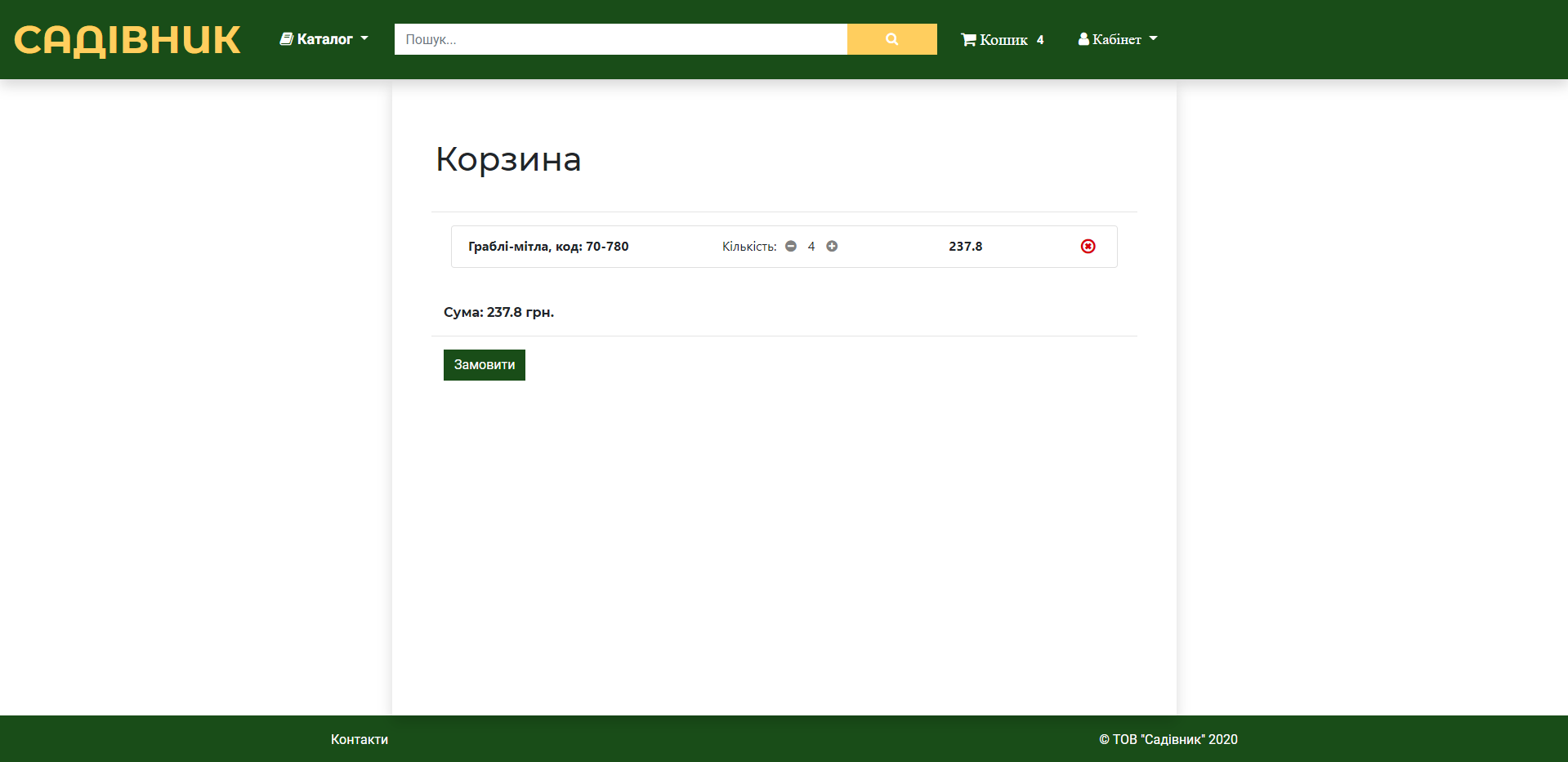


Рис. 3.22. Сторінка корзини зі зміненою кількістю товарів

Після обрання необхідної кількості товару можна перейти до їх замовлення, але цього не можна зробити, якщо користувач не авторизований. Тому при натисненні на кнопку Замовити з’явиться наступне вікно (рис. 3.23, наступна сторінка):

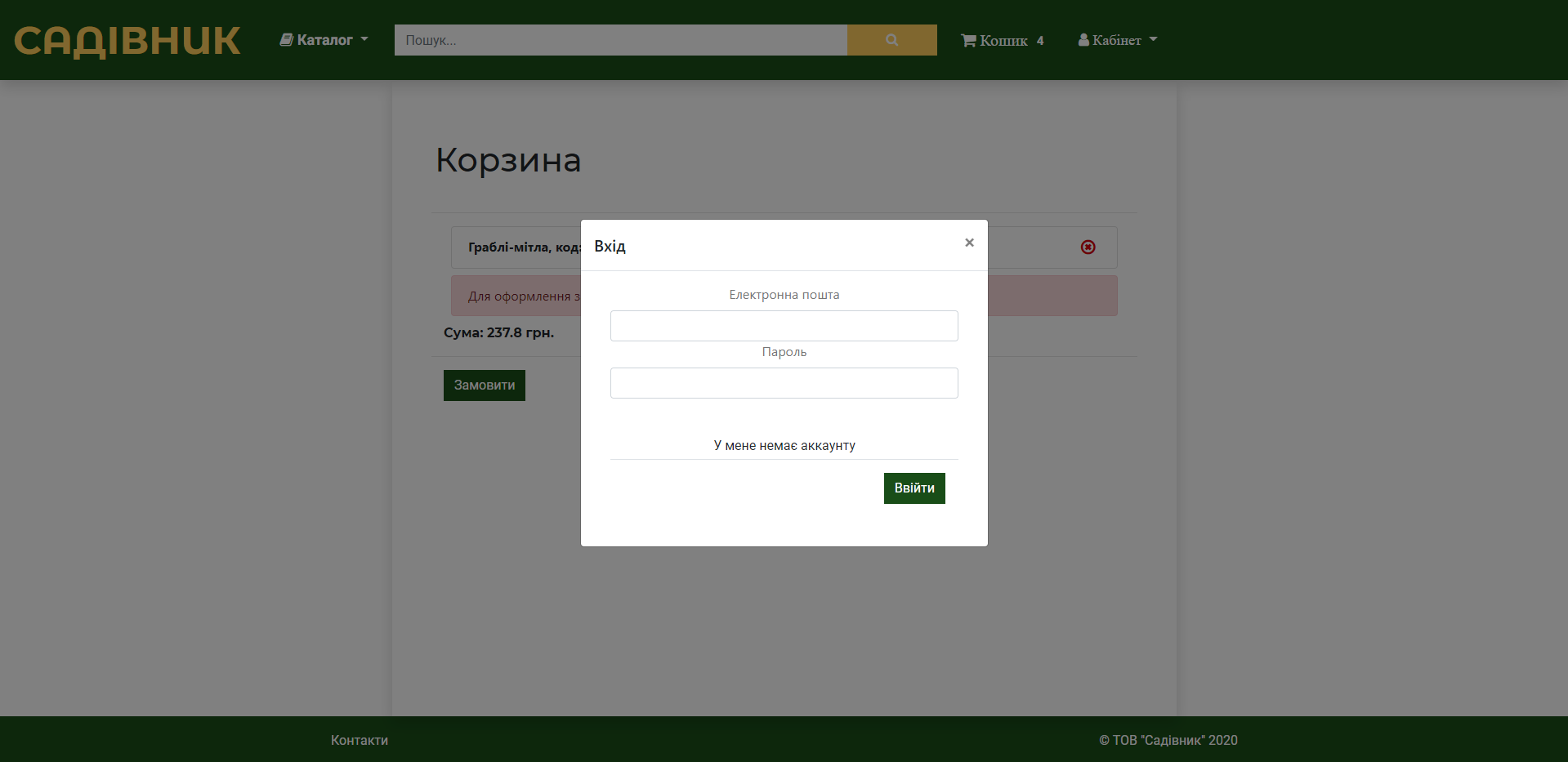


Рис. 3.23. Вигляд сторінки без авторизації при замовленні

Якщо у користувача немає акаунту, то він може його створити, натиснувши на кнопку У мене немає акаунту, після чого з’явиться наступна форма:

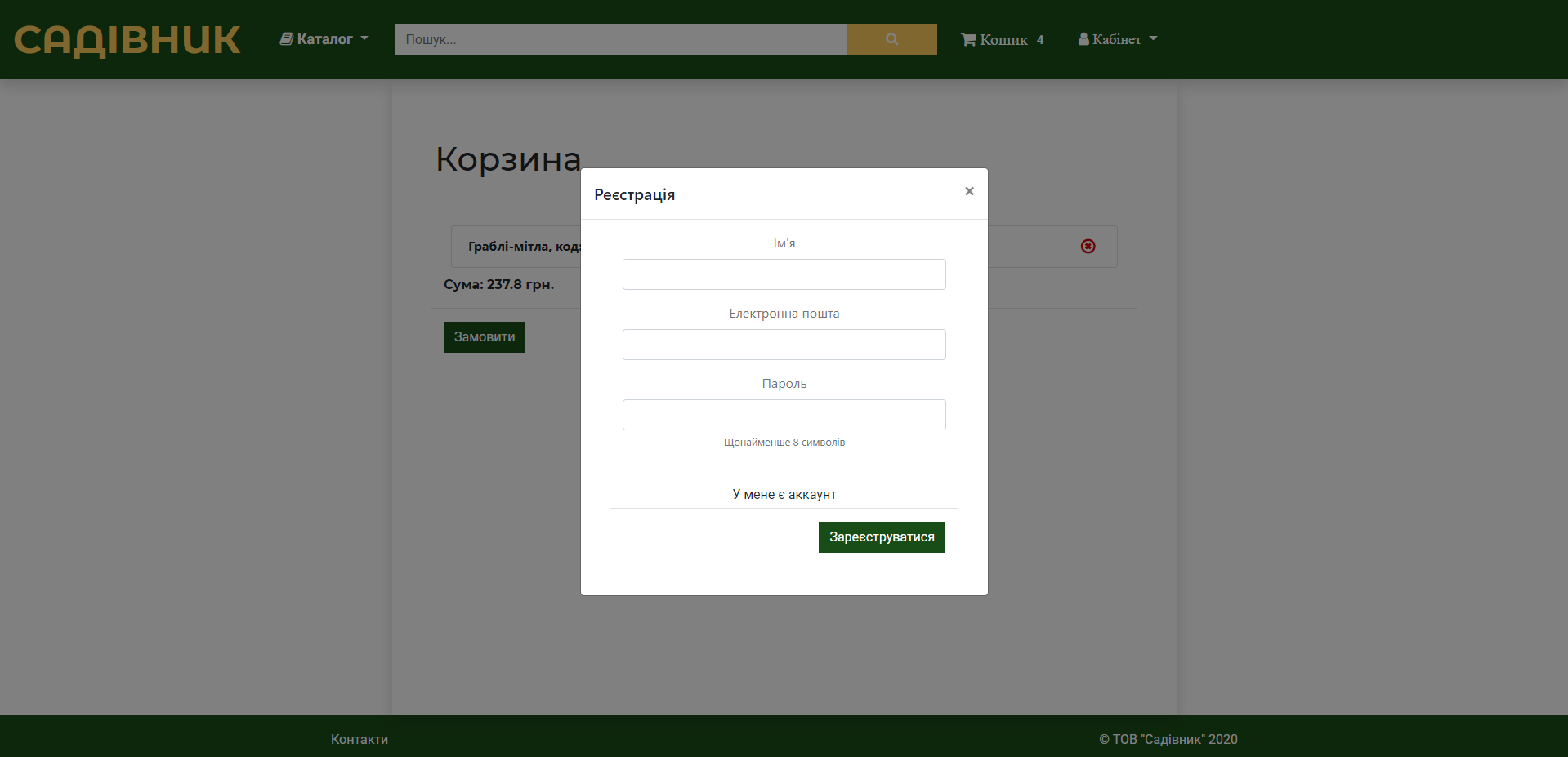


Рис 3.24. Вигляд сторінки з формою для реєстрації

Після вдалої авторизації користувача одразу буде перенаправлено до сторінки, де можна оформити замовлення. У полях буде деяка інформація, яка присутня у персональному акаунті клієнта.

