Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Оновлення інтерфейсу та розробка програмних компонент веб-сайту ТОВ ”СВ Лайт” (комплексна тема)

Виконали: студенти IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Коваль М.О.
(підпис) (прізвище та ініціали)

_____ Романчук Р.О.
(підпис) (прізвище та ініціали)

Керівник _____ Струтинська І.В
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Тиш Є.В.
(підпис) (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Ковалю Максиму Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Оновлення інтерфейсу та розробка програмних компонент веб-сайту ТОВ "СВ Лайт" (комплексна тема)

Керівник роботи Струтинська Ірина Володимирівна, доктор економічних наук, професор кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *16* » *березня* 2022 року № *4/7-161*

2. Термін подання студентом завершеної роботи 21 червня 2022р.

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ ОНОВЛЕННЯ ІНТЕРФЕЙСУ ТА РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ВЕБ-САЙТУ ТОВ "СВ ЛАЙТ". 1.1 Технічне завдання до модернізації проекту. 1.2 Історія мови програмування PHP. 1.3 Вибір середовища проектування логічної частини проекту. 1.4 Стандартизована мова розмітки HTML. 1.5 Вибір середовища редизайну. 1.6 Вибір фреймворку для верстання HTML-документів. . 1.7 Варіанти використання модернізованої системи. 1.8 Висновки до першого розділу. РОЗДІЛ 2. ОНОВЛЕННЯ ТА РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ВЕБ-САЙТУ ТОВ "СВ ЛАЙТ". 2.1 Проектування архітектури. 2.2 Проектування нової бази даних для оновлюваного веб- ресурсу. 2.3 Перелік файлів реалізації функціоналу веб-сайту. 2.4 Модернізація дизайну інтерфейсу. 2.5 Висновок до другого розділу. РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ. 3.1 Роль центральної нервової системи в трудовій діяльності людини. 3.2 . Надзвичайні ситуації екологічного характеру. 3.3. Вимоги безпеки до робочих місць для виконання робіт. 3.4 Загальні вимоги безпеки з охорони праці для користувачів ПК. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про оновлення ПЗ сайту	04.01.2022-30.01.2022	*Виконано*
3.	Переклад та опрацювання джерел про використання	31.01.2022-06.02.2022	*Виконано*
	середовища проектування логічної частини сайту		
4.	Виконання дослідження щодо оновлення дизайну та	07.02.2022-13.02.2022	*Виконано*
	розробка веб-сайту		
5.	Оформлення розділу «Постановка завдання	14.02.2022-06.03.2022	*Виконано*
	Оновлення інтерфейсу та програмних компонент веб-		
	сайту ТОВ “СВ Лайт”»		
6.	Оформлення розділу «Оновлення та розробка	07.03.2022-03.04.2022	*Виконано*
	Інтерфейсу та розробка програмних компонент веб-		
	сайту ТОВ “СВ Лайт”»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи хорони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	07.06.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	08.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	21.06.2022	

Студент _____ Коваль М.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Струтинська І.В.
(підпис) (прізвище та ініціали)

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Романчуку Роману Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Оновлення інтерфейсу та розробка програмних компонент веб-сайту ТОВ "СВ Лайт" (комплексна тема)

Керівник роботи Струтинська Ірина Володимирівна, доктор економічних наук, професор кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *16* » *березня* 2022 року № *4/7-161*

2. Термін подання студентом завершеної роботи 21 червня 2022р.

3. Вихідні дані до роботи

Вступ. РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ ОНОВЛЕННЯ ІНТЕРФЕЙСУ ТА РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ВЕБ-САЙТУ ТОВ "СВ ЛАЙТ". 1.1 Технічне завдання до модернізації проекту. 1.2 Історія мови програмування PHP. 1.3 Вибір середовища проектування логічної частини проекту. 1.4 Стандартизована мова розмітки HTML. 1.5 Вибір середовища редизайну. 1.6 Вибір фреймворку для верстання HTML-документів. . 1.7 Варіанти використання модернізованої системи. 1.8 Висновки до першого розділу. РОЗДІЛ 2. ОНОВЛЕННЯ ТА РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ВЕБ-САЙТУ ТОВ "СВ ЛАЙТ". 2.1 Проектування архітектури. 2.2 Проектування нової бази даних для оновлюваного веб- ресурсу. 2.3 Перелік файлів реалізації функціоналу веб-сайту. 2.4 Модернізація дизайну інтерфейсу. 2.5 Висновок до другого розділу. РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ. 3.1 Роль центральної нервової системи в трудовій діяльності людини. 3.2 . Надзвичайні ситуації екологічного характеру. 3.3. Вимоги безпеки до робочих місць для виконання робіт. 3.4 Загальні вимоги безпеки з охорони праці для користувачів ПК. Висновки. Перелік джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.2022	*Виконано*
2.	Підбір джерел про оновлення ПЗ сайту	04.01.2022-30.01.2022	*Виконано*
3.	Переклад та опрацювання джерел про використання	31.01.2022-06.02.2022	*Виконано*
	середовища проектування логічної частини сайту		
4.	Виконання дослідження щодо оновлення дизайну та	07.02.2022-13.02.2022	*Виконано*
	розробка веб-сайту		
5.	Оформлення розділу «Постановка завдання	14.02.2022-06.03.2022	*Виконано*
	Оновлення інтерфейсу та програмних компонент веб-		
	сайту ТОВ “СВ Лайт”»		
6.	Оформлення розділу «Оновлення та розробка	07.03.2022-03.04.2022	*Виконано*
	Інтерфейсу та розробка програмних компонент веб-		
	сайту ТОВ “СВ Лайт”»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.04.2022-17.04.2022	*Виконано*
8.	Виконання завдання до підрозділу «Основи хорони	18.04.2022-01.05.2022	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	02.05.2022-15.05.2022	*Виконано*
10.	Нормоконтроль	16.05.2022-22.05.2022	*Виконано*
11.	Перевірка на плагіат	07.06.2022	*Виконано*
12.	Попередній захист кваліфікаційної роботи	08.06.2022	*Виконано*
13.	Захист кваліфікаційної роботи	21.06.2022	

Студент _____ Романчук Р.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Струтинська І.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Оновлення інтерфейсу та розробка програмних компонент веб-сайту ТОВ ”СВ Лайт” // Кваліфікаційна робота освітнього рівня «Бакалавр» // Коваль Максим Олексійович, Романчук Роман Олексійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СН-41 // // Тернопіль, 2022 // С. 98, рис. – 71, табл. – 3 , кресл. – 18, додат. – 2, бібліогр. – 31.

Ключові слова: php, веб-сайт, користувач, адміністратор, база даних, laravel, код, сторінка, ресурс.

Кваліфікаційна робота присвячена оновлення інтерфейсу та розробка програмних компонент веб-сайту ТОВ ”СВ Лайт”.

Мета роботи: оновлення інтерфейсу та розробка програмних компонент веб-сайту ТОВ ”СВ Лайт” для збільшення клієнтської бази користувачів та кількості замовлень на освітлювальні пристрої.

У першому розділі кваліфікаційної роботи було розглянуто теоретичні відомості які стосуються теми кваліфікаційної роботи, вибране середовище розробки та редизайну, побудовано діаграми та графіки роботи основних елефентів веб-ресурсу.

В другому розділі кваліфікаційної роботи здійснено розробку функціоналу та редизайн веб сайту ТОВ ”СВ Лайт”, оновлено його базу даних, панель адміністрування веб-ресурсом та реструктуровано файлову систему.

ANNOTATION

Interface refinement and software components development of the website of SV Light LLC // Qualification work of the educational level "Bachelor" // Koval Maksym Oleksiyovych, Romanchuk Roman Oleksiyovych // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SN-41 Group // Ternopil, 2021 // P. – 98, pic. – 71, table – 3, draw. – 18, annexes – 2, ref – 31.

Keywords: php, website, user, administrator, database, laravel, code, page, resource.

Qualification work is designed to update the interface and develop software components of the website of SV Light LLC.

Purpose: to update the interface and develop software components of the website of SV Light LLC to increase the customer base of users and the number of orders for lighting devices.

The first section of the qualification work considered theoretical information on the topics of the qualification work, the chosen development and editing environment, created diagrams and graphs of the main elements of the web resource.

In the second section of the qualification work, the development of functions and editing of the website of SV Light LLC was implemented, its database was updated, the web resource administration panel was updated and the system files were restructured.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

CSS – каскадна таблиця стилів (каскадні таблиці стилів).

HTML – мова гіпертекстової розмітки (HyperText Markup Language).

БД – база даних.

PHP – скриптова мова програмування з відкритим програмним кодом (Hypertext Preprocessor).

LARAVEL – безкоштовний, з відкритим кодом PHP-фреймворк.

ADMIN – користувач, що виконує керування сайтом.

ARTISAN – інтерфейс командної стрічки.

ПЗ – програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність дослідження. Наш світ динамічно розвивається та в процесі цього глибоко інтегрує глобальну мережу під назвою Інтернет у життя людей.

Дана мережа увійшла у всі сфери і в даний проміжок часу є дуже великою за своїм розміром, підтримуючи багато новітніх технологій та створюючи власні екосистеми.

В наш час, коли ми живемо в епоху високих технологій, все ж таки залишаються люди, що мають власний бізнес і досі не просувають власну справу у мережі вважаючи це зовсім не актуальним та не перспективним. Проте, багато хто з підприємців забуває про те, що саме веб-сайт показує їхнє серйозне ставлення до роботи та надає вагому перевагу перед конкурентами коли перед клієнтом стоїть вибір.

Інтернет магазин, представляє з себе одну з найбільш популярних комерційних моделей, яка суттєво підвищує кількість клієнтів, створюючи легкі можливості для замовлення потрібного товару чи ознайомлення з ним.

У наш час, коли поважаюча себе компанія має новітній веб-ресурс – це показує, що вона дійсно відповідальна та серйозно ставиться для своєї справи. В даній кваліфікаційній роботі буде розглянуто веб-сайт компанії ”СВ Лайт”. Так як вона займається виготовлення різного виду освітлювальних приладів, то для неї буде актуально провести оновлення свого власного сайту, який може слугувати як і вітриною з товарами, так і буде давати можливість оформити інтернет замовлення.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи є оновлення інтернет ресурсу за для збільшенні продажів світлових засобів у глобальній мережі інтернет, виготовлених компанією “СВ Лайт” та залученні нових клієнтів які у подальшому стануть постійними покупцями.

Для виконання поставленої мети було сформульовано ряд наступних завдань:

– спроектувати архітектуру (Коваль М. О., Романчук Р. О.);

– спроектувати нову базу даних для оновлюваного веб-ресурсу (Романчук Р. О.);

– реалізувати функціонал веб-сайту (Романчук Р. О.);

– модернізувати дизайн інтерфейсу (Коваль М. О.).

Практичне значення одержаних результатів. Полягає у глибокій модернізації інтернет-ресурсу, який у подальшому буде легко піддаватись модифікації та притягне нових клієнтів які стануть постійними покупцями. А саме було спроектовано нову архітектуру (Коваль М. О., Романчук Р. О., п. 2.1), спроектовано нову базу даних для оновлюваного веб-ресурсу (Романчук Р. О., п. 2.2), реалізовано функціонал веб-сайту (Романчук Р. О., п. 2.3), модернізовано дизайн інтерфейсу (Коваль М. О., п. 2.4).

РОЗДІЛ 1. ПОСТАНОВКА ЗАВДАННЯ ОНОВЛЕННЯ ІНТЕРФЕЙСУ ТА РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ВЕБ-САЙТУ ТОВ ”СВ ЛАЙТ”

1.1 Технічне завдання до модернізації проекту

Постановка технічного завдання та його опис є невід’ємним етапом перед початком розробки. На даному етапі узгоджуються всі необхідні умови щодо результату роботи. Також формулюються обмеження технічної реалізації. Більшість ідей, які неможливі на етапі реалізації або є недоцільними – відкидаються на кроці постановки технічного завдання.

1.1.1 Постановка завдання проекту

Завданням проекту є оновлена версія веб-сайту з продажу електронних приладів, що включає в себе оновлення інтерфейсу, модернізації адмін-панелі, покращення роботи веб сайту за рахунок нових версій програмних компонент. Також взяти до уваги існуючу версію зі збереженням функціоналу та кольорової палітри інтерфейсу.

1.1.2 Формулювання основних функціональних блоків

Оскільки завданням проекту в основному є оновлення, то зміни в проект неглобальні і повторюються з вже існуючою версією сайту. Проект включає в себе загальноприйнятий функціонал інтернет-магазину, який містить такі основні функціональні блоки:

- реєстрація та авторизація користувача;
- пошук товарів через категорії;
- додавання в кошик товарів;

- оформлення замовлення;
- панель перегляду статусу замовлень;
- додавання товарів та редагування інформації про товар (для адміністратора);
- додавання категорій товарів (для адміністратора);
- панель керування замовленнями (для адміністратора).

1.1.3 Формування вимог до технічної реалізації

Існуюча версія веб-сайту працює на PHP версії 5.4. Оскільки хостинг дозволяє використовувати тільки Apache та PHP, слід зберігати набір цих технологій. Необхідне оновлення даного програмного забезпечення для покращеної підтримки. На даний момент актуальною та стабільною версією PHP є 7 версія. Слід уточнити, що хостинг не підтримує повноцінно найсвіжішу 8 версію, тому прийнято рішення оновлюватися тільки до 7 версії.

Щодо редизайну інтерфейсу, то необхідна модернізація, теперішня версія містить велику кількість застарілих стандартів, мало оптимізована під мобільні пристрої та не надає зручного користування в порівнянні з іншими інтернет магазинами. В оновленому інтерфейсі необхідна підтримка мобільних пристроїв, видалення лишнього функціоналу та покращення UX. Також важливо зберегти теперішню палітру кольорів, оскільки підприємство вже має існуючі візитки та мерчендайзинг, в основах якого використовується вже вибрані оранжеві та темно-сірі відтінки.

1.1.4 Опис основного функціоналу

Відштовхуючись від вищеописаного основного функціоналу, слід деталізувати кожен функціональний блок для кращого розуміння основних потреб на етапі розробки.

1.1.4.1 Реєстрація та авторизація

Відштовхуючись від загальноприйнятих стандатрів веб-сайтів, а також вже існуючої версії, веб-сторінка повинна містити форму для реєстрації та авторизації, а також зміни паролю. При реєстрації користувач вводить основну інформацію про себе. При авторизації користувач вводить e-mail та пароль. Авторизований користуач має можливість перегляду статусу своїх замовлень. Також маючи дані користувача стає зручним етам комунікації з клієнтом.

1.1.4.2 Пошук товарів через категорії

Зі збільшенням кількості товарів пошук необхідного продукту ускладнюється. Категоризація товарів надасть змогу швидко та зручно для користувача фільтрувати необхідне серед запропонованого списку продуктів.

1.1.4.3 Додавання в кошик товарів

Кошик є основою для збереження товарів перед купівлею. Серед функціоналу кошика необхідно реалізувати: додавання товару, видалення товару, збереження товару після виходу з сайту.

1.1.4.4 Оформлення замовлення

Після додавання у кошик товару, необхідно виводити форму для оформлення замовлення, а саме дані, які необхідні для доставки продуктів клієнту.

1.1.4.5 Панель перегляду статусу замовлень

Панель перегляду статусу замовлень повинна містити інформацію щодо кожного товару, який був оформлений. Клієнту надається інформація в якому статусі перебуває товар.

1.1.4.6 Додавання товарів та редагування інформації про товар

Дана функція надається адміністраторам інтернет-магазину. Необхідно реалізувати форму, в якій описується основна інформація про товар, його характеристики, а також фотографії. Також слід розробити функціонал для подальшого редагування товарів і додавання до категорій.

1.1.4.7 Додавання категорій

Додавання категорій можливе тільки через адмін-панель. Категорії необхідні для прив’язки товарів і подальшого пошуку. Для категорій розробити форму з введенням назви, а також редагування.

1.1.4.8 Панель керування замовленнями.

Після того, як клієнт оформляє замовлення, адміністратор має мати можливість перегляду замовлення для подальшої комунікації з клієнтом, а уточнення деталей, необхідних для відправлення товарів, в даній панелі обов’язковим є зміна статусу замовлення та інформація, яку залишив клієнт при оформленні

1.1.5 Запропонований стек технологій

Щодо стеку, який пропонується в реалізації оновленої версії веб-сайту, то за основу взято теперішню версію, з оновленим ПЗ, а саме для серверної частини рекомендується:

- PHP 7.4;
- MySQL 8.0;
- Apache 2.4;

Для фронтенду рекомендується використовувати такі рішення:

- Bootstrap 5;

На етапі аналізу технічного завдання та реалізації, стек може доповнюватися згідно потреб розробки, але необхідне чітке дотримання використання основних технологій.

1.2 Історія мови програмування PHP

На сьогоднішній день, мова програмування PHP є дуже поширеною та водночас і потужним рішенням для створення різного виду Веб-застосунків. Також варто зазначити, що PHP був розроблений в рамках проекту Open Source, тобто з відкритим початковим кодом, що у подальшому дало йому можливість набути більшого поширення у широких масах. Дана мова програмування є доволі простою у вивченні, і тим самим має низький поріг входження для початківців.

Історія самої ж мови програмування починається ще далеко в 1995 році, коли програміст ентузіаст Расмус Лердорф створив невеликий застосунок для перегляду статистики відвідування його резюме на мові Perl. Пізніше, коли даним функціоналом для моніторинга статистики користувалось вже декілька людей, а число охочих ставало все більше і більше, Расмус назвав його

Personal Home Page Tools першої версії та виставив у публічний доступ для вільного використання іншими користувачами [1].