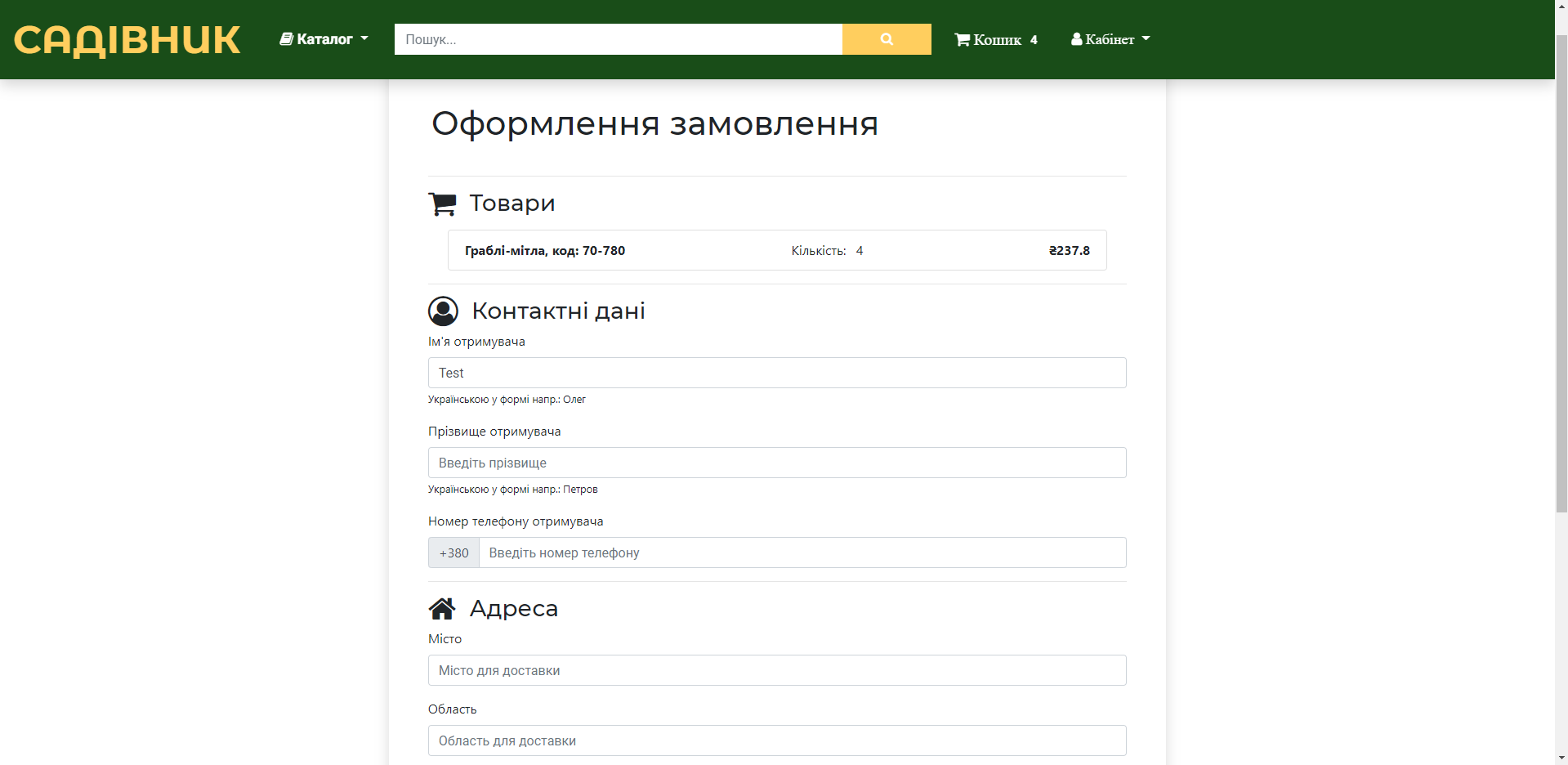


Рис. 3.25. Вигляд сторінки для оформлення замовлення

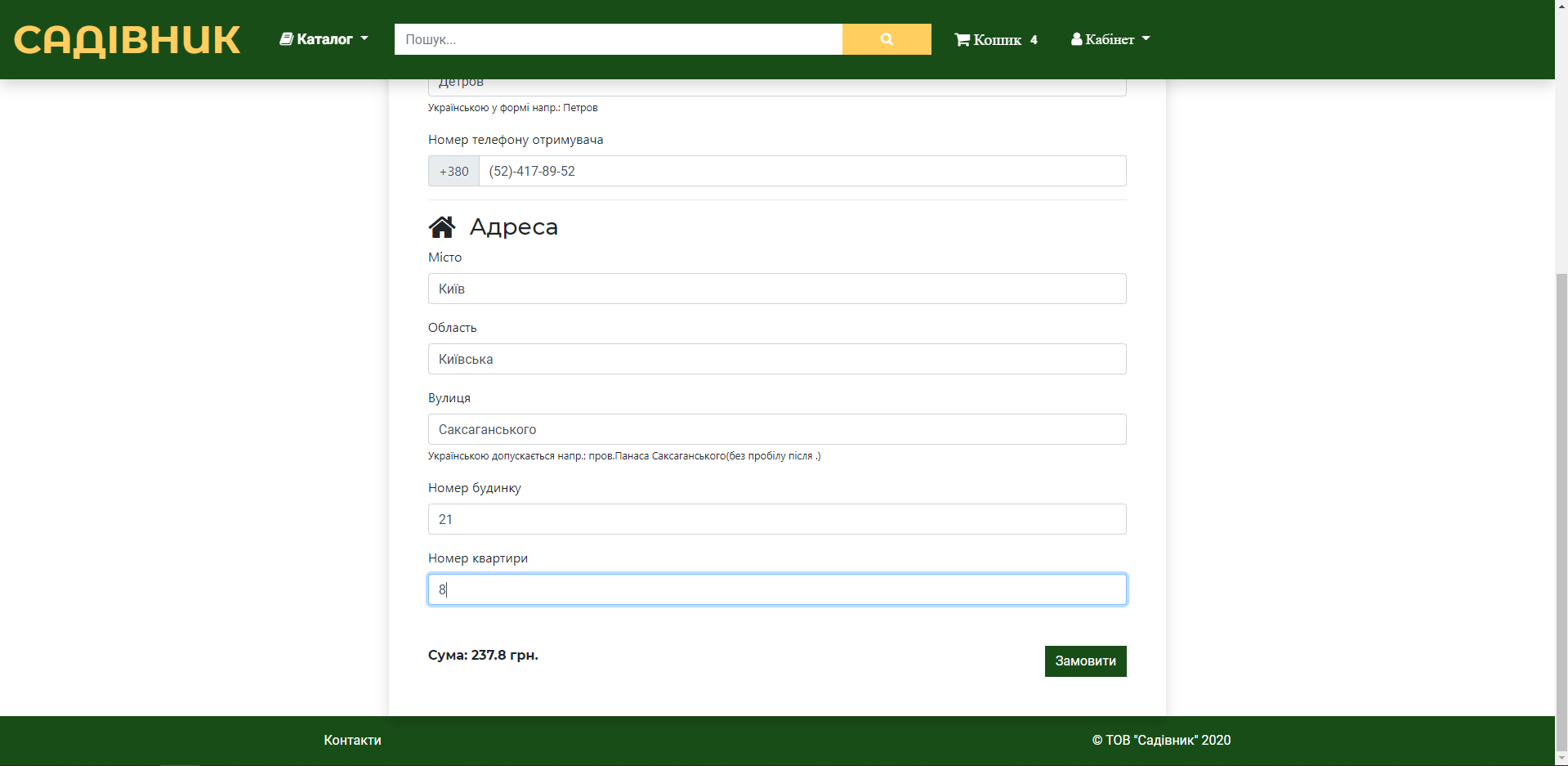


Рис. 3.26. Вигляд сторінки для оформлення замовлення з заповненими даними

Якщо всі необхідні поля були заповнені правильно, то на сторінці з’явиться наступне повідомлення (наступна сторінка, рис. 3.27):

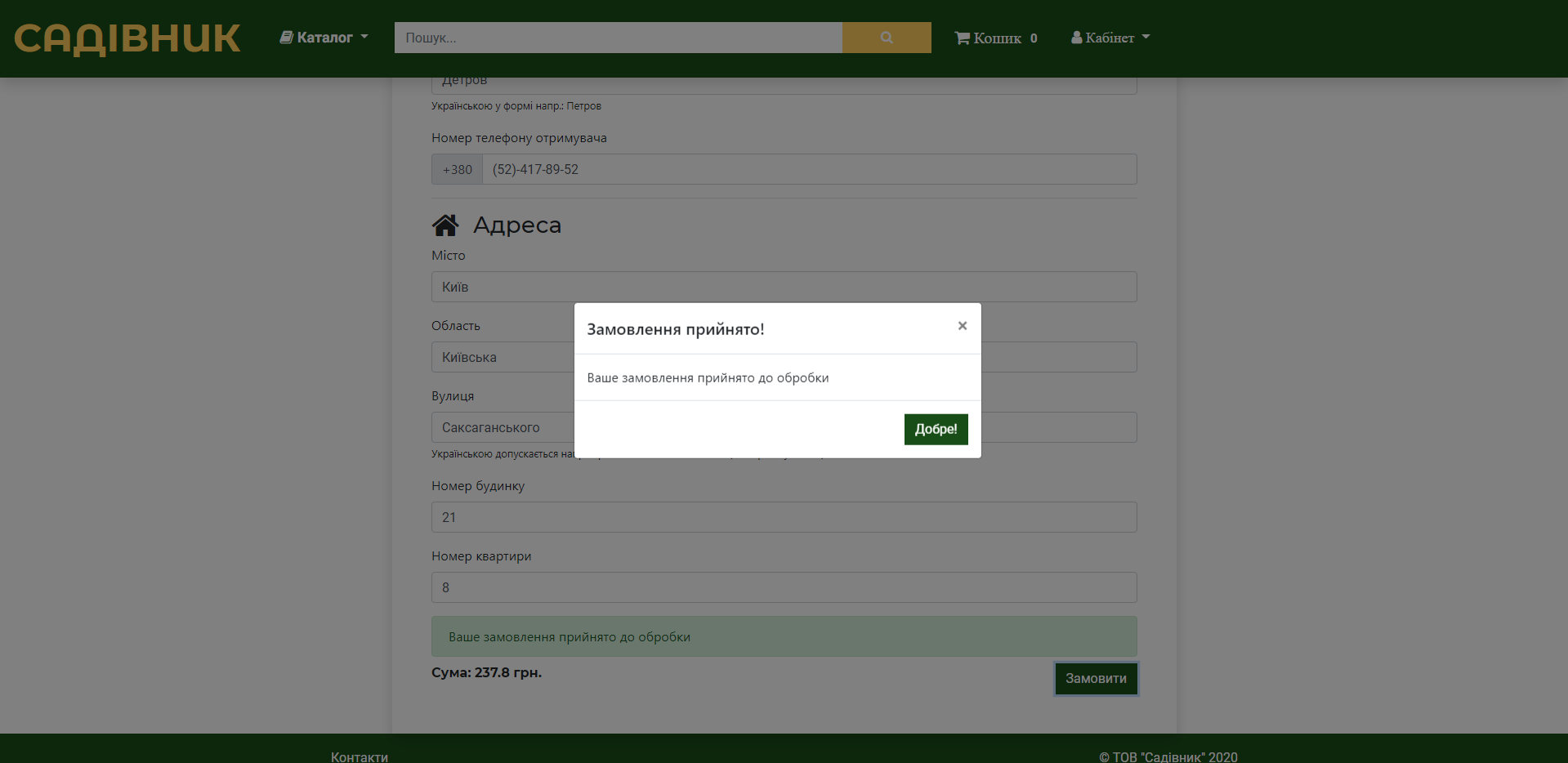


Рис. 3.27. Вигляд сторінки після вдалого замовлення

Всі успішні замовлення можна переглянути у особистому кабінеті.

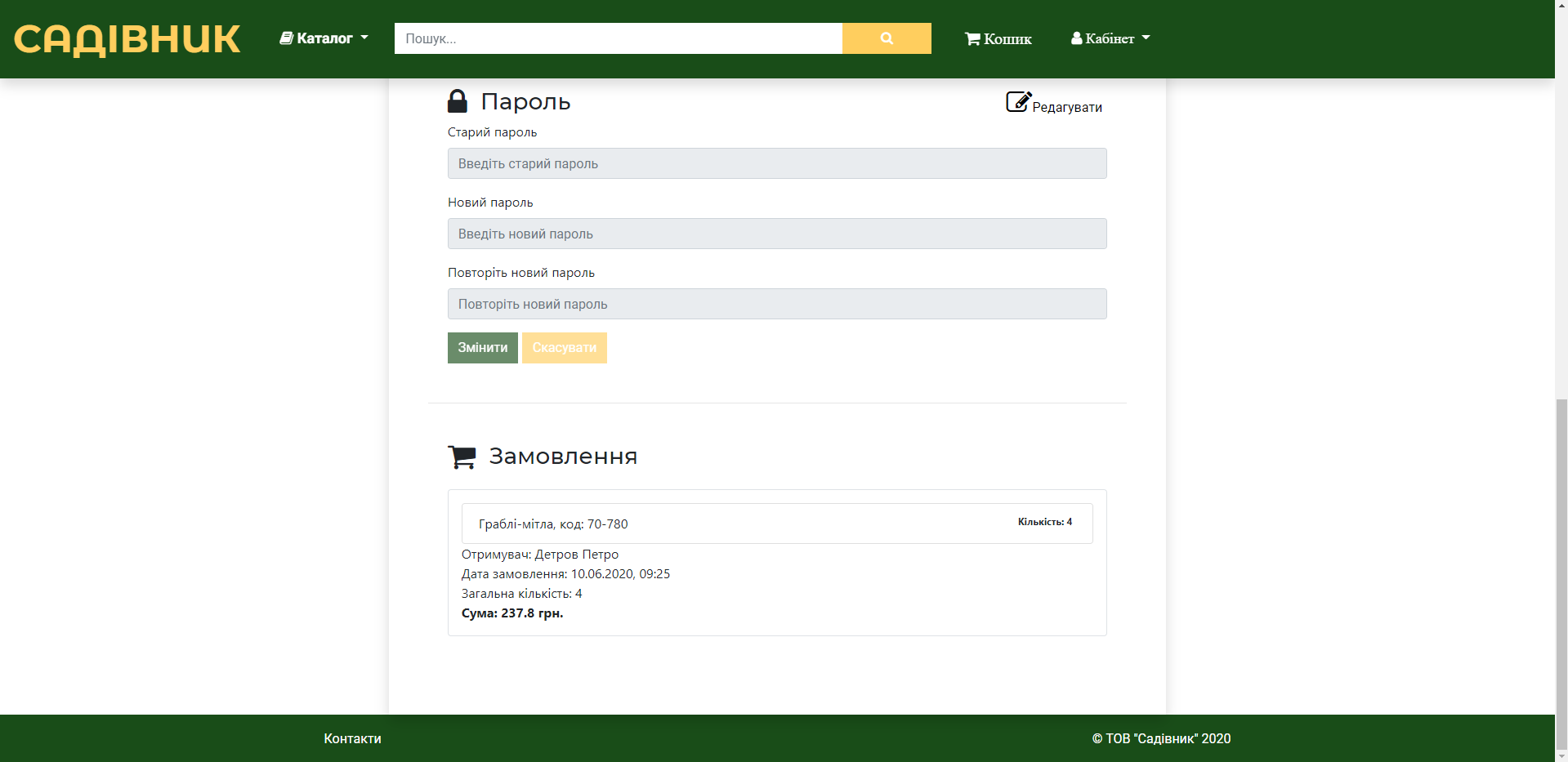


Рис. 3.28. Замовлення кориистувача у особистому кабінеті

Також у особистому кабінеті можна змінити такі свої дані, як: ім’я, прізвище, по-батькові, дату народження, стать, контактний номер телефону та пароль. Для прикладу змінімо деякі особисті дані, для цього у частині форми з назвою Особисті дані потрібно натиснути кнопку Редагувати.

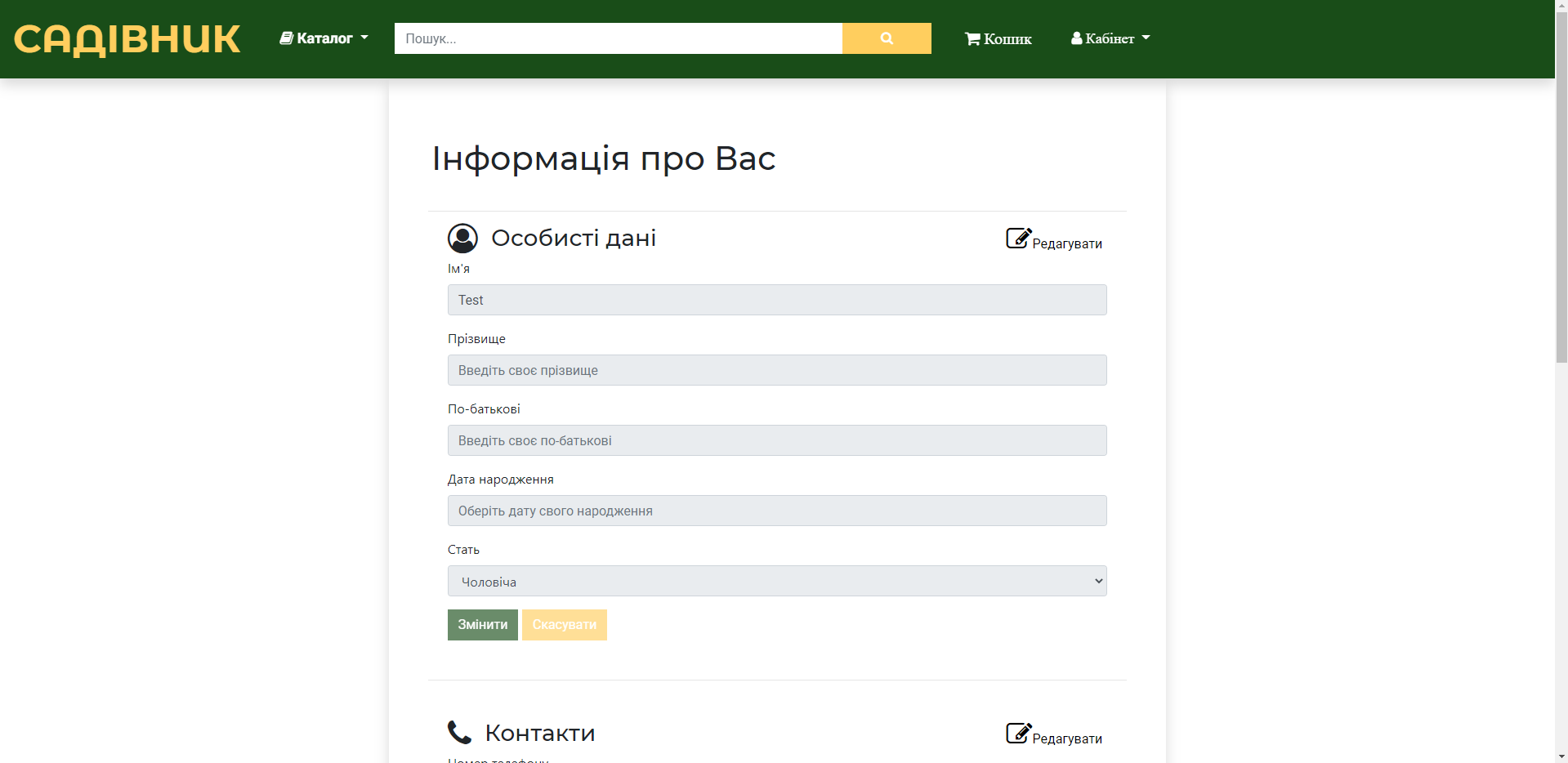


Рис. 3.29. Форма для зміни особистих даних

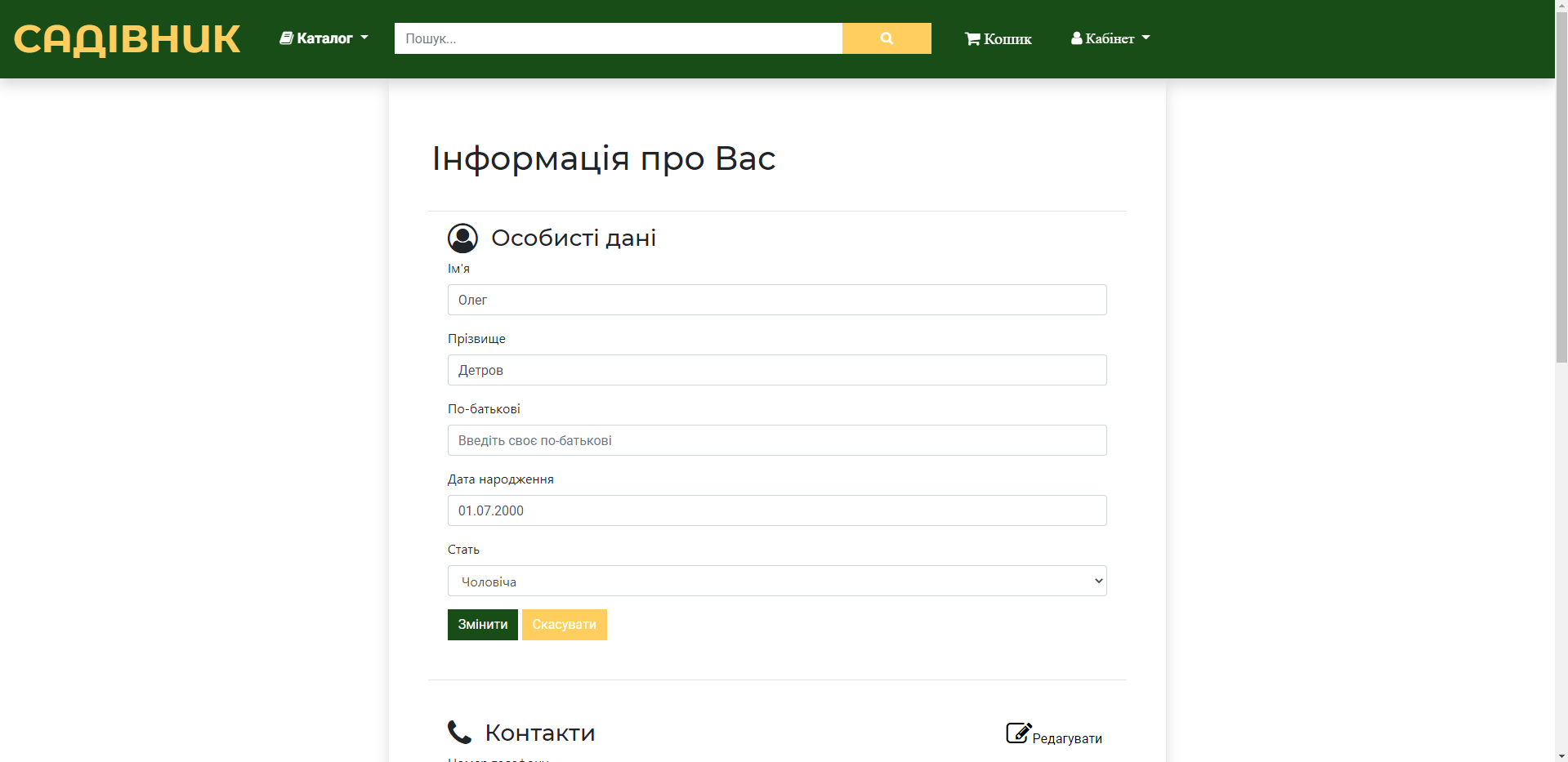


Рис. 3.30. Заповнена форма для зміни особистих даних

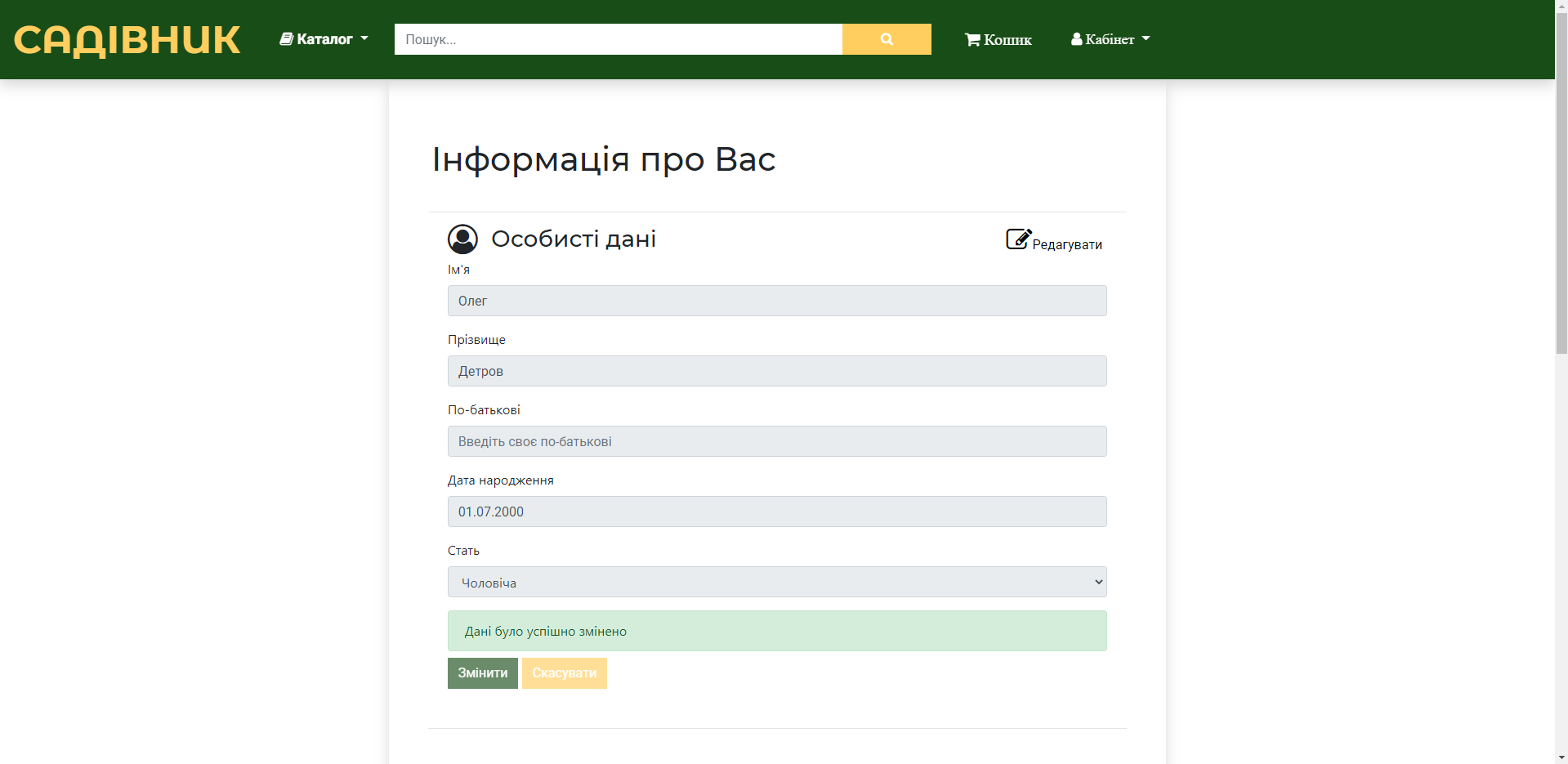


Рис. 3.31. Вигляд форми після успішної зміни персональних даних

## Тестування роботи програмного забезпечення

Настав час для тестування. Для цього будемо використовувати метод функціонального тестування: виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог. В ТЗ (див. ст. 9) було зазначено список функцій, які мають працювати в проекті:

* Можливість знайти товар
* Можливість додати товар у корзину
* Можливість змінювати кількість замовленого товару
* Можливість замовити товар
* Змінювати особисті дані
* Редагувати записи, пов’язані з товарами (для адміністратора)
* Додавати записи, пов’язані з товарами (для адміністратора)
* Видаляти записи, пов’язані з товарами (для адміністратора)

Таблиця 3.1

**Тестування програмного додатку**

|  |  |  |
| --- | --- | --- |
| № | Функція | Успішність |
| 1 | Можливість знайти товар | + |
| 2 | Можливість додати товар у корзину | + |
| 3 | Можливість змінювати кількість замовленого товару | + |
| 4 | Можливість замовити товар | + |
| 5 | Змінювати особисті дані | + |
| 6 | Редагувати записи, пов’язані з товарами (для адміністратора) | + |
| 7 | Додавати записи, пов’язані з товарами (для адміністратора) | + |
| 8 | Видаляти записи, пов’язані з товарами (для адміністратора) | + |

# ВИСНОВКИ

В цьому курсовому проекті було досліджено алгоритм створення інтернет-магазину садівничих товарів. Інтернет-магазин – веб-ресурс, який дозволяє користувачу замовляти товари, а також має переваги для продавця, адже відкриває нові можливості для продажу товарів.

Уся робота над проектом розділялась на декілька частин:

* Визначення основного функціоналу, який буде присутній у веб-сайті
* Розподілення основного функціоналу на менші частини задля структурованого процесу розробки
* Створення схеми бази даних та визначення сутностей, присутніх у ній
* Написання системи керування вмістом
* Написання серверної частини веб-сайту
* Написання зовнішнього вигляду веб-сайту
* Відлагодження кожної з частин проекту

У першому розділі визначені підстави для розробки курсового проекту, більш детально описані основні поняття.

У другому розділі була детально спроектована основна схема функціонування веб-сайту. Розроблено базу даних проекту, систему керування вмістом, зовнішній вигляд та функціонал інтернет-магазину.

У третьому розділі було проведено тестування, та виправленні деякі помилки які були виявлені у проекті, а також детально дана інструкція користування програмним продуктом.

Проект реалізований з використанням платформи Node.JS, Javascript, HTML5, препроцесору Sass.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Stack Overflow [Електронний ресурс] – Режим доступу до ресурсу: <https://stackoverflow.com/>.
2. Метаніт. Сайт про програмування. [Електронний ресурс] – Режим доступу до ресурсу: <https://metanit.com/>.
3. Proglib. Медиакит [Електронний ресурс] – Режим доступу до ресурсу: <https://proglib.io/>.
4. Tproger [Електронний ресурс] – Режим доступу до ресурсу: <https://tproger.ru/>
5. Passport. Офіційна документація. [Електронний ресурс] – Режим доступу до ресурсу: <http://www.passportjs.org/>.
6. Sequelize. Офіційна документація. [Електронний ресурс] – Режим доступу до ресурсу: https://sequelize.org/.

# ДОДАТКИ

## Додаток А

Лістинг adminpanel.hbs:

<div *class*="shadow d-flex flex-lg-row flex-column m-5">  
 <div *class*="container col-lg-3 col-11 p-3" >  
 <h1 *class*="my-5">Admin Panel</h1>  
 <hr>  
 <h2>Sequelize tables</h2>  
 <div *class*="list-group">  
 {{#*each tables*}}  
 <a *href*="/admin/panel/resources/{{*this*.*link*}}/1" *class*="list-group-item list-group-item-action">{{*this*.*tableName*}}</a>  
 {{/*each*}}  
 </div>  
 <a *href*="/admin/panel/logout" *role*="button" *class*="btn btn-primary px-3 my-4 w-100">Logout</a>  
 </div>  
 <div *class*="gray-background container col-lg-9 col-11">  
 {{#*if isCreation*}}  
 {{{*creationFormHbs*}}}  
 {{*else if isEditing*}}  
 {{{*editingFormHbs*}}}  
 {{*else*}}  
 <div *id*="table-content-container">  
 <h2 *class*="p-3">{{*tableName*}}</h2>  
 <div *class*="d-flex justify-content-end">  
 <div *class*="mx-3 align-middle">Total records: <span *id*="total-records-span">{{*totalRecords*}}</span></div>  
 <div *class*="mx-3">  
 <a *href*="/admin/panel/create/{{*tableNameForLink*}}" *role*="button" *class*="btn btn-primary px-3">+ Create New</a>  
 </div>  
 </div>  
 <div *class*="container">  
 {{#*isHaveContent totalRecords*}}  
 <table *class*="mt-5" *id*="displayTable">  
 <thead *class*="gray-background">  
 <tr>  
 {{#*each object*}}  
 <td *id*="th{{*@index*}}" *class*="px-3">  
 {{*@key*}}  
 </td>  
 {{/*each*}}  
 <td *id*="thedit">Edit</td>  
 <td *id*="thdelete">Delete</td>  
 </tr>  
 </thead>  
 <tbody>  
 {{#*each entities*}}  
 <tr>  
 {{#*each this*}}  
 <td *headers*="th{{*@index*}}" *class*="px-3">{{*this*}}</td>  
 {{/*each*}}  
 {{#*each this*}}  
 {{#*isId @key*}}  
 <td *headers*="thedit">  
 <a *role*="button" *class*="btn btn-outline-dark mx-1" *href*="/admin/panel/edit/{{*..*/*..*/*tableNameForLink*}}/{{*this*}}" ><i *class*="fa fa-pencil fa-lg"></i></a>  
 </td>  
 <td *headers*="thdelete">  
 <a *role*="button" *class*="btn btn-outline-dark mx-1" *onclick*="deleteRecord('/admin/panel/delete/{{*..*/*..*/*tableNameForLink*}}/{{*this*}}')"><i *class*="fa fa-trash-o fa-lg"></i></a>  
 </td>  
 {{/*isId*}}  
 {{/*each*}}  
 </tr>  
 {{/*each*}}  
 </tbody>  
 </table>  
 {{> *alertcontainer*}}  
 <ul *class*="pagination my-3 mx-3">  
 <li *class*="page-item" *id*="previous-pagination-link-container">  
 <a *href*="" *class*="page-link" *id*="previous-pagination-link">Previous</a>  
 </li>  
 {{#*each numberOfPages*}}  
 <li *class*="page-item">  
 <a *href*="/admin/panel/resources/{{*..*/*tableNameForLink*}}/{{*this*}}" *class*="page-link class-for-activation">{{*this*}}</a>  
 </li>  
 {{/*each*}}  
 <li *class*="page-item" *id*="next-pagination-link-container">  
 <a *href*="" *class*="page-link" *id*="next-pagination-link">Next</a>  
 </li>  
 </ul>  
 {{*else*}}  
 <div *class*="d-flex justify-content-center align-items-center flex-column">  
 <h2 *class*="my-2">No records in this table, create the first one!</h2>  
 <div *class*="mx-3">  
 <a *href*="/admin/panel/create/{{*tableNameForLink*}}" *role*="button" *class*="btn btn-primary px-5 py-2">+ Create New</a>  
 </div>  
 </div>  
 {{/*isHaveContent*}}  
  
 </div>  
 {{/*if*}}  
 <span *id*="table-name-for-link-span" *style*="display: none">{{*tableNameForLink*}}</span>  
 <span *id*="max-page-span" *style*="display: none">{{*maxPage*}}</span>  
 <span *id*="is-visible-span" *style*="display: none">{{*isRightSideOfPanelVisible*}}</span>  
 </div>  
 </div>  
</div>

## Додаток Б

Лістинг adminpanelcontroller.js

*var* fs = require('fs');  
*var* Handlebars = require('handlebars');  
*const* csrf = require('csurf');  
  
*var* csrfProtection = *new* csrf();  
  
*const* tableLink = require('../../models/helpers/tablelink');  
*const* numberOfRecordsOnPage = 10;  
  
*var* tables = [];  
*var* pagesNumbers = [];  
*var* adminsEntities = [];  
*var* categoryEntities = [];  
*var* manufacturerEntities = [];  
*var* generalProductsEntities = [];  
*var* subProductsEntities = [];  
  
*var* adminEntitiesCount = 0;  
*var* categoryEntitiesCount = 0;  
*var* manufacturerEntitiesCount = 0;  
  
*const* scripts = [{script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/adminpaneljs.js'}];  
*const* mainScripts = [{script: '/javascripts/mainadminpaneljs.js'}];  
  
*var* currentPageIndex = 0;  
  
*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* Admin = require('../../models/admin/admin')(Sequelize, sequelize);  
*const* Category = require('../../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../../models/product/subproduct')(Sequelize, sequelize);  
  
setTablesLinks();  
exports.returnAdminPanel = *function*(request, response) {  
 response.render('admin/adminpanel', {  
 title: 'Admin panel',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'false',  
 scripts: mainScripts  
 })  
};  
  
exports.returnAdminPanelAdminEntities = *function*(request, response, newPageNumber) {  
 currentPageIndex = newPageNumber - 1;  
 Admin.findAll({raw: *true*, attributes: {exclude: ['Password']}, offset: currentPageIndex \* numberOfRecordsOnPage, limit: numberOfRecordsOnPage}).then(admins => {  
 adminsEntities = admins;  
 Admin.count().then(recordsCount => {  
 adminEntitiesCount = recordsCount;  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - Admins',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'Admins',  
 tableNameForLink: 'admin',  
 totalRecords: adminEntitiesCount,  
 object: adminsEntities[0],  
 entities: adminsEntities,  
 numberOfPages: calculateNumberOfPagesForPanel(adminEntitiesCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 });  
 }).catch(err => console.log(err));  
};  
  
exports.returnAdminPanelCategoryEntities = *function*(request, response, newPageNumber) {  
 currentPageIndex = newPageNumber - 1;  
 Category.findAll({raw: *true*, offset: currentPageIndex \* numberOfRecordsOnPage, limit: numberOfRecordsOnPage}).then(categories => {  
 categoryEntities = categories;  
 Category.count().then(recordsCount => {  
 categoryEntitiesCount = recordsCount;  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - Category',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'Categories',  
 tableNameForLink: 'category',  
 totalRecords: categoryEntitiesCount,  
 object: categoryEntities[0],  
 entities: categoryEntities,  
 numberOfPages: calculateNumberOfPagesForPanel(categoryEntitiesCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 }  
 );  
  
 }).catch(err => console.log(err));  
  
};  
  
exports.returnAdminPanelManufacturerEntities = *function*(request, response, newPageNumber) {  
 currentPageIndex = newPageNumber - 1;  
 Manufacturer.findAll({raw: *true*, offset: currentPageIndex \* numberOfRecordsOnPage, limit: numberOfRecordsOnPage}).then(manufacturers => {  
 manufacturerEntities = manufacturers;  
 Manufacturer.count().then(recordsCount => {  
 manufacturerEntitiesCount = recordsCount;  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - Manufacturer',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'Manufacturers',  
 tableNameForLink: 'manufacturer',  
 totalRecords: manufacturerEntitiesCount,  
 object: manufacturerEntities[0],  
 entities: manufacturerEntities,  
 numberOfPages: calculateNumberOfPagesForPanel(manufacturerEntitiesCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 }  
 )  
  