Після чого, розробник зрозумів, що на даному етапі не потрібно зупинятись і почав допрацьовувати свій застосунок, додаючи новий функціонал. Тут він зрозумів, що мова програмування Perl не дасть реалізувати усі задумки автора, і тоді було вирішено випустити нову версію на мові програмування C. Тепер вже нова версія інструменту набула назви PHP/FI (Personal Home Page / Forms Interpreter), в подальшому його будуть називати як PHP 2. Нова версія застосунку вже стала більш схожою на сьогоденну мову програмування PHP. Тепер він вже мав спосіб іменування змінних та синтаксис мови програмування Perl, інтеграцію з базами даних, можливість автоматичної інтерпретації форм та поміщення операторів PHP в HTML структуру сторінки. Незважаючи на шалений прогрес в розробці PHP, ним все-одно користувались не більше ніж на п'яти тисячах доменах, що навіть на той час становило не більше одного відсотка від усіх тоді існуючих.

У 1997 році до проекту приєднались Енді Гутманс і Зев Сураскі, які і дали шалений поштовх проекту. На той момент, вони були найкращими студентами одного з найпрестижніших університетів ізраїля, та в процесі створення одного з проектів їхнього навчального закладу намагались використати PHP/FI. Проте вони зіткнулись із багатьма труднощами, та після досконалого вивчення відкритого коду PHP 2, розробники дійшли до висновку, що потрібно докорінно переробити певні аспекти мови. Після декількох місяців кропіткої та наполегливої робити вони все ж таки досягли успіху і після закінчення дійшли згоди з Расмусом про співпрацю в напрямку вдосконалення даної мови. Тоді трапився переломний момент, починаючи з якого утворилась PHP Group – група ентузіастів, що вдосконалювали PHP і продовжую це робити далі. Результатом спільних зусиль у 1998 році було створено вже нову версію цієї мови програмування під назвою PHP 3.

Головною особливість нової версії стало те, що вона дала можливість підключати до ядра нові модулі, що у свою чергу дозволило створювати новий функціонал та цим привабило безліч сторонніх розробників. Даний функціонал дозволив працювати з великою кількістю протоколів, баз даних та міг забезпечити підтримку шаленої на той час кількості API. Поєднання усіх цих нововведень посприяв шаленій затребуваність PHP серед користувачів.

Після цього, програмісти з ізраїля зайнялися переробкою двигуна для мови програмування PHP. Даними модифікаціями, розробники зробили опір на суттєве підвищення продуктивності ти швидкості роботи. З цим завдання вони також успішно справились та в кінці 1999 року, було реалізовано новий двигун під назвою Zend Engine, який вже був на голову вищий за свого попередника. Далі, у 2000 році була випущена мова програмування PHP 4, в основу якої і ліг новий двигун. В цій версії розробники додали ряд корисних нововведень та суттєво збільшили продуктивність. Також слід зазначити, що в четвертій версії з’явилась можливість використовувати сесії, яка і в наш час є доволі актуальною та затребуваною у багатьох проектах, і також сам мова програмування у цій версії стала суттєво безпечнішою.

Наступна версі PHP, під номером п’ять, вийшла вже у середині липня 2004 року. Головним оновлення було новий двигун Zend Engine під номером 2. Дана версія отримала суттєвий приріст у продуктивності, повне підтримку об’єктно орієнтованого програмування, яке в рази полегшело розробку великих проектів, та розширення PDO, що представляє собою універсальний та доволі простий інтерфейс для роботи з різними базами даних.

Далі, мова програмування PHP, почала отримувати змішані відгуки від користувачів, що було пов’язано з тим, що у ній на той момент була відсутня внутрішня підтримка Unicode на рівні мови. Тому у 2005 році, Андрій Змієвський створив проект, метою якого було додавання підтримки Unicode у PHP, за допомогою бібліотеки ICU. Проте, проект не увінчався успіхом, та через брак розробників які не до кінця розуміли важливість даного

нововведення та великий об’єм роботи, що потягне за собою суттєві зміни як для програмістів які модифікують PHP, так і для розробників у майбутньому. Тому проект було вирішено закрити, а функції які були анонсовані для версії PHP 6 та не входили в Unicode, були випущені в якості оновлення для п’ятої версії, та дістали назву PHP 5.

Незважаючи на те, що шоста версія мови програмування так і не була випущена, у 2015 році була анонсована вже нова версія, яка дістала назву PHP 7, та вона містила деякі з таких нових можливостей як анонімні класи, новий оператор “??” та як це було заведено раніше, суттєве збільшення продуктивності. Після чого, вже через п’ять років, а саме в 2020 році, розробники анонсували нову, та на даний момент останню версію мови програмування PHP під номером 8. Серед нових функцій можна відмітити такі як можливість об’єднувати типи змінних, використовувати структуровані метадані з власним синтаксисом та іменовані аргументи [2].

1.3 Вибір середовища проектування логічної частини проекту

Важливим аспектом при виконання поставленого керівництвом завдання, а саме оновлення інтерфейсу та розробка програмних компонентів веб-сайту ТОВ ”СВ Лайт”, є пояснення та перелічення переваг обраних технічних компонентів для середовища проектування технічної складової проекту, що відповідає за логічну частину роботи.

Також слід зауважити, що для модифікації і далі буде використовуватись мова програмування PHP, оскільки попередня версія сайту вже на ній написана, і також дана мова є чудовим вибором для різних інтернет ресурсів.

Першим кроком для реалізації поставленого на передодні завданні, було прийняте рішення, про перенесення попереднього сайту компанії на вже готову систему з вже попередньо реалізованим функціоналом та можливістю легкої модернізації та підтримці проекту у майбутньому. Вибір був серед

трьох систем, таких як OpenCart, Laravel та Magento (Див. таблицю 1.1). Спершу перейдемо до переваг та недоліків систем які вже були відкинуті.

Таблиця 1.1 - Характеристика диверсифікованих систем проектування

№	Система проектування	Характеристика
1	OpenCart	Система контролю вмісту, яка є чудовим вибород для переносу попередньо створеного інтернет-магазину [3]. Вона добре справляється із усіма поставленими перед нею завданнями, але не більше.
2	Laravel	Представляє собою надбудову над голим кодом мови програмування PHP. За допомогою цієї особливості Laravel можна легко модифікувати відповідно до поставленого завдання та створювати різні індивідуальні рішення, які можуть знадобитись чи вже потрібні у проекті. Даний фреймворк є доволі поширений, його широко використовують на багатьох ресурсах, а подекуди, хостинги пропонують автоматично встановити його.
3	Magento	Доволі продуктивна, пропонує широкий функціонал та кастомізується в подальшому. Magento чудово підходить для дуже великих компаній, які мають широкий асортимент товару, фізичні магазини, безліч складів та в період знижок чудово впорається з обробкою великої кількості даних.

Першою була OpenCart. Доволі просте рішення, що у подальшому не може дати можливість вводити різні цікаві та функціональні рішення які можуть бути у своєму роді доволі унікальними [4].

Далі було розглянуто ще одну систему контролю вмісту сайта як Magento. Дана система має потужну складову безпеки, яку система може надати як користувачам, чиї дані повинні знаходитись у безпеці, так і самому інтернет-магазину, що дозволить забезпечити безперебійну роботу виключивши фактор взлому. До мінусів, які стали вирішальними, були віднесено те, що використання даної системи є доволі ресурсозатратно, як в плані технічних вимог так і в фінансовому, оскільки Magento є доволі великою та складною системою. Вона є чудовим вибором для монополістів, але для середніх та малих підприємств принесе лиш витрати.

Отже, після розгляду вище перелічених систем, вибір припав на фреймворк Laravel.

Тепер, коли було обрано основу, на якій буде розміщуватись оновлений інтернет ресурс, оберемо локальне середовище для розробки та відлагодження проекту. Серед таких можна розглянути OpenServer, Xampp та MAMP. Звідси можна одразу зробити висновок, що чудовим вибором буде OpenServer, оскільки він має великий спектр налаштувань, що роблять його потужним інструментом для локальної розробки Веб-додатків, та дане програмне забезпечення містить у собі усі плюси з програм Xampp та MAMP.

Далі, для цього оновлюваного веб-ресурсу потрібно базу даних. Оскільки ми використовуємо фреймворк Laravel, то будемо використовувати одну з баз даних, яку вимагає по офіційній документації дана надбудова і обираємо MySQL. Вона буде використовуватись для обробки та зберігання даних у таблицях, що у свою чергу будуть поєднані одна з одною різними типами зв’язків. Для того щоб проводити ті чи інші маніпуляції з вже створеними таблицями напряму та буде використовуватись система керування базами даних MySQL. Фактично, це є система з відкритим програмним кодом,

який доступний усім охочим для його вивчення та дослідження. MySQL чудово підходить для невеликих та середніх інтернет ресурсів, забезпечуючи високу швидкодію. До головних переваг, можна віднести такі як систему гібридного доступу зі створенням різного виду додаткових акаунтів для різних користувачів, підтримку декількох одночасних запитів та записи стрічок різної довжини.