 }).catch(err => console.log(err));  
};  
  
exports.returnAdminPanelGeneralProductsEntities = *function* (request, response, newPageNumber) {  
 currentPageIndex = newPageNumber - 1;  
 GeneralProduct.findAll({raw: *true*,  
 offset: currentPageIndex \* numberOfRecordsOnPage,  
 limit: numberOfRecordsOnPage,  
 attributes: {exclude: ['createdAt', 'updatedAt', 'Recommend\_Percentage', 'Average\_Rating']},  
 }).then(generalProducts => {  
 *let* counter = 0;  
 *if* (generalProducts.length === 0) {  
 generalProductsEntities = generalProducts;  
 GeneralProduct.count().then(recordsCount => {  
 currentPageIndex = newPageNumber - 1;  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - General product',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'General products',  
 tableNameForLink: 'generalproduct',  
 totalRecords: recordsCount,  
 object: generalProductsEntities[0],  
 entities: generalProductsEntities,  
 numberOfPages: calculateNumberOfPagesForPanel(recordsCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 }  
 )  
 }  
 *else* {  
 generalProducts.forEach(generalProduct => {  
 Category.findOne({where: {Category\_ID: generalProduct.Category\_ID\_FK}}).then(category => {  
 *if* (category)  
 generalProduct.Category\_ID\_FK = category.Category\_Name;  
 *else* generalProduct.Category\_ID\_FK = 'No category selected';  
 });  
 Manufacturer.findOne({where: {Manufacturer\_ID: generalProduct.Manufacturer\_ID\_FK}}).then(manufacturer => {  
 *if* (manufacturer)  
 generalProduct.Manufacturer\_ID\_FK = manufacturer.Manufacturer\_Name;  
 *else* generalProduct.Manufacturer\_ID\_FK = 'No manufacturer selected';  
 counter++;  
 *if* (counter === generalProducts.length) {  
 generalProductsEntities = generalProducts;  
 GeneralProduct.count().then(recordsCount => {  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - General product',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'General products',  
 tableNameForLink: 'generalproduct',  
 totalRecords: recordsCount,  
 object: generalProductsEntities[0],  
 entities: generalProductsEntities,  
 numberOfPages: calculateNumberOfPagesForPanel(recordsCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 }  
 )  
 }  
 })  
 });  
 }  
 }).catch(err => console.log(err));  
};  
  
exports.returnAdminPanelSubProductEntities = *function* (request, response, newPageNumber) {  
 currentPageIndex = newPageNumber - 1;  
 SubProduct.findAll({raw: *true*,  
 offset: currentPageIndex \* numberOfRecordsOnPage,  
 limit: numberOfRecordsOnPage,  
 attributes: {exclude: ['createdAt', 'updatedAt']}}).then(subProducts => {  
 *let* counter = 0;  
 *if* (subProducts.length === 0) {  
 subProductsEntities = subProducts;  
 SubProduct.count().then(recordsCount => {  
 currentPageIndex = newPageNumber - 1;  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - Sub product',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'Sub products',  
 tableNameForLink: 'subproduct',  
 totalRecords: recordsCount,  
 object: subProducts[0],  
 entities: subProducts,  
 numberOfPages: calculateNumberOfPagesForPanel(recordsCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 })  
 }  
 *else* {  
 subProducts.forEach(subProduct => {  
 GeneralProduct.findOne({where: {General\_Product\_ID: subProduct.General\_Product\_ID\_FK}}).then(generalProduct => {  
 subProduct.General\_Product\_ID\_FK = generalProduct.Product\_Name;  
 counter++;  
 *if* (counter === subProducts.length) {  
 subProductsEntities = subProducts;  
 SubProduct.count().then(recordsCount => {  
 response.render('admin/adminpanel', {  
 title: 'Admin panel - Sub product',  
 layout: 'adminlayout',  
 tables: tables,  
 tableName: 'Sub products',  
 tableNameForLink: 'subproduct',  
 totalRecords: recordsCount,  
 object: subProducts[0],  
 entities: subProducts,  
 numberOfPages: calculateNumberOfPagesForPanel(recordsCount),  
 maxPage: pagesNumbers[pagesNumbers.length - 1],  
 isRightSideOfPanelVisible: 'true',  
 scripts: scripts  
 })  
 })  
 }  
 })  
 })  
 }  
})  
};  
  
*function* setTablesLinks() {  
 tables.push(*new* tableLink('admin', 'Admin'));  
 tables.push(*new* tableLink('category', 'Category'));  
 tables.push(*new* tableLink('manufacturer', 'Manufacturer'));  
 tables.push(*new* tableLink('generalproduct', 'General product'));  
 tables.push(*new* tableLink('subproduct', 'Sub product'));  
}  
  
*function* calculateNumberOfPagesForPanel(numberOfRecords) {  
 createNewPagingArray(numberOfRecords);  
 *if* (currentPageIndex < 5)  
 *return* pagesNumbers.slice(0, numberOfRecordsOnPage);  
 *return* pagesNumbers.slice(currentPageIndex - 5, currentPageIndex + 4);  
}  
  
*function* createNewPagingArray(numberOfRecords) {  
 *let* totalNumberOfPages = calculateNumberOfPages(numberOfRecords);  
 pagesNumbers = [];  
 *for* (*let* i = 1; i <= totalNumberOfPages; i++) {  
 pagesNumbers.push(i);  
 }  
}  
  
*function* calculateNumberOfPages(numberOfRecords) {  
 *return* Math.ceil(numberOfRecords / numberOfRecordsOnPage);  
}  
  
exports.returnAdminCreationPage = *function*(request, response) {  
 *let* creationPageHtml = getHtmlFromFile('views/admin/create/admincreate.hbs');  
 creationPageHtml = creationPageHtml.replace('{csrfToken}', request.csrfToken());  
 creationPageHtml = creationPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Admin Creation',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/create/admincreationjs.js'}],  
 isCreation: *true*,  
 creationFormHbs: creationPageHtml  
 })  
};  
  
exports.returnCategoryCreationPage = *function*(request, response) {  
 *let* creationPageHtml = getHtmlFromFile('views/admin/create/categorycreate.hbs');  
 creationPageHtml = creationPageHtml.replace('{csrfToken}', request.csrfToken());  
 creationPageHtml = creationPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Category Creation',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/create/categorycreationjs.js'}],  
 isCreation: *true*,  
 creationFormHbs: creationPageHtml  
 })  
};  
  
exports.returnManufacturerCreationPage = *function*(request, response) {  
 *let* creationPageHtml = getHtmlFromFile('views/admin/create/manufacturercreate.hbs');  
 creationPageHtml = creationPageHtml.replace('{csrfToken}', request.csrfToken());  
 creationPageHtml = creationPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Manufacturer Creation',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/create/manufacturercreationjs.js'}],  
 isCreation: *true*,  
 creationFormHbs: creationPageHtml  
 })  
};  
  
exports.returnGeneralProductCreationPage = *function* (request, response) {  
 *let* manufacturerOptions;  
 *let* categoryOptions;  
 Manufacturer.findAll({raw: *true*}).then(manufacturers => {  
 manufacturers.forEach(manufacturer => manufacturerOptions += `<option>${manufacturer.Manufacturer\_Name}</option>`);  
 Category.findAll({raw: *true*}).then(categories => {  
 categories.forEach(category => categoryOptions += `<option>${category.Category\_Name}</option>`);  
 *let* creationPageHtml = getHtmlFromFile('views/admin/create/generalproductcreate.hbs');  
 creationPageHtml = creationPageHtml.replace('{csrfToken}', request.csrfToken());  
 creationPageHtml = creationPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 creationPageHtml = creationPageHtml.replace('{manufactureroptions}', manufacturerOptions);  
 creationPageHtml = creationPageHtml.replace('{categoryoptions}', categoryOptions);  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - General Product Creation',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/create/generalproductcreationjs.js'}],  
 isCreation: *true*,  
 creationFormHbs: creationPageHtml  
 })  
 })  
 })  
};  
  
exports.returnSubProductCreationPage = *function* (request, response) {  
 *let* generalProductOptions;  
 GeneralProduct.findAll({raw: *true*}).then(generalProducts => {  
 *let* counter = 0;  
 *if* (generalProducts.length === 0) {  
 *let* creationPageHtml = getHtmlFromFile('views/admin/create/subproductcreate.hbs');  
 creationPageHtml = creationPageHtml.replace('{csrfToken}', request.csrfToken());  
 creationPageHtml = creationPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 creationPageHtml = creationPageHtml.replace('{generalproductoptions}', 'No general products');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Manufacturer Creation',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/create/subproductcreationjs.js'}],  
 isCreation: *true*,  
 creationFormHbs: creationPageHtml  
 })  
 }  
 *else* {  
 generalProducts.forEach(generalProduct => {  
 generalProductOptions += `<option>${generalProduct.Product\_Name}</option>`;  
 counter++;  
 *if* (counter === generalProducts.length) {  
 *let* creationPageHtml = getHtmlFromFile('views/admin/create/subproductcreate.hbs');  
 creationPageHtml = creationPageHtml.replace('{csrfToken}', request.csrfToken());  
 creationPageHtml = creationPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 creationPageHtml = creationPageHtml.replace('{generalproductoptions}', generalProductOptions);  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Manufacturer Creation',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/create/subproductcreationjs.js'}],  
 isCreation: *true*,  
 creationFormHbs: creationPageHtml  
 })  
 }  
 })  
 }  
 })  
};  
  
exports.returnAdminEditingPage = *function*(request, response, recordId) {  
 Admin.findByPk(recordId).then(result => {  
 *let* editPageHtml = getHtmlFromFile('views/admin/edit/adminedit.hbs');  
 editPageHtml = editPageHtml.replace('{csrfToken}', request.csrfToken);  
 editPageHtml = editPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 editPageHtml = editPageHtml.replace('{login}', result.Login);  
 editPageHtml = editPageHtml.replace('{requestPath}', request.originalUrl + '/submit');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Admin Editing',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/edit/admineditjs.js'}],  
 isEditing: *true*,  
 editingFormHbs: editPageHtml  
 })  
 });  
};  
  
exports.returnCategoryEditingPage = *function*(request, response, recordId) {  
 Category.findByPk(recordId).then(result => {  
 *let* editPageHtml = getHtmlFromFile('views/admin/edit/categoryedit.hbs');  
 editPageHtml = editPageHtml.replace('{csrfToken}', request.csrfToken);  
 editPageHtml = editPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 editPageHtml = editPageHtml.replace('{category}', result.Category\_Name);  
 editPageHtml = editPageHtml.replace('{requestPath}', request.originalUrl + '/submit');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Admin Editing',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/edit/categoryeditjs.js'}],  
 isEditing: *true*,  
 editingFormHbs: editPageHtml  
 })  
 });  
};  
  
exports.returnManufacturerEditingPage = *function*(request, response, recordId) {  
 Manufacturer.findByPk(recordId).then(result => {  
 *let* editPageHtml = getHtmlFromFile('views/admin/edit/manufactureredit.hbs');  
 editPageHtml = editPageHtml.replace('{csrfToken}', request.csrfToken);  
 editPageHtml = editPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 editPageHtml = editPageHtml.replace('{manufacturerName}', result.Manufacturer\_Name);  
 editPageHtml = editPageHtml.replace('{manufacturerLogo}', result.Manufacturer\_Logo\_Path);  
 editPageHtml = editPageHtml.replace('{requestPath}', request.originalUrl + '/submit');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Admin Editing',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/edit/manufacturereditjs.js'}],  
 isEditing: *true*,  
 editingFormHbs: editPageHtml  
 })  
 });  
};  
  
exports.returnGeneralProductEditingPage = *function*(request, response, recordId) {  
 GeneralProduct.findByPk(recordId).then(result => {  
 *let* currentCategory;  
 *let* currentManufacturer;  
 *let* categoryOptions;  
 *let* manufacturerOptions;  
 *let* generalProduct = result;  
 Category.findOne({where: {Category\_ID: result.Category\_ID\_FK}}).then(category => {  
 *if* (category)  
 currentCategory = category.Category\_Name;  
 *else* currentCategory = 'No category selected';  
 Manufacturer.findOne({where: {Manufacturer\_ID: generalProduct.Manufacturer\_ID\_FK}}).then(manufacturer => {  
 *if* (manufacturer)  
 currentManufacturer = manufacturer.Manufacturer\_Name;  
 *else* currentManufacturer = 'No manufacturer selected';  
 Category.findAll().then(categories => {  
 categories.forEach(category => categoryOptions += `<option>${category.Category\_Name}</option>`);  
 Manufacturer.findAll().then(manufacturers => {  
 manufacturers.forEach(manufacturer => manufacturerOptions += `<option>${manufacturer.Manufacturer\_Name}</option>`);  
 *let* editPageHtml = getHtmlFromFile('views/admin/edit/generalproductedit.hbs');  
 editPageHtml = editPageHtml.replace('{csrfToken}', request.csrfToken);  
 editPageHtml = editPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 editPageHtml = editPageHtml.replace('{productName}', generalProduct.Product\_Name);  
 editPageHtml = editPageHtml.replace('{currentCategory}', currentCategory);  
 editPageHtml = editPageHtml.replace('{categoryOptions}', categoryOptions);  
 editPageHtml = editPageHtml.replace('{currentManufacturer}', currentManufacturer);  
 editPageHtml = editPageHtml.replace('{manufacturerOptions}', manufacturerOptions);  
 editPageHtml = editPageHtml.replace('{productImage}', generalProduct.General\_Product\_Image\_Path);  
 editPageHtml = editPageHtml.replace('{productDescription}', generalProduct.General\_Product\_Description);  
 editPageHtml = editPageHtml.replace('{requestPath}', request.originalUrl + '/submit');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Product Editing',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/edit/generalproducteditjs.js'}],  
 isEditing: *true*,  
 editingFormHbs: editPageHtml  
 })  
 })  
 })  
 })  
 });  
  
 });  
};  
  
exports.returnSubProductEditingPage = *function* (request, response, recordId) {  
 SubProduct.findByPk(recordId).then(result => {  
 *let* currentGeneralProduct;  
 *let* generalProductOptions;  
 *let* subProduct = result;  
 GeneralProduct.findOne({where: {General\_Product\_ID: subProduct.General\_Product\_ID\_FK}}).then(generalProduct => {  
 currentGeneralProduct = generalProduct.Product\_Name;  
 GeneralProduct.findAll().then(generalProducts => {  
 *if* (generalProducts.length === 0) {  
 *let* editPageHtml = getHtmlFromFile('views/admin/edit/subproductedit.hbs');  
 editPageHtml = editPageHtml.replace('{csrfToken}', request.csrfToken);  
 editPageHtml = editPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 editPageHtml = editPageHtml.replace('{currentGeneralProduct}', currentGeneralProduct);  
 editPageHtml = editPageHtml.replace('{generalproductoptions}', 'No general products');  
 editPageHtml = editPageHtml.replace('{subproductcode}', subProduct.Sub\_Product\_Code);  
 editPageHtml = editPageHtml.replace('{subproductprice}', subProduct.Price);  
 editPageHtml = editPageHtml.replace('{subproductdescription}', subProduct.Sub\_Product\_Description);  
 editPageHtml = editPageHtml.replace('{requestPath}', request.originalUrl + '/submit');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Sub Product Editing',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/edit/subproducteditjs.js'}],  
 isEditing: *true*,  
 editingFormHbs: editPageHtml  
 })  
 }  
 *else* {  
 *let* counter = 0;  
 generalProducts.forEach(generalProduct => {  
 generalProductOptions += `<option>${generalProduct.Product\_Name}</option>`;  
 counter++;  
 *if* (counter === generalProducts.length) {  
 *let* editPageHtml = getHtmlFromFile('views/admin/edit/subproductedit.hbs');  
 editPageHtml = editPageHtml.replace('{csrfToken}', request.csrfToken);  
 editPageHtml = editPageHtml.replace('{pageNumber}', currentPageIndex + 1);  
 editPageHtml = editPageHtml.replace('{currentGeneralProduct}', currentGeneralProduct);  
 editPageHtml = editPageHtml.replace('{generalproductoptions}', generalProductOptions);  
 editPageHtml = editPageHtml.replace('{subproductcode}', subProduct.Sub\_Product\_Code);  
 editPageHtml = editPageHtml.replace('{subproductprice}', subProduct.Price);  
 editPageHtml = editPageHtml.replace('{subproductdescription}', subProduct.Sub\_Product\_Description);  
 editPageHtml = editPageHtml.replace('{requestPath}', request.originalUrl + '/submit');  
 response.render('admin/adminpanel', {  
 title: 'Admin Panel - Sub Product Editing',  
 layout: 'adminlayout',  
 tables: tables,  
 isRightSideOfPanelVisible: 'true',  
 scripts: [{script: '/javascripts/mainadminpaneljs.js'}, {script: '/javascripts/helpers/alertContainer.js'}, {script: '/javascripts/admin/edit/subproducteditjs.js'}],  
 isEditing: *true*,  
 editingFormHbs: editPageHtml  
 })  
 }  
 })  
 }  
 })  
 })  
 })  
};  
  
*function* getHtmlFromFile(filePath) {  
 *let* fileHtml = fs.readFileSync(filePath, *function*(err, data) {  
 *if* (!err) {  
 }  
 *else* {  
 console.log(err);  
 }  
 });  
 *let* source = fileHtml.toString();  
 *let* template = Handlebars.compile(source);  
 *return* template(fileHtml);  
}  
  
exports.logoutAdmin = *function* (request, response) {  
 *if* (request.user && 'Admin\_ID' *in* request.user) {  
 request.logout();  
 response.redirect('/');  
 }  
 *else* {  
 response.redirect('/admin')  
 }  
};

## Додаток В

Лістинг app.js:

*var* createError = require('http-errors');  
*var* express = require('express');  
*var* expressHbs = require('express-handlebars');  
*var* session = require('express-session');  
*var* passport = require('passport');  
*var* flash = require('connect-flash');  
*var* bodyParser = require('body-parser');  
*var* path = require('path');  
*var* cookieParser = require('cookie-parser');  
*var* logger = require('morgan');  
*var* sassMiddleware = require('node-sass-middleware');  
*var* fs = require('fs');  
*const* Handlebars = require('handlebars');  
*const* {allowInsecurePrototypeAccess} = require('@handlebars/allow-prototype-access');  
  
*var* app = express();  
  
*// controllers  
const* contactsController = require('./controllers/contactsController.js');  
*const* homeController = require('./controllers/homeController.js');  
*const* categoriesController = require('./controllers/categoriesController.js');  
*const* productController = require('./controllers/productController.js');  
*const* searchController = require('./controllers/searchController.js');  
*const* cartController = require('./controllers/cartController.js');  
*const* orderController = require('./controllers/orderController.js');  
  
*// routers  
const* adminRouter = require('./routes/admin/adminrouter');  
*const* userRouter = require('./routes/userrouter');  
require('./config/passport');  
  
*// view engine setup*app.set('views', path.join(\_\_dirname, 'views'));  
  
app.engine('hbs', expressHbs(  
 {  
 layoutsDir: 'views/layouts',  
 defaultLayout: 'layout',  
 extname: 'hbs',  
 handlebars: allowInsecurePrototypeAccess(Handlebars),  
 helpers: {  
 isId: *function*(key, options) {  
 console.log(key);  
 *if* (key === 'Admin\_ID' || key === 'Category\_ID' || key === 'Manufacturer\_ID' || key === 'General\_Product\_ID' || key === 'Sub\_Product\_ID')  
 *return* options.fn(*this*);  
 *else  
 return* options.inverse(*this*);  
 },  
 isHaveContent: *function*(length, options) {  
 *if* (parseInt(length) > 0)  
 *return* options.fn(*this*);  
 *else  
 return* options.inverse(*this*);  
 }  
 }  
 }  
));  
  
*var* SequelizeStore = require('connect-session-sequelize')(session.Store);  
*const* { sequelize, Sequelize } = require('./models/sequelize');  
*const* Category = require('./models/product/category')(Sequelize, sequelize);  
*const* SubProduct = require('./models/product/subproduct')(Sequelize, sequelize);  
*const* GeneralProduct = require('./models/product/generalproduct')(Sequelize, sequelize);  
*var* SessionSequelize = *new* sequelize(  
 'sessiondatabase',  
 'username',  
 'password',  
 {  
 "dialect": "sqlite",  
 "storage": "./session.sqlite"  
 }  
);  
  
*var* sessionStore = *new* SequelizeStore({  
 db: SessionSequelize,  
 checkExpirationInterval: 60 \* 60 \* 1000,  
 expiration: 24 \* 60 \* 60 \* 1000  
});  
  
app.set('view engine', 'hbs');  
app.use(logger('dev'));  
app.use(express.json());  
app.use(express.urlencoded({ extended: *false* }));  
app.use(bodyParser.urlencoded({ extended: *true* }));  
app.use(bodyParser.json());  
app.use(cookieParser());  
app.use(session({secret: 'adminsecret',  
 resave: *false*,  
 saveUninitialized: *false*,  
 store: sessionStore  
}));  
sessionStore.sync();  
app.use(flash());  
app.use(passport.initialize());  
app.use(passport.session());  
app.use(sassMiddleware({  
 src: path.join(\_\_dirname, 'public'),  
 dest: path.join(\_\_dirname, 'public'),  
 indentedSyntax: *true*, *// true = .sass and false = .scss* sourceMap: *true*}));  
app.use(express.static(path.join(\_\_dirname, 'public')));  
  
*const* csrf = require('csurf');  
  
*var* csrfProtection = *new* csrf();  
  
app.use(csrfProtection);  
  
app.use(*function* (request, response, next) {  
 response.locals.login = request.isAuthenticated();  
 response.locals.session = request.session;  
 Category.findAll().then(categories => {  
 response.locals.categories = categories;  
 next();  
 });  
});  
app.get('/trytoorder', orderController.tryToRedirectToOrderPage);  
app.get('/order', isLoggedIn, orderController.returnOrderPage);  
app.post('/placeorder', orderController.placeOrder);  
app.use('/user', userRouter);  
app.use('/admin', adminRouter);  
app.get('/addtocart/:id', cartController.addProductToCart);  
app.get('/reduce/:id', cartController.reduceProductQty);  
app.get('/remove/:id', cartController.removeProduct);  
app.get('/cart', cartController.returnCartPage);  
app.get('/products/filter', searchController.returnFilteredProducts);  
app.get('/search', searchController.returnProductsByName);  
app.use('/product/:productId', *function* (request, response) {  
 productController.returnProductPage(request, response, request.params.productId);  
});  
app.use('/categories/:categoryId/filter', *function* (request, response) {  
 categoriesController.returnFilteredProducts(request, response, request.params.categoryId)  
});  
app.use('/categories/:categoryId', *function*(request, response) {  
 *let* categoryId = request.params.categoryId;  
 categoriesController.returnCategoryProductsPage(request, response, categoryId);  
});  
app.use('/contacts', contactsController.returnContactsPage);  
app.use('/', homeController.returnHomePage);  
  
*function* isLoggedIn(request, response, next) {  
 *if* (request.user) {  
 *let* isUser = 'User\_ID' *in* request.user;  
 *if* (request.isAuthenticated() && isUser) {  
 *return* next();  
 }  
 }  
 response.redirect('/cart');  
}module.exports = app;  
app.listen(3000);

Лістинг admincreatecontroller.js:

*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* Admin = require('../../models/admin/admin')(Sequelize, sequelize);  
*const* Category = require('../../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../../models/product/subproduct')(Sequelize, sequelize);  
  
exports.createNewAdmin = *function*(request, response) {  
 *let* login = request.body.login;  
 *let* password = request.body.password;  
 Admin.findOne({where: {Login: login}}).then(admin => {  
 *if* (admin)  
 response.end('Admin exists');  
 *else* Admin.create({  
 Login: login,  
 Password: password  
 }).then(res => {  
 response.end('Record created');  
 }).catch(err => {  
 console.log(err);  
 response.end('Creation failed');  
 })  
 })  
};  
  
exports.createNewCategory = *function*(request, response) {  
 *let* categoryName = request.body.category;  
 Category.findOne({where: {Category\_Name: categoryName}}).then(category => {  
 *if* (category)  
 response.end('Category exists');  
 *else* Category.create({  
 Category\_Name: categoryName  
 }).then(res => {  
 response.send('Record created');  
 }).catch(err => {  
 console.log(err);  
 response.send('Creation failed');  
 })  
 })  
};  
  
exports.createNewManufacturer = *function*(request, response) {  
 *let* manufacturerName = request.body.manufacturerName;  
 *let* manufacturerLogoURL = request.body.manufacturerLogoURL;  
 Manufacturer.findOne({where: {Manufacturer\_Name: manufacturerName}}).then(manufacturer => {  
 *if* (manufacturer)  
 response.end('Manufacturer exists');  
 *else* {  
 Manufacturer.create({  
 Manufacturer\_Name: manufacturerName,  
 Manufacturer\_Logo\_Path: manufacturerLogoURL  
 }).then(res => {  
 response.send('Record created');  
 }).catch(err => {  
 console.log(err);  
 response.send('Creation failed');  
 })  
 }  
 })  
};  
  
exports.createNewGeneralProduct = *function* (request, response) {  
 *let* productName = request.body.productName;  
 *let* category = request.body.category;  
 *let* manufacturer = request.body.manufacturer;  
 *let* imageURL = request.body.imageURL;  
 *let* description = request.body.description;  
 *let* categoryId;  
 *let* manufacturerId;  
 Category.findOne({where: {Category\_Name: category}}).then(result => {  
 categoryId = result.Category\_ID;  
 Manufacturer.findOne({where: {Manufacturer\_Name: manufacturer}}).then(result => {  
 manufacturerId = result.Manufacturer\_ID;  
 GeneralProduct.create({  
 Product\_Name: productName,  
 General\_Product\_Image\_Path: imageURL,  
 General\_Product\_Description: description,  
 Category\_ID\_FK: categoryId,  
 Manufacturer\_ID\_FK: manufacturerId  
 }).then(result => {  
 response.send('Record created');  
 }).catch(err => {  
 console.log(err);  
 response.send('Creation failed');  
 })  
 })  
 })  
};  
  
exports.createNewSubProduct = *function*(request, response) {  
 *let* generalProductId;  
 GeneralProduct.findOne({where: {Product\_Name: request.body.generalProduct}}).then(generalProduct => {  
 generalProductId = generalProduct.General\_Product\_ID;  
 SubProduct.create({  
 General\_Product\_ID\_FK: generalProductId,  
 Sub\_Product\_Code: request.body.code,  
 Price: request.body.price,  
 Sub\_Product\_Description: request.body.description  
 }).then(result => {  
 response.send('Record created');  
 }).catch(err => {  
 console.log(err);  
 response.send('Creation failed');  
 })  
 })  
};