Для взаємодії з базою даних MySQL, Lavavel має вбудовану систему ORM під назвою Eloquent. Вона надає змогу зручно зберігати, читати, редагувати та видаляти з підключеної бази даних записи, які беруть за основу описані моделі в коді. Після створення опису моделі, Eloquent генерує міграції до бази даних зі створенням у ній таблиці, яка містить всі поля моделі. При взаємодії з описаним класом моделі, Laravel звертається до бази даних і надає інтерфейс для роботи з методами моделі [5].

Редактором для коду, буде програмне забезпечення під назвою Visual Studio Code. Воно було випущене компанією Microsoft ще у 2015 році, та є доволі зручним та допомагає підвищити продуктивність роботи. Що до головних переваг, то до них можна віднести безкоштовність даного ПЗ, активна підтримка та постійні оновлення від розробників, велика кількість додаткових функцій яких може самому додавати з урахуванням індивідуальних потреб, широкі можливості щодо налаштування, як візуальних так і технічних.

1.4 Стандартизована мова розмітки HTML

HTML є мовою гіпертекстової розмітки, що була створена на початку дев’яностих років двадцятого століття, а самим розробником є британськи спеціаліст з інформатики Тім Бернерс-Лі. Сама ж мова було розроблена на основі стандартної узагальненої мови розмітки, скорочено SGML. Головною ідеєю SGML було створити між платформенну мову розмітки для документів,

і цим завдання занялись розробники з компанії IBM, головною метою яких було відокремити саму структуру документа від його ж оформлення.

Обгрунтуванням такої розмітки є введення в код спеціальних структур управління, які відокремлюють елементи структури, але не вказують, як ці вони мають відображатися. А структура, що відповідає дизайну документа, оформляється в окремому файлі, так званій таблиці стилів. Така структура розмітки документа дозволяє відображати ту саму інформацію в різних візуальних дизайнах [6].

Розробники системи вважали, що надійно обробляються лише документи, які відповідають одному стандарту, а всі документи, повинні бути відхилені. Тому необхідно чітко визначити структуру файлу. Для цього створено механізм визначення типу документа, так званий Document Type Definition. DTD, як і таблиці стилів, є зовнішніми файлами щодо відповідного документа.

Поділ документів на структуру, стилі та стандарти дозволяє їх редагувати, не змінюючи саму структуру документа. Дана мова, що відповідає всьому вищезазначеному, була створена в 1969 році і отримала назву Generalized Markup Language, а вже у 1986 році ISO прийняла мову GML як міжнародний стандарт. Після цих подій до назви мови була додана англійська літера “S”.

Мова SGML мала попит, до тих пір поки у науковій спільноті не знадобився інструмент, який міг би швидко та легко передавати інформацію. Система Gopher була розроблена в 1991 році, але вона не підтримує графіку або гіперпосилання. Далі було розпочато роботу над створенням першого текстового браузера з підтримкою гіпертексту. Нова мова HTML, була розроблена для забезпечення простоти та ефективності передачі інформації. Слід зауважити, що вона часто повторює своїх предків, але головна відмінність полягає в тому, що окремі елементи конструкції повинні бути відображені особливим чином.

У вересні 1993 року група програмістів на чолі з Марком Андерсоном випустила перший текстовий браузер Mosaic. Він став дуже поширеним, оскільки його розповсюджували безкоштовно та його можна було записати всього лиш на одну дискету. Програма підтримувала ввід за допомогою комп’ютерної мишки та могла відображати зображення.

Далі, 1994 року була створена організація зі стандартизації гіпертекстової розмітки. Вона отримала назву World Wide Web Consortium. W3C розпочала свою роботу над HTML версії 2.0. Починаючи з HTML 3.0, була реалізована підтримка CSS, яка спрямована на відокремлення елементів. Деякі з відокремлених елементів відповідали за відображення елементів, інші ж за структуру документа.

На початку 1994 року група розробників браузерів Mosaic разом із Джеймсом Кларком, заснували компанію Netscape Communications, а через півроку вийшов перший комерційний браузер Netscape. Метою його створення, стало бажання охопити якомога більшу групу користувачів. Netscape представив нові теги й вдосконалення Html, що підтримувались лише цим браузером. Також слід зазначити, що майже усі нещодавно введені теги, призначені для покращення зовнішнього вигляду документа та розширення його можливостей форматування, підтримувались даним браузером.

Влітку 1996 року було випущено Internet Explorer 3.0, який підтримує майже всі розширення Netscape і мав дружній до користувачів інтерфейс. Четверта версія браузера вийшла практично одразу після попередньої і не сильно відрізнялася за швидкістю та іншими параметрами від попередника, але однією з чудових переваг Internet Explorer було те, що він був безкоштовним та попередньо встановлений на операційну систему Windows і фактично став стандартом у цій галузі.

1.5 Вибір середовища редизайну

Ключову роль при оновленні інтерфейсу відіграє середовище для створення дизайну веб-сайта. При редизайні необхідний комплексний набір інструментів для організації розробки оновленого інтерфейсу. На етапі обговорення з власником прийнято рішення не змінювати палітру кольорів для збереження брендингу. Серед інструментів розробки інтерфейсів слід розглядати такі варіанти (Див. таблицю 1.2).

Таблиця 1.2 – Характеристика інструментів розробки інтерфейсів веб ресурсів

№	Інструмент	Переваги	Недоліки
1	Figma	1. Безкоштовний доступ; 2. Можливість працювати у браузері; 3. Надання доступу до розробки для членів команди; 4. Бібліотека шрифтів.	1. При розробці великих проектів спостерігається значне підвисання сервіса; 2. При нестабільному інтернет з'єднанні рекомендується використовувати десктопне рішення.
2	Adobe XD	Переваги: – Можливість створення інтерактивного дизайну; – Подібність функціоналу для користувачів продукції компанії Adobe Systems; – Велика кількість плагінів для розширення функціоналу; – Колекція шрифтів сувмістима з Adobe Photoshop.	– Можливість працювали тільки в дектопному режимі і необхідність завантажувати на ПК; – Безкоштовна версія містить тільки обмежений функціонал.

Продовження таблиці 1.2

3	Sketch;	–Запозичує фірменні дизайнерські рішення від Apple; –Приємний інтерфейс. –Недоліки: –Можливість працювати тільки на продукції від компанії Apple; –Обмежена кількість плагінів від третіх сторін.	-
4	Adobe Photoshop	– Підтримка спільнотою; – Універсальність середовища; – Знайомий інтерфейс.	– Продукт платний; – Здебільшого функціонал призначений для обробки фотографій.

Figma – векторний графічний редактор, метою якого є безкоштовне надання інструментарію для створення інтерфейсів. Даний сервіс розроблений двома студентами спеціальності “комп’ютерні науки” Діланом Філдом та Еваном Волесом під час навчання в університеті Брауна. Метою їх розробки стало надання безкоштовного доступу до створення макетів у браузері. На даний доступ до інструментарію надається через веб-браузер або окреме десктопне рішення. Figma надає можливість працювати з великою кількістю шрифтів, а також роботу в команді [7].

Adobe XD – програмне забезпечення для дизайну інтерфейсів, розроблена компанією Adobe Systems. Через поширене використання Adobe Photoshop у сфері веб-дизайну, компанія Adobe Systems взяла до уваги прихильність цього продукту серед дизайнерів та розробила окреме рішення для цієї групи людей. Adobe XD успадкував особливості інтерфейсу, набір гарячих клавіш та бібліотеку шрифтів від попередніх продуктах компанії-розробника, що надало перевагу у виборі даного програмного забезпечення для користувачів продукції Adobe. Adobe XD дає можливість використання плагінів, розроблених третіми сторонами, а також інтегрувати з популярним

продуктом компанії Adobe Photoshop. Бібліотека шрифтів не обмежує творців у створенні дизайнерських рішень [8].

Sketch – векторний графічний редактор для розробки інтерфейсів, який призначений для розробки на продукції Apple. Даний графічний редактор розроблений компанією Bohemian Coding і отримував нагороду Apple Design Award у 2012 році. Обмеженість операційною системою Apple часто критикується серед дизайнерів, які віддають перевагу іншим іншим операційним системам, що часто зупиняє компанії у виборі саме цього графічного редактора в якості основного інструменту для створення дизайну своїх продуктів. Також інклюзивність, на думку багатьох творців, не дає змогу розширення функціоналу за рахунок написання плагінів третіми сторонами [9].