Лістинг admineditcontroller.js:

*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* Admin = require('../../models/admin/admin')(Sequelize, sequelize);  
*const* Category = require('../../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../../models/product/subproduct')(Sequelize, sequelize);  
  
exports.editAdmin = *function*(request, response, recordId) {  
 *let* login = request.body.login;  
 *let* oldPassword = request.body.oldPassword;  
 *if* (request.body.isChangePassword) {  
 *let* newPassword = request.body.newPassword;  
 Admin.findOne({where: {Admin\_ID: recordId}}).then(admin => {  
 *if* (admin) {  
 *if* (!admin.validatePassword(oldPassword))  
 response.send('Password mismatched');  
 *else* Admin.update({ Login: login, Password: Admin.generatePassword(newPassword)}, {where: {Admin\_ID: recordId}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 })  
 }  
 *else* {  
 Admin.findOne({where: {Admin\_ID: recordId}}).then(admin => {  
 *if* (admin) {  
 *if* (!admin.validatePassword(oldPassword))  
 response.send('Password mismatched');  
 *else* Admin.update({ Login: login}, {where: {Admin\_ID: recordId}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed')  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
};  
  
exports.editCategory = *function*(request, response, recordId) {  
 Category.findOne({where: {Category\_ID: recordId}}).then(category => {  
 *if* (category) {  
 Category.update({Category\_Name: request.body.categoryName}, {where: {Category\_ID: recordId}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
};  
  
exports.editManufacturer = *function*(request, response, recordId) {  
 Manufacturer.findOne({where: {Manufacturer\_ID: recordId}}).then(manufacturer => {  
 *if* (manufacturer) {  
 Manufacturer.update({Manufacturer\_Name: request.body.manufacturerName, Manufacturer\_Logo\_Path: request.body.manufacturerLogo}, {where: {Manufacturer\_ID: recordId}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
};  
  
exports.editGeneralProduct = *function*(request, response, recordId) {  
 *let* generalProductInstance;  
 *let* categoryId;  
 *let* manufacturerId;  
 *let* categoryName = request.body.category;  
 *let* manufacturerName = request.body.manufacturer;  
 *let* productName = request.body.productName;  
 *let* description = request.body.description;  
 *let* imageURL = request.body.imageURL;  
 GeneralProduct.findOne({where: {General\_Product\_ID: recordId}}).then(generalProduct => {  
 generalProductInstance = generalProduct;  
 Category.findOne({where: {Category\_Name: categoryName}}).then(category => {  
 categoryId = category.Category\_ID;  
 Manufacturer.findOne({where: {Manufacturer\_Name: manufacturerName}}).then(manufacturer => {  
 manufacturerId = manufacturer.Manufacturer\_ID;  
 *if* (generalProductInstance) {  
 GeneralProduct.update({Product\_Name: productName, General\_Product\_Description: description, General\_Product\_Image\_Path: imageURL, Category\_ID\_FK: categoryId, Manufacturer\_ID\_FK: manufacturerId}, {where: {General\_Product\_ID: recordId}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 })  
 })  
 })  
};  
  
exports.editSubProduct = *function* (request, response, recordId) {  
 *let* subProductInstance;  
 *let* generalProductId;  
 SubProduct.findByPk(recordId).then(subProduct => {  
 subProductInstance = subProduct;  
 GeneralProduct.findOne({where: {Product\_Name: request.body.generalProduct}}).then(generalProduct => {  
 generalProductId = generalProduct.General\_Product\_ID;  
 *if*(subProductInstance) {  
 SubProduct.update({Sub\_Product\_Description: request.body.description, Sub\_Product\_Code: request.body.code, Price: request.body.price, General\_Product\_ID\_FK: generalProductId}, {where: {Sub\_Product\_ID: recordId}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 })  
 })  
};

Лістинг adminsignincontroller.js:

*const* csrf = require('csurf');  
  
*var* csrfProtection = *new* csrf();  
  
exports.returnAdminSignInPage = *function*(request, response) {  
 response.render('admin/adminsignin', {  
 title: 'Admin Sign-In',  
 layout: 'adminlayout',  
 csrfToken: request.csrfToken()  
 })  
};

Лістинг prodilecontroller.js:

*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* User = require('../../models/user/user')(Sequelize, sequelize);  
*const* GeneralProduct = require('../../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../../models/product/subproduct')(Sequelize, sequelize);  
*const* Order = require('../../models/user/order')(Sequelize, sequelize);  
*const* OrderProduct = require('../../models/user/orderproduct')(Sequelize, sequelize);  
*var* Cart = require('../../models/helpers/cart');  
  
exports.returnProfilePage = *function* (request, response) {  
 *if*('User\_ID' *in* request.user) {  
 *let* userOrders = [];  
 User.findByPk(request.user.User\_ID).then(user => {  
 Order.findAll({where: {User\_ID\_FK: request.user.User\_ID}}).then(orders => {  
 *if* (orders.length > 0) {  
 *let* orderCounter = 0;  
 orders.forEach(order => {  
 *let* orderCart = *new* Cart({});  
 *let* orderForHbs = {};  
 OrderProduct.findAll({where: {Order\_ID\_FK: order.Order\_ID}}).then(orderProducts => {  
 *let* orderProductsCounter = 0;  
 orderProducts.forEach(orderProduct => {  
 SubProduct.findByPk(orderProduct.Sub\_Product\_ID\_FK).then(subProduct => {  
 GeneralProduct.findByPk(subProduct.General\_Product\_ID\_FK).then(generalProduct => {  
 *for* (*let* i = 0; i < orderProduct.Product\_Amount; i++)  
 orderCart.add(subProduct, generalProduct.Product\_Name + ', код: ' + subProduct.Sub\_Product\_Code, subProduct.Sub\_Product\_ID);  
 orderProductsCounter++;  
 *if* (orderProductsCounter === orderProducts.length) {  
  
 *let* options = {  
 year: 'numeric',  
 month: 'numeric',  
 day: 'numeric',  
 hour: '2-digit',  
 minute: '2-digit'  
 };  
 orderForHbs.orderDate = *new* Date(Date.parse(order.Order\_Datetime)).toLocaleString('uk', options);;  
 orderForHbs.name = order.Name;  
 orderForHbs.lastName = order.Last\_Name;  
 orderForHbs.totalPrice = orderCart.totalPrice;  
 orderForHbs.totalQty = orderCart.totalQty;  
 orderForHbs.products = orderCart.generateArray();  
 userOrders.push(orderForHbs);  
 orderCounter++;  
 *if* (orderCounter === orders.length) {  
 response.render('profile', {  
 title: 'Садівник - Особистий кабінет',  
 user: user,  
 orders: userOrders,  
 csrfToken: request.csrfToken(),  
 scripts: [{script: '/javascripts/profilejs.js'}]  
 })  
 }  
 }  
 })  
 })  
 })  
 })  
 })  
 }  
 *else* {  
 response.render('profile', {  
 title: 'Садівник - Особистий кабінет',  
 user: user,  
 orders: userOrders,  
 csrfToken: request.csrfToken(),  
 scripts: [{script: '/javascripts/profilejs.js'}]  
 })  
 }  
 });  
 });  
 }  
};  
  
exports.changePersonalData = *function* (request, response, userId) {  
 User.findByPk(userId).then(user => {  
 *if* (user) {  
 *let* values = {};  
 values.Name = request.body.name;  
 values.Last\_Name = request.body.lastName;  
 values.Surname = request.body.surname;  
 *if* (request.body.birthDate)  
 values.Birth\_Date = request.body.birthDate;  
 *if* (request.body.sex)  
 values.Sex = request.body.sex;  
 User.update(values, {where: {User\_ID: user.User\_ID}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
};  
  
exports.changeContactsData = *function* (request, response, userId) {  
 User.findByPk(userId).then(user => {  
 *if* (user) {  
 *let* values = {};  
 *if* (request.body.phoneNumber)  
 values.Mobile\_Phone\_Number = request.body.phoneNumber;  
 User.update(values, {where: {User\_ID: user.User\_ID}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
};  
  
exports.changePasswordData = *function* (request, response, userId) {  
 User.findByPk(userId).then(user => {  
 *if* (user) {  
 *let* values = {};  
 *if* (!user.validatePassword(request.body.oldPassword))  
 response.send('Password mismatched');  
 *else* {  
 values.Password = User.generatePassword(request.body.newPassword);  
 User.update(values, {where: {User\_ID: user.User\_ID}}).then(result => {  
 *if* (result)  
 response.send('Record updated');  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
 }  
 }  
 *else* response.send('Update failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Update failed');  
 })  
};

Лістинг signupcontroller.js:

*const* { sequelize, Sequelize } = require('../../models/sequelize');  
  
*const* User = require('../../models/user/user')(Sequelize, sequelize);  
  
exports.tryToRegisterUser = *function* (request, response) {  
 *let* email = request.body.email;  
 *let* name = request.body.name;  
 *let* password = request.body.password;  
 User.findOne({where: {Email: email}}).then(user => {  
 *if* (user)  
 response.end('Email exists');  
 *else* User.create({  
 Email: email,  
 Name: name,  
 Password: password  
 }).then(res => {  
 response.end('Record created');  
 }).catch (err => {  
 console.log(err);  
 response.end('Creation failed');  
 })  
 })  
};

Лістинг cartcontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
*const* GeneralProduct = require('../models/product/generalproduct')(Sequelize, sequelize);  
  
*var* Cart = require('../models/helpers/cart');  
  
exports.addProductToCart = *function* (request, response) {  
 *let* productId = request.params.id;  
 *let* cart = *new* Cart(request.session.cart ? request.session.cart : {});  
 SubProduct.findByPk(productId).then (subProduct => {  
 *if* (subProduct) {  
 GeneralProduct.findOne({where: {General\_Product\_ID: subProduct.General\_Product\_ID\_FK}}).then(generalProduct => {  
 cart.add(subProduct, generalProduct.Product\_Name + ', код: ' + subProduct.Sub\_Product\_Code, subProduct.Sub\_Product\_ID);  
 request.session.cart = cart;  
 request.session.save();  
 console.log(request.session.cart);  
 response.send('Product added');  
 })  
 }  
 *else* response.send('Addition failed');  
 })  
};  
  
exports.reduceProductQty = *function* (request, response) {  
 *let* productId = request.params.id;  
 *let* cart = *new* Cart(request.session.cart ? request.session.cart : {});  
 cart.reduceByOne(productId);  
 request.session.cart = cart;  
 request.session.save();  
 response.send('Reduced');  
};  
  
exports.removeProduct = *function* (request, response) {  
 *let* productId = request.params.id;  
 *let* cart = *new* Cart(request.session.cart ? request.session.cart : {});  
 cart.removeItem(productId);  
 request.session.cart = cart;  
 request.session.save();  
 response.send('Removed');  
};  
  
exports.returnCartPage = *function* (request, response) {  
 *if* (!request.session.cart)  
 *return* response.render('cart', {  
 title: 'Садівник - Корзина',  
 products: *null*,  
 csrfToken: request.csrfToken  
 });  
 *let* cart = *new* Cart(request.session.cart);  
 response.render('cart', {  
 title: 'Садівник - Корзина',  
 products: cart.generateArray(),  
 totalPrice: cart.totalPrice,  
 csrfToken: request.csrfToken,  
 scripts: [{script: '/javascripts/cartjs.js'}]  
 });  
};

Лістинг categoriescontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* { Op } = require('sequelize');  
  
*const* Category = require('../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
  
exports.returnCategoryProductsPage = *function*(request, response, categoryId) {  
 *let* categoryName;  
 *let* manufacturers = [];  
 Category.findByPk(categoryId).then(category => {  
 categoryName = category.Category\_Name;  
 GeneralProduct.findAll({where: {Category\_ID\_FK: categoryId}}).then(generalProducts => {  
 *if* (generalProducts.length > 0) {  
 *let* counter = 0;  
 *for* (*let* i = 0; i < generalProducts.length; i++) {  
 Manufacturer.findOne({where: {Manufacturer\_ID: generalProducts[i].Manufacturer\_ID\_FK}}).then(databaseManufacturer => {  
 *if* (!manufacturers.some(manufacturer => manufacturer.Manufacturer\_ID === databaseManufacturer.Manufacturer\_ID))  
 manufacturers.push(databaseManufacturer);  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: generalProducts[i].General\_Product\_ID}}).then(subProducts => {  
 *let* min = subProducts[0].Price;  
 *for* (*let* j = 0; j < subProducts.length; j++) {  
 *if* (subProducts[j].Price < min)  
 min = subProducts[j].Price;  
 }  
 generalProducts[i].setDataValue('minimumPrice', min);  
 generalProducts[i] = generalProducts[i].get({plain:*true*});  
 counter++;  
 *if* (counter === generalProducts.length) {  
 *let* generalProductChunks = [];  
 *let* chunkSize = 4;  
 *for* (*let* i = 0; i < generalProducts.length; i += chunkSize) {  
 generalProductChunks.push(generalProducts.slice(i, i + chunkSize));  
 }  
 response.render('categoryproducts', {  
 title: 'Садівник - Товари',  
 categoryName: categoryName,  
 products: generalProductChunks,  
 manufacturers: manufacturers,  
 filterURL: '/categories/' + categoryId + '/filter',  
 scripts: [{script: '/javascripts/categoryproductsjs.js'}],  
 csrfToken: request.csrfToken()  
 });  
 }  
 });  
 });  
 }  
 }  
 *else* {  
 response.render('categoryproducts', {  
 title: 'Садівник - Товари',  
 categoryName: categoryName,  
 manufacturers: manufacturers,  
 filterURL: '/categories/' + categoryId + '/filter',  
 scripts: [{script: '/javascripts/categoryproductsjs.js'}],  
 csrfToken: request.csrfToken  
 })  
 }  
  
 })  
 })  
};  
  
exports.returnFilteredProducts = *function* (request, response, categoryId) {  
 *let* manufacturer = request.query.manufacturer;  
 *let* options = {where: {}};  
 *if* (manufacturer)  
 options.where.Manufacturer\_ID\_FK = manufacturer;  
 options.where.Category\_ID\_FK = categoryId;  
 *let* fromPrice = parseFloat(request.query.fromPrice);  
 *let* toPrice = parseFloat(request.query.toPrice);  
 GeneralProduct.findAll(options).then(generalProducts => {  
 *let* counter = 0;  
 generalProducts.forEach(generalProduct => {  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: generalProduct.General\_Product\_ID, Price: {[Op.between]: [fromPrice, toPrice]}}}).then(subProducts => {  
 *if* (subProducts.length !== 0) {  
 *let* min = subProducts[0].Price;  
 *for* (*let* i = 0; i < subProducts.length; i++) {  
 *if* (subProducts[i].Price < min)  
 min = subProducts[i].Price;  
 }  
 generalProduct.setDataValue('minimumPrice', min);  
 generalProduct = generalProduct.get({plain: *true*});  
 counter++;  
 }  
 *else* {  
 *let* index = generalProducts.indexOf(generalProduct);  
 generalProducts.splice(index, 1);  
 }  
 *if* (counter === generalProducts.length)  
 response.send(JSON.stringify(generalProducts));  
 })  
 });  
 });  
};

Лістинг contactscontroller.js:

*const* scripts = [{script: '/javascripts/contactsjs.js'}];  
*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* Category = require('../models/product/category')(Sequelize, sequelize);  
  
exports.returnContactsPage = *function*(request, response) {  
 response.render('contacts', {  
 title: 'Контакти',  
 scripts: scripts,  
 csrfToken: request.csrfToken()  
 });  
};

Лістинг homecontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
  
  
*const* Category = require('../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
  
exports.returnHomePage = *function*(request, response) {  
 GeneralProduct.findAll({limit: 8, order: [['updatedAt', 'DESC']]}).then(generalProducts => {  
 *if* (generalProducts.length > 0) {  
 *let* counter = 0;  
 *for* (*let* i = 0; i < generalProducts.length; i++) {  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: generalProducts[i].General\_Product\_ID}}).then(subProducts => {  
 *if*(subProducts[0]) {  
 *let* min = subProducts[0].Price;  
 *for* (*let* j = 0; j < subProducts.length; j++) {  
 *if* (subProducts[j].Price < min)  
 min = subProducts[j].Price;  
 }  
 generalProducts[i].setDataValue('minimumPrice', min);  
 generalProducts[i] = generalProducts[i].get({plain: *true*});  
 counter++;  
 }  
 *else* {  
 counter++;  
 }  
 *if* (counter === generalProducts.length) {  
 *let* generalProductChunks = [];  
 *let* chunkSize = 4;  
 *for* (*let* i = 0; i < generalProducts.length; i += chunkSize) {  
 generalProductChunks.push(generalProducts.slice(i, i + chunkSize));  
 }  
 response.render('home', {  
 title: 'Садівник - головна сторінка',  
 products: generalProductChunks,  
 csrfToken: request.csrfToken()  
 })  
 }  
 });  
 }  
 }  
 *else* {  
 response.render('home', {  
 title: 'Садівник - головна сторінка',  
 csrfToken: request.csrfToken()  
 })  
 }  
 });  
};

Лістинг ordercontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
*const* Order = require('../models/user/order')(Sequelize, sequelize);  
*const* OrderProduct = require('../models/user/orderproduct')(Sequelize, sequelize);  
  
*var* Cart = require('../models/helpers/cart');  
  
exports.returnOrderPage = *function* (request, response) {  
 *if* (!request.session.cart)  
 *return* response.redirect('/cart');  
 *let* cart = *new* Cart(request.session.cart);  
 response.render('order', {  
 title: 'Садівник - Оформлення Замовлення',  
 csrfToken: request.csrfToken,  
 user: request.user,  
 products: cart.generateArray(),  
 totalPrice: cart.totalPrice,  
 scripts: [{script: '/javascripts/orderjs.js'}]  
 })  
};  
  
exports.tryToRedirectToOrderPage = *function* (request, response) {  
 *if* (request.user) {  
 *let* isUser = 'User\_ID' *in* request.user;  
 *if* (request.isAuthenticated() && isUser) {  
 response.send('Redirect');  
 }  
 *else* response.send('Not logged');  
 }  
 *else* {  
 request.session.orderUrl = '/order';  
 request.session.save();  
 response.send('Not logged');  
 }  
};  
  
exports.placeOrder = *function* (request, response) {  
 *if* (!request.session.cart)  
 *return* response.send('Redirect cart');  
 *if* (!request.user) {  
 *return* response.send('Log in');  
 }  
 *if* (request.user) {  
 *let* isUser = 'User\_ID' *in* request.user;  
 *if* (request.isAuthenticated() && !isUser) {  
 *return* response.send('Admin');  
 }  
 }  
 *let* cart = *new* Cart(request.session.cart);  
 Order.create({Name: request.body.name, Last\_Name: request.body.lastName, City: request.body.city, Area: request.body.area, Street: request.body.street, House\_Number: request.body.building, Apartment\_Number: request.body.apartment, Mobile\_Phone\_Number: request.body.phone, User\_ID\_FK: request.user.User\_ID }).then(result => {  
 *if* (result) {  
 *let* count = 0;  
 *for* (*let* item *in* cart.items) {  
 OrderProduct.create({Product\_Amount: cart.items[item].qty, Sub\_Product\_ID\_FK: cart.items[item].item.Sub\_Product\_ID, Order\_ID\_FK: result.Order\_ID}).then(result => {  
 *if* (result) {  
 count++;  
 *if* (count === Object.keys(cart.items).length) {  
 request.session.cart = {};  
 request.session.save();  
 response.send('Order placed');  
 }  
 }  
 });  
 }  
 }  
 *else* response.send('Creation failed');  
 }).catch(err => {  
 console.log(err);  
 response.send('Creation failed');  
 })  
};

Лістинг productcontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
  
*const* Manufacturer = require('../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
  
  
exports.returnProductPage = *function* (request, response, productId) {  
 *let* product;  
 *let* subProducts;  
 *let* productManufacturer;  
 GeneralProduct.findByPk(productId).then(generalProduct => {  
 product = generalProduct;  
 Manufacturer.findByPk(generalProduct.Manufacturer\_ID\_FK).then(manufacturer => {  
 productManufacturer = manufacturer;  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: product.General\_Product\_ID}}).then(subProductsFromDatabase => {  
 subProducts = subProductsFromDatabase;  
 response.render('product', {  
 title: 'Садівник - ' + product.Product\_Name,  
 product: product,  
 manufacturer: productManufacturer,  
 subProducts: subProducts,  
 csrfToken: request.csrfToken(),  
 scripts: [{script: '/javascripts/productjs.js'}]  
 })  
 })  
 })  
  
 })  
};

Лістинг searchcontroller.js:

*const* { sequelize, Sequelize } = require('../models/sequelize');  
*const* { Op } = require('sequelize');  
  
*const* Category = require('../models/product/category')(Sequelize, sequelize);  
*const* Manufacturer = require('../models/product/manufacturer')(Sequelize, sequelize);  
*const* GeneralProduct = require('../models/product/generalproduct')(Sequelize, sequelize);  
*const* SubProduct = require('../models/product/subproduct')(Sequelize, sequelize);  
  
exports.returnProductsByName = *function* (request, response) {  
 *let* productName = request.query.searchText;  
 *let* manufacturers = [];  
 *let* filteredCategories = [];  
 GeneralProduct.findAll({where: {Product\_Name: {[Op.like]: '%' + productName + '%'}}}).then(generalProducts => {  
 *if* (generalProducts.length > 0) {  
 *let* counter = 0;  
 *for* (*let* i = 0; i < generalProducts.length; i++) {  
 Manufacturer.findOne({where: {Manufacturer\_ID: generalProducts[i].Manufacturer\_ID\_FK}}).then(databaseManufacturer => {  
 *if* (!manufacturers.some(manufacturer => manufacturer.Manufacturer\_ID === databaseManufacturer.Manufacturer\_ID))  
 manufacturers.push(databaseManufacturer);  
 Category.findOne({where: {Category\_ID: generalProducts[i].Category\_ID\_FK}}).then (databaseCategory => {  
 *if* (!filteredCategories.some(category => category.Manufacturer\_ID === databaseCategory.Manufacturer\_ID))  
 filteredCategories.push(databaseCategory);  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: generalProducts[i].General\_Product\_ID}}).then(subProducts => {  
 *let* min = subProducts[0].Price;  
 *for* (*let* i = 0; i < subProducts.length; i++) {  
 *if* (subProducts[i].Price < min)  
 min = subProducts[i].Price;  
 }  
 generalProducts[i].setDataValue('minimumPrice', min);  
 generalProducts[i] = generalProducts[i].get({plain:*true*});  
 counter++;  
 *if* (counter === generalProducts.length) {  
 *let* generalProductChunks = [];  
 *let* chunkSize = 4;  
 *for* (*let* i = 0; i < generalProducts.length; i += chunkSize) {  
 generalProductChunks.push(generalProducts.slice(i, i + chunkSize));  
 }  
 response.render('products', {  
 title: 'Садівник - Пошук за назвою "' + productName + '"',  
 productName: productName,  
 products: generalProductChunks,  
 filteredCategories: filteredCategories,  
 manufacturers: manufacturers,  
 isHaveRecords: *true*,  
 filterURL: '/products/filter',  
 scripts: [{script: '/javascripts/productsjs.js'}],  
 csrfToken: request.csrfToken  
 })  
 }  
 });  
 });  
 });  
 }  
 }  
 *else* {  
 response.render('products', {  
 title: 'Садівник - Пошук за назвою "' + productName + '"',  
 productName: productName,  
 isHaveRecords: *false*,  
 filterURL: '/products/filter',  
 scripts: [{script: '/javascripts/productsjs.js'}],  
 csrfToken: request.csrfToken  
 })  
 }  
 })  
};  
  
exports.returnFilteredProducts = *function* (request, response) {  
 *let* category = request.query.category;  
 *let* manufacturer = request.query.manufacturer;  
 *let* productName = request.query.productName;  
 *let* options = {where: {}};  
 options.where.Product\_Name = {[Op.like]: '%' + productName + '%'};  
 *if* (category)  
 options.where.Category\_ID\_FK = category;  
 *if* (manufacturer)  
 options.where.Manufacturer\_ID\_FK = manufacturer;  
 GeneralProduct.findAll(options).then(generalProducts => {  
 *if* (generalProducts.length > 0) {  
 *let* counter = 0;  
 generalProducts.forEach(generalProduct => {  
 SubProduct.findAll({where: {General\_Product\_ID\_FK: generalProduct.General\_Product\_ID, Price: {[Op.between]: [request.query.fromPrice, request.query.toPrice]}}}).then(subProducts => {  
 *if* (subProducts.length !== 0) {  
 *let* min = subProducts[0].Price;  
 *for* (*let* i = 0; i < subProducts.length; i++) {  
 *if* (subProducts[i].Price < min)  
 min = subProducts[i].Price;  
 }  
 generalProduct.setDataValue('minimumPrice', min);  
 generalProduct = generalProduct.get({plain: *true*});  
 counter++;  
  
 }  
 *else* {  
 *let* index = generalProducts.indexOf(generalProduct);  
 generalProducts.splice(index, 1);  
 }  
 *if* (counter === generalProducts.length)  
 response.send(JSON.stringify(generalProducts));  
 })  
 });  
 }  
 *else* response.send();  
 });  
};