Adobe Photoshop – графічний редактор від компанії Adobe Systems. Здебільшого призначений для редагування цифрових фотографій та обтобки растової графіки, але також активно застосовується для розробки дизайну графічних інтерфейсів. Неможливо виділяти популярність даного продукту серед творців різних професій. Системне використання Adobe Photoshop в компаніях різноманітних сфер дозволило також досягнупи популярності і серед дизайнерів, оскільки дозволяє працювати з вже знайомим інтерфейсом. Також слід виділити бібліотеку шрифтів і велику кількість плагінів, які розширюють функціонал для зручної розробки дизайнів інтерфейсів [10].

З вищеперерахованих середовищ редизайну вибрано інструмент Figma. На відмінно від альтернатив, Figma надає повний безкоштовний доступ до основного інструментарію, а також має велику спільноту, що надає змогу швидко знайти потрібну інформацію. Вибраний графічний редактор також рекомендується для використання багатьма спеціалістами у сфері створення графічних інтерфейсів. Також слід звернути увагу на можливість працювати над одним проектом в команді, надаючи доступ до основного документу [11].

1.6 Вибір фреймворку для верстання HTML-документів

Сучасна розробка веб-сайтів неможлива без етапу верстки HTML-документів. На даному етапі формується основний вигляд веб-сторінок. Після створення дизайну в графічному редакторі, основною задачею є перенесення вигляду інтерфейсу в HTML та CSS документи. На сьогоднішній день більшість веб-сторінок мають подібку структуру. Стандартизація вигляду веб-сайтів надала змогу появитися великій кількості інструментів для зручної розробки. З найпопулярніших рішень виділяється Bootstrap.

Bootstrap – фреймворк для розробки вигляду веб сторінок, який включає у себе весь необхідний інструментарій для швидкої реалізації дизайну веб-сторінок. Bootstrap, як внутрішня розробка компанії Twitter починає свою історію в 2011 році. Раніше компанія використовувала різні бібліотеки для створення вигляду сторінок своєї соціальної мережі, але з часом це привело до ускладнення підтримки і суперечностей у команді. Після успіху в реаліхації та використанні Bootstrap у внутрішніх проектах Twitter, соціальна мережа вирішила відкрити сирцевий код свого розробленого фреймворку. З часом фреймворк розвивався за рахунок активного використання його розробниками у своїх проектах та відкритості серцевого коду, що дало змогу швидко виправляти помилки спільнотою та розвивати новий функціонал. На даний момент актуальною версією є Bootstrap 5. Серед основних змін слід відзначити відсутність використання JQuery, зміна під нові версії браузера, а також виправлення багів [12].

Також слід розглянути алтернативи фреймворку Bootstrap, однією із них є Foundation. Foundation – це фроеймворк для фронтенду, який орієнтований для розробки адаптивного дизайну. Серед популярних компаній, які використовують даний фреймворк числяться Adobe Systems, Mozilla, Cisco. Foundation розроблений компанією ZURB, серцевий код опублікований у жовтні 2011 року. Фреймворк активно розробляється і тестується на багатьох

браузерах. Foundation взяв за основу надати можливість розробникам швидко та зручно адаптувати свої проекти під різноманітні дивайси, це означає, що веб-сторінка, яка розроблена за допомогою даного фреймворка, не матиме проблем з відображенням блоків дизайну як на широкоформатному моніторі, так і на мобільному пристрої [13]. Також в основному написання стилів для веб-сторінок, виконується на мові Sass, яка є своєрідною абстракцією над мовою CSS, що дає можливість краще організувати код проекту для розробників.

Material UI – це ще один аналог до Bootstrap, розроблений компанією Google. В основному стиль Material UI є популярним у мобільних пристроях на ОС Android, але також є фреймворком для робробки веб-додатків. Даний фреймворк дотримується основних принципів дизайну, які започаткував Google у своїй продукції. Веб-сторінка, яка використовує Material UI у своєму інтерфейсі містить чітко виділені тіні, просторовість об’єктів та візуальну мову у іконках, що надає змогу краще орієнтуватися у інтерфейсі. Google використав свої ресурси та досвід у розробці інтерфейсів користувачів для побудови базових принципів Material UI. Фреймворк активно адаптується під потреби часу та тренди в дизайні інтерфейсів. Хоча і Material UI є знайомим великій кількості людей, які користуються ОС Android, дизайнери не активно використовують його у своїх працях, оскільки фреймворк не надає великого простору для розробки унікальних інтерфейсів користувача [14].

Skeleton є ще одним аналогічним рішенням до Bootstrap. В основному орієнтований для простих проектів з мінімалістичним дизайноом. Його основна відмінність від вище перерахованих фреймворків є у простому використанні та швидкому підключенні. Skeleton, як і Bootstrap, включає систему сіток на веб-сторінці з адаптивністю під мобільні пристрої. Також містить свої рішення для дизайну кнопок, таблиць та форм, а унікальний мінімалістичний стиль дозволяє без досвіду в дизайні графічних інтерфейсів створювати приємні дизайни веб-сторінок [15].

Після огляду найпопулярніших фреймворків для розробки веб-сторінок, прийнято рішення використовувати Bootstrap. Bootstrap на сьогоднішній день є найпопулярнішим рішенням. Команда Twitter та незалежні розробники створили унікальний проект, який вважається стандартом у розробці сітки веб-сторінок.

1.7 Варіанти використання модернізованої системи

Кожен Веб-ресурс, який розміщений в інтернеті, має своє власне зовнішнє оточення, що складається з тих чи інших користувачів. Самими ж користувачами є люди, які у свою чергу використовують ресурс за заздалегідь визначеними алгоритмами та у результаті їх виконання очікують певного результату. Схеми використання даного інтернет ресурса який був оновлений, називають варіантами використання, або ж з англійської Use case.

У даному інтернет ресурсі присутні такі актори:

- Незареєстрований користувач.
- Зареєстрований користувач.
- Адміністратор ресурсу.

Звідси, на рисунку 1.1 можна ознайомитись з діаграмою акторів.

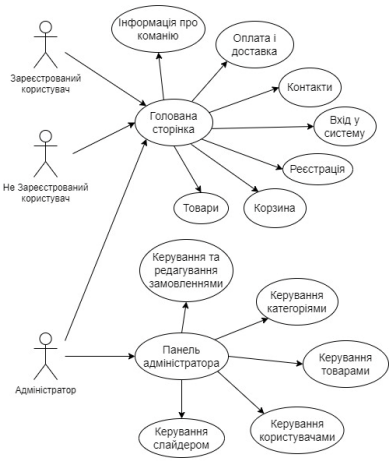

Рисунок 1.1 – Діаграма акторів

Веб-сайт повинен повною мірою реалізовувати покладених на нього функцій, таких як реєстрація користувачів, можливість увійти до системи, вибору потрібних товарів та додавання їх у корзину та оформлення замовлення. Короткий опис даних функцій наведено у таблиці 1.3.

Таблиця 1.3 – Таблиця варіантів використання

Актор	**Назва варіанту використання**	**Формулювання варіанту**
Не зареєстрований користувач	Реєстрація користувача	Незареєстрований користувач має змогу зареєструватись у системі.
	Оформлення замовлення	Незареєстрований користувач може обрати товари, додати їх у корзину та оформити замовлення.
Зареєстрований користувач	Вхід у систему	Користувач може увійти в систему.
	Повідомлення про невдалий вхід	При невірно введених даних користувач отримає повідомлення про це.
	Оформлення замовлення	Користувач має змогу додати товар у корзину та оформити заказ.
Адміністратор	Вхід у панель адміністратора	Для входу у панель адміністратора, сам адміністратор використовує форму для входу у систему яке призначене для всіх користувачів, та після перевірки доступу, його буде автоматично направлено до панелі керування.

Продовження таблиці 1.3

Адміністратор	Обробка заказів	Адміністратор може обробляти закази, змінювати певну інформацію про доставку та статус самого заказу.
	Керування категоріями	Адміністратор має змогу редагувати та видаляти вже існуючі категорії, а якщо якоїсь бракує то може створити нову
	Керування товарами	Адміністратор може керувати усіма продуктами які є у інтернет магазині, тобто редагувати продукти, видаляти чи додавати нові, вказуючи категорії кожного з товарів
	Керування користувачами	Адміністратор має змогу редагувати контактні дані користувачів та змінювати їхні паролі до облікових записів
	Керування слайдером	Адміністратор має можливість керувати зображеннями банера

Ознайомившись з даною таблицею, можна оглянути усіх акторів та їхніми можливими варіантами використання оновленого ресурсу.

Розглядаючи простих користувачів даного інтернет-магазину можна сказати, що не важливо чи користувач зареєстрований і увійшов у систему чи він зовсім не зареєстрований, можливість створити та оформити замовлення

присутня. Це зроблено для збільшення клієнтської бази, а саме полегшення можливостей для здійснення заказу постими користувачами які мають власні акаунти та щоб не відлякувати нових, які вперше потрапили на сайт і не хочуть вносити свої данні у систему.

Беручи до уваги адміністратора, у нього є можливості не лише простого користувача, а й можливість керування Веб-ресурсом. А саме, вхід в обліковий запис адміністратора реалізований через форму для входу усіх користувачів. Система сама проводить перевірку чи даний користувач є адміністратором, і коли перевірка успішно пройдена, то відбувається автоматична переадресація на панель керування. На даній сторінці він має змогу обробляти усі закази та проводити оперативне управління тими чи іншими аспектами ресурсу. На рисунку 1.2 можна побачити розширену UML діаграму адміністративної частини.

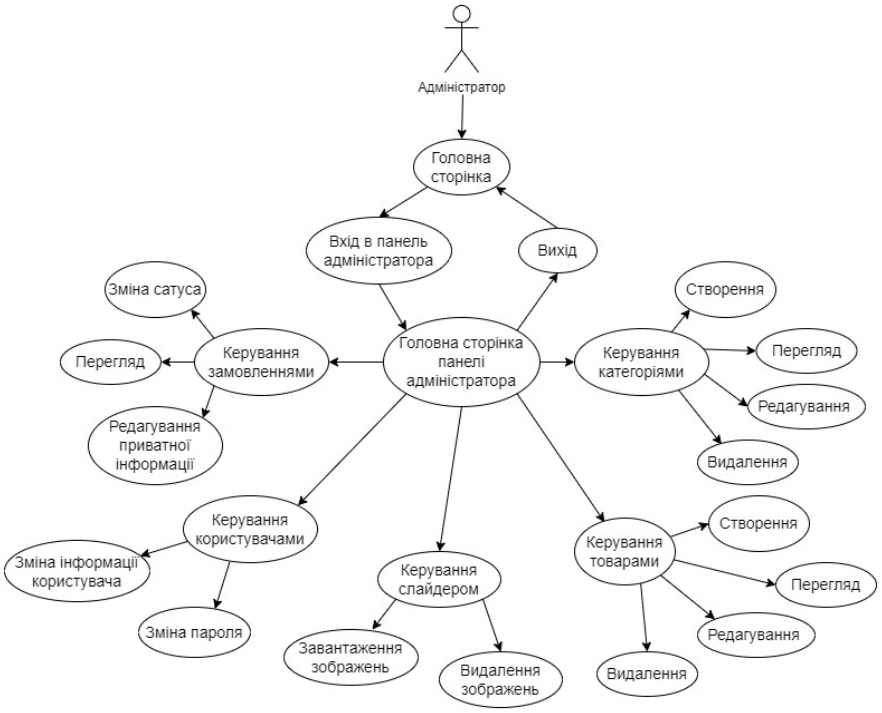

Рисунок 1.2 – Розширена UML діаграма адміністративної частини

Вище зображена діаграма показує усі можливі варіанти використання панелі керування адміністратором [16].

1.8 Висновки до першого розділу

В даному розділі ми оглянули технічне завдання по оновленні інтерфейсу та розробки програмних компонент веб-сайту ТОВ ”СВ Лайт”, на основі якого вибрали основний інструментарій для розробки.

Склали діаграму акторів та UML-діаграму для адміністративної частини сайту.

Обрали середовище редизайну та основний стиль для інтерфейсу проекту.

РОЗДІЛ 2. ОНОВЛЕННЯ ТА РОЗРОБКА ІНТЕРФЕЙСУ ТА РОЗРОБКА ПРОГРАМНИХ КОМПОНЕНТ ВЕБ-САЙТУ ТОВ ”СВ ЛАЙТ”

2.1 Проектування архітектури

Після проведення попереднього аналізу оновлюваного веб-ресурсу, було прийнято рішення, що найкращим вибором буде розділення системи на дві складові, клієнтську та серверну. Клієнтська частина представляє з себе аспекти ресурсу з якими взаємодіє користувач, як наприклад продукти, корзина, сторінка з новинами та інше. Серверна ж частина представляє з себе всі логічні аспекти проекту, а саме обробка тих чи інших запитів сервером, таких як додавання товарів в корзину, редагування корзини чи оформлення замовлення. Детальніше, з даною архітектурою можна ознайомитись на рисунку 2.1.

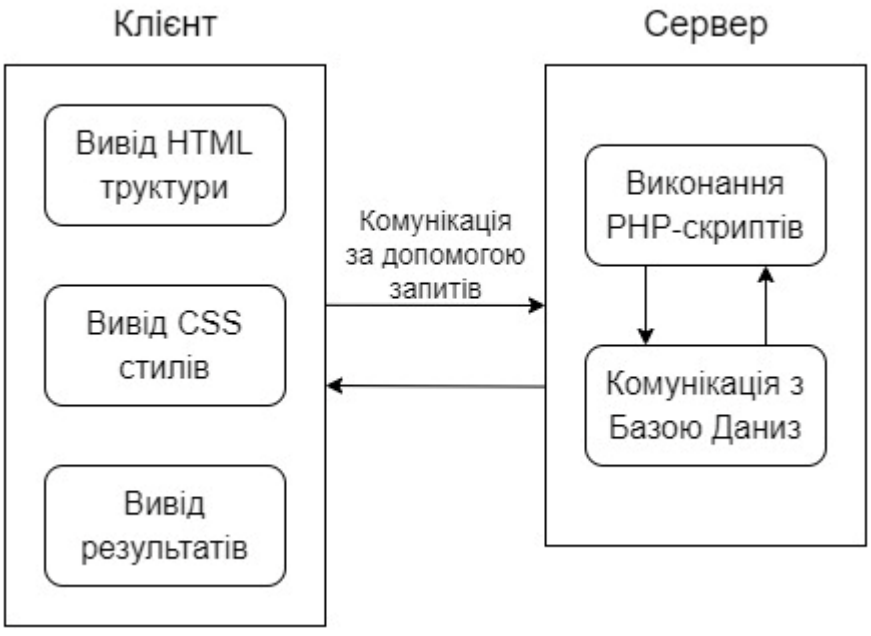

Рисунок 2.1 – Архітектура веб-сайту

Разом, ці два аспекти обраної архітектури чудово комунікують між собою за допомогою запитів між клієнтською частиною та серверною. У сумі,

вони чудово працюють та забезпечують безперебійну роботу ресурсу як для користувачів так і для адміністраторів, які проводять обробку оформлених замовлень, комунікацію з замовниками та створення різного типу новинних оголошень.

2.2 Проектування нової бази даних для оновлюваного веб-ресурсу

Оскільки, проводиться повне оновлення веб ресурсу, було прийнято рішення щодо повної реорганізації бази даних. Тобто вона буде містити у собі нові таблиці та зв'язки між ними, що покращить роботу даного ресурсу позитивним чином та позитивно впливає як на безпеку так і на швидкодію. На рисунку 2.2 зображено структуру оновленої бази даних.

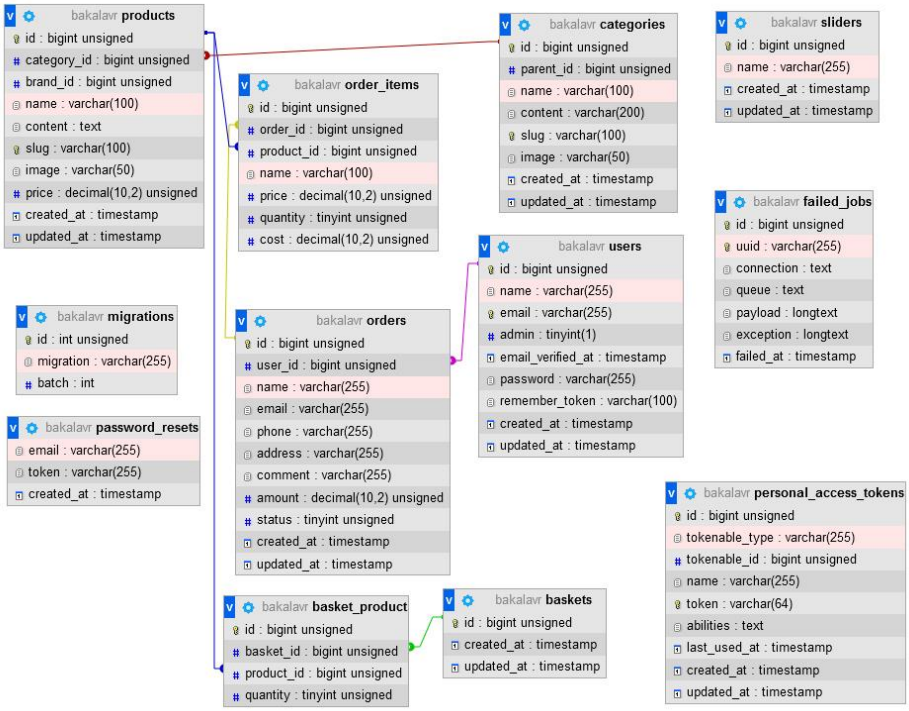

Рисунок 2.2 – Структура БД оновлюваного веб-сайту

Розглянувши зображення вище, можна побачити, такі таблиці як: baskets, basket_product, categories, failed_jobs, migrations, orders, order_items,

password_resets, personal_access_tokens, products, users, slider. Тепер розглянемо їх та для чого вони потрібні. Спершу почнемо з таблиць які потрібні самому фреймворку Laravel. Першою з них буде failed_jobs, у ній поміщається невдало виконані задачі, наступною буде personal_access_tokens у якій будуть знаходитись токени користувачів які будуть використовувати API даного сервісу. Таблиця migrations призначена для збереження вже виконаних міграцій у фрейморці за допомогою яких розробники створюють власні таблиці. Тепер перейдемо до таблиць які вже створені за допомогою міграцій. Спершу розглянемо таблиці categories та products, у них відповідно будуть зберігатись категорії та товари які до них належать. Також слід відмітити що ці таблиці мають зв’язки між собою, і з кожного товару у системі управління базами даних можна перейти відразу до категорії до якої цей товар належить. Далі розглянемо такий перелік таблиць як baskets, basket_product, orders та order_items. Усі вони пов’язані між собою, як і зв’язками у самій БД так логічними зв'язками, оскільки вони містять у собі усю інформацію про те чи інше замовлення, хто це замовлення оформив та коли. У таблиці basket міститься інформація про порядковий номер замовлення та коли воно було оформлено. Відповідно до цього номера, у таблиці basket_product вміщується інформація про індивідуальний номер кожного продукта. У таблиці ж orders, поміщуються вже оформлені замовлення, а саме головна інформація відповідно до нього, таку як куди воно повинно відправлятись, коментар, статус та приватна інформація клієнта і також можна зрозуміти, чи зареєстрований даний користувач чи ні. Далі, у таблиці order_item знаходять товари які замовив користувач і водночас вони прикріплені до відповідного їм елемента у таблиці orders. У ній поміщені такі значення як порядковий номер замовлення, порядковий номер товару, його назва, ціна, кількість та загальна вартість того чи іншого товару, якщо наприклад його кількість більша однієї одиниці. У таблиці slider розміщуються інформація про зображення які будуть зберігатись адміністратором ресурсу та використовуватись у слайдері.

Останніми ж будуть таблиці users та password_resets. У першій розміщується інформація про створених користувачів, їхні адреса електронних адрес та чи вони підтверджені, поле яке показує, чи є даний користувач адміністратором, захешований пароль, та особистий токен, якщо такий існує. У другій таблиці зберігається інформація про відновлення паролю, якщо трапляються випадки коли користувач забув вже існуючий [17].

2.3 Перелік файлів реалізації функціоналу веб-сайту

При оновленні даного ресурсу, було створено велику кількість файлів які пов’язані між собою та забезпечують роботу веб-сервісу. З ними можна ознайомитись на рисунку 2.3.

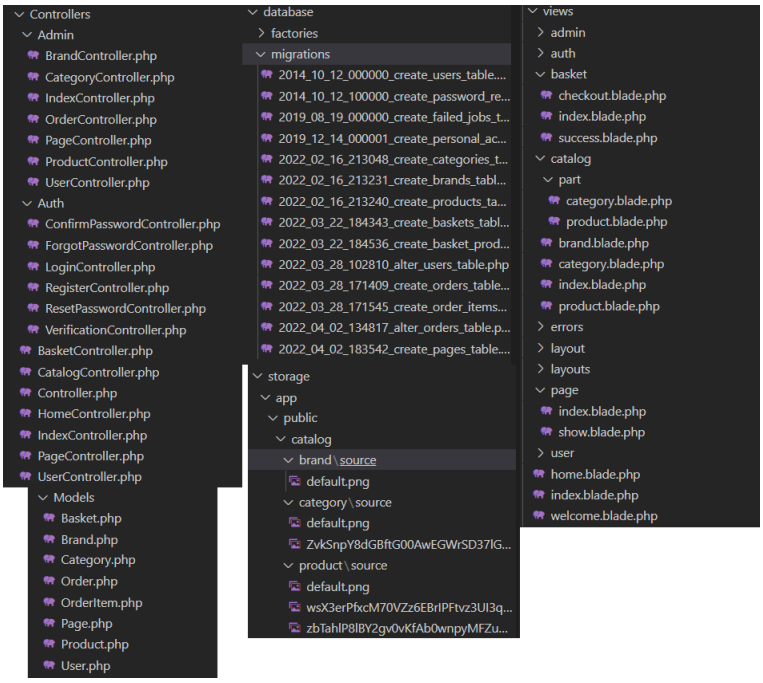

Рисунок 2.3 – Файлова структура оновлюваного веб-сайту

Файли, зображені на рисунку вище використовуються для реалізації усіх функцій сайту, далі ми детально поговоримо про увесь функціонал та розглянемо частини коду.

2.3.1 Створення таблиці з продуктами та категоріями

Першим кроком при оновленні веб ресурсу, було створення головних таблиці, у яких і будуть зберігатись всі дані про продукти та категорії. Для того щоб це зробити, створимо дві моделі та одночасно з ними, відповідні їм файли міграцій за допомогою команд:

- php artisan make:model Category -m;
- php artisan make:model Product -m.

Фреймворк Laravel, побудований таким чином, що за допомогою моделей ми можемо повністю комунікувати з таблицями у базі даних. Кожна таблиця має відповідний файл моделі, за допомогою яких можна отримувати, видаляти, додавати чи змінювати ті чи інші записи. Файли міграцій представляють з себе систему, схожу до контролю версій БД. За допомогою них ми створюємо таблиці, зв'язки між таблицями, зовні ключі та інше, у спрощеній формі при порівнянні з мовою SQL.

Після того як було створено моделі та міграції, в других ми пропишемо поля які повинні міститись в таблицях. На рисунку 2.4 можна побачити поля які будуть у таблиці category.

```php
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('parent_id')->nullable(false)->default(0);
        $table->string('name', 100);
        $table->string('content', 200)->nullable();
        $table->string('slug', 100)->unique();
        $table->string('image', 50)->nullable();
        $table->timestamps();
    });
}
```

Рисунок 2.4 – Робоча частина файлу
2022_02_16_213048_create_categories_table.php

У другому файлі, який буде відповідати за створення таблиці products будуть деякі відмінності у порівнянні з попередніми. Вони заключаються в тому, що крім прописаних полів, також будуть прописані зовнішні ключі, що будуть створювати зв’язки з таблицями категорій та брендів. Робочу частину даного файлу можна побачити на рисунку 2.5.

```php
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->bigInteger('category_id')->unsigned()->nullable();
        $table->bigInteger('brand_id')->unsigned()->nullable();
        $table->string('name', 100);
        $table->text('content')->nullable();
        $table->string('slug', 100)->unique();
        $table->string('image', 50)->nullable();
        $table->decimal('price', 10, 2, true)->default(0);
        $table->timestamps();

        // внешний ключ, ссылается на поле id таблицы categories
        $table->foreign('category_id')
            ->references('id')
            ->on('categories')
            ->nullOnDelete();
    });
}
```

Рисунок 2.5 – Робоча частина файлу
2022_02_16_213240_create_products_table.php

Після цього, як файли моделей та міграцій створені, а останні наповнені потрібним кодом, потрібно запустити міграцію за допомогою командного рядка та команди:

– php artisan migrate.

При виконанні даної команди, фреймворк автоматично запустить процес міграції, який зчитує код з файлів міграцій та створить таблиці [18]. Також слід зауважити, що при виконанні міграцій, в базі даних автоматично створюється таблиця migrations у яку будуть записуються вже виконані файли.

Тепер, таблиці готові до роботи та в них вже можна поміщати інформацію через панель адміністратора. Спершу потрібно створити категорію, після чого, вже під час створення продукта, можна буде посилатись одразу на ці дані [19].

2.3.2 Передача інформації з БД на користувацьку частину

Тепер, коли у базу даних добавлені деякі продукти та категорії, з ними можна комунікувати. Вся комунікація відбувається за допомогою контролерів, у яких і прописується код який буду виконували різні логічні завдання.

Спершу створимо контроллер. Це можна зробити за допомогою командної стрічки та команди:

- php artisan make:controller CatalogController.

Після виконання команди, буде створений файл контролера під назвою Catalog, де і буде прописуватись уся логіка. Також, не слід забувати, що для того, щоб у контролері можна було обробляти дані з бази даних, до нього потрібно під’єднати відповідні моделі за допомогою стандартного простору імен мови програмування PHP. Це можна побачити на рисунку 2.6.

```
use App\Models\Category;
use App\Models\Product;
```

Рисунок 2.6 – Підключення простору імен у контролері Catalog.php

Наступним кроком, ми створимо роути. Це шляхи, які вказують на той чи інший файл у системі, який вміщує в собі структури сторінок та у нього можна передавати різну оброблену інформацію з контролерів. Тому, у файлі web.php, пропишемо потрібні нам шляхи, які можна побачити на рисунку 2.7.

```
Route::get('/catalog/index', 'App\Http\Controllers\CatalogController@index')->name('catalog.index');

Route::get('/catalog/category/{slug}', 'App\Http\Controllers\CatalogController@category')->name('catalog.
category');

Route::get('/catalog/brand/{slug}', 'App\Http\Controllers\CatalogController@brand')->name('catalog.brand');

Route::get('/catalog/product/{slug}', 'App\Http\Controllers\CatalogController@product')->name('catalog.
product');
```

Рисунок 2.7 – Прописані роути у файлі web.php

Тепер, коли все підготовлено, можна починати працювати з контролером та створювати логіку файлу. У ньому буде знаходитись три функцій, їх можна побачити на рисунку 2.8.

```
public function index()
{
    $roots = Category::where('parent_id', 0)->get();
    return view('catalog.index', compact('roots'));
}

public function category($slug)
{
    $category = Category::where('slug', $slug)->firstOrFail();
    return view('catalog.category', compact('category'));
}

public function product($slug)
{
    $product = Product::where('slug', $slug)->firstOrFail();
    return view('catalog.product', compact('product'));
}
```

Рисунок 2.8 – Робоча частина файлу Catalog.php

Перша функція, під назвою index, отримує в себе усі створені категорії з бази даних. Далі, за допомогою функції return, у ньому повертається вигляд catalog.index у який передається усі отримані категорії. Вигляд, або так звана функція view, в свою чергу звертається до роута, який був прописаний у файлі

web.php та відбувається вже завантаження сторінку на яку була передана інформація.

Наступні ж дві функції, такі як category та product, спершу отримують в собі змінну. Наприклад функція category, отримує назву категорії з URL стрічки і тоді вже з бази даних отримуються усі продукти відповідно до цієї категорії і аналогічно до першого метода передаються до клієнтської частини. А ось з методом product вже є відмінності. При виборі продукти, та натисканні на нього, в URL стрічці відображається поле slug, яке є індивідуальним для кожного продукту, і за цим значенням, з таблиці, де розміщуються продукти, обирається лише одне. Тоді ці дані передаються на сторінку продукту [20].

Метод виводу даних практично ідентичний для усіх сторінок, відмінність полягає лише у структурі файлу, тобто як він буде відображатись користувачеві. Розглянемо приклад виводу усіх продуктів певної категорії, та як це відбувається за допомогою шаблонізатора Blade. На рисунку 2.9 можна побачити код.

```
<h1>{{ $category->name }}</h1>
<p>{{ $category->content }}</p>
<div class="row">
    @foreach ($category->products as $product)
    @include('catalog.part.product')
    @endforeach
</div>
```

Рисунок 2.9 – Робоча частина файлу category.blade.php

Суть шаблонізатора полягає в тому, що він має вже свій визначений синтаксис, який дозволяє динамічно генерувати сторінки, при тому мінімізуючи використання чистого PHP коду у файлах, або і зовсім дозволяє його не використовувати, на заміну пропонуючи вже готові функції шаблонізатора. Як можна побачити на рисунку 2.9, на сторінці динамічно виводяться всі продукти певної категорії, та слід врахувати і те, що

шаблонизатор Blade дозволяє розбивати сторінкин на частини, так як тут. Оскільки сама ж карточка продукту розміщується в окремому файлі.

2.3.3 Реалізація функціоналу корзини

Тепер перейдемо до однієї з найголовніших функцій оновлюваного веб-ресурсу, це реалізація корзини для покупок. Вона реалізується таким чином, що спершу клієнт натискає додати товар в корзину, після чого створюється новий запис в таблиці baskets, у якій створюється запис з власним індивідуальним номером та часом створення. Після чого, даний номер запису поміщується у куки, які будуть використовуватись для виводу вмісту корзини. Далі, вже у таблиці basket_product будуть поміщуватись записи з продуктами, які будуть посилатись на особистий номер корзини користувача. Перейдемо до детального розгляду реалізації вищеописаного функціоналу.

Спершу, створимо моделі з міграцією для самої корзини, за допомогою командної стрічки та команди:

- php artisan make:model -m Basket.

Після чого, перейдемо у саму міграцію, яка буде створювати таблицю basket, та вкажемо, які поля повинна містити у таблиці [21]. Це можна побачити на рисунку 2.10.

```php
public function up()
{
    Schema::create('baskets', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

Рисунок 2.10 – Робоча частина файлу 2022_03_22_184343_create_baskets_table.php

На рисунку вище, можна побачити, які поля буде вміщувати в собі таблиця baskets. Їх не багато, оскільки вся інформація про продукти буде зберігатись в іншій таблиці, яка буде називатись basket_product. Для неї нам вже не буде потрібна модель, тому одразу створимо нову міграцію за допомогою команди:

- php artisan make:migration create_basket_product_table.

Після того, як міграція створена, перейдемо до самого файлу та створимо усі потрібні поля. Це можна побачити на рисунку 2.11.

```php
public function up()
{
    Schema::create('basket_product', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('basket_id');
        $table->unsignedBigInteger('product_id');
        $table->unsignedTinyInteger('quantity');

        $table->foreign('basket_id')
            ->references('id')
            ->on('baskets')
            ->cascadeOnDelete();
        $table->foreign('product_id')
            ->references('id')
            ->on('products')
            ->cascadeOnDelete();
    });
}
```

Рисунок 2.11 – Робоча частина файлу
2022_03_22_184536_create_basket_product_table.php

Як можна побачити з рисунку, були створені поля, що вміщують в себе ідентифікатори корзини, продукта який був доданий в корзину та його кількість. Оскільки у таблиці basket_product, використовуються елементи з таблиць basket та products, то одразу створимо для них зовнішні ключі, які будуть посилатись на потрібні нам поля [22]. Далі, коли файли міграцій вже підготовлені, запустимо саму міграцію за допомогою команди:

- php artisan migrate.

Тепер, у базі даних створені таблиці які потрібно для формування корзини. Після цього, створимо контролер, у якому вже буде виконуватись уся логіка корзини. Це можна зробити за допомогою команди:

- php artisan make:controller BasketController.

Наступний кроком буде створення функціоналу по додаванні товару в корзину. На сторінці товару реалізуємо поля для вводу кількості товару та кнопку, яка відповідає за додавання товару до корзини, або ж за нашою логікою буде створений новий запис в таблиці basket_product. Функцію, що буде відповідати за даний функціонал, наведено на зображені 2.12.

```php
public function add(Request $request, $id)
{
    $basket_id = $request->cookie('basket_id');
    $quantity = $request->input('quantity') ?? 1;
    if (empty($basket_id)) {
        $basket = Basket::create();
        $basket_id = $basket->id;
    } else {
        $basket = Basket::findOrFail($basket_id);
        $basket->touch();
    }
    if ($basket->products->contains($id)) {
        $pivotRow = $basket->products()->where('product_id', $id)->first()->pivot;
        $quantity = $pivotRow->quantity + $quantity;
        $pivotRow->update(['quantity' => $quantity]);
    } else {
        $basket->products()->attach($id, ['quantity' => $quantity]);
    }
    return back()->withCookie(cookie('basket_id', $basket_id, 525600));
}
```

Рисунок 2.12 – Реалізація функціоналу по додаванні товару в корзину

Як можна побачити з цього рисунку, у функцію add, передається системна змінна request та ідентифікаційний номер продукта. Після цього, ми отримуємо дані що знаходяться в куках сайту, та перевіряємо кількість товару який повинен додатись у корзину, і якщо вона не задана то буде дорівнювати одному. Далі проводимо перевірку, чи існують дані куки з ідентифікаційним номером корзини, якщо ж ні, ми створюємо нову корзину та копіюємо її номер

в нові куки. Коли ж куки існують, це означає що наша корзина існує і ми отримуємо з бази даних існуючу таблицю. Коли вже корзина існує або ж заново створена, ми робимо перевірку, чи даний продукт був доданий до корзини, тобто чи створені нові записи у таблиці basket_product з відповідними індексами корзини та продукту. Якщо ж даний продукт доданий, то ми збільшуємо його кількість, в іншому ж випадку, ми просто додаємо його до корзини та вказуємо кількість, що була передана з користувацької сторони.

Тепер коли корзина існує, можна реалізувати її вивід на сторону користувача, а саме передати існуючі дані самої корзини з бази даних на сторінку. Функціонал отримання та виводу даних буде реалізований за допомогою коду, зображеного на рисунку 2.13.

```php
public function index(Request $request)
{
    $basket_id = $request->cookie('basket_id');
    if (!empty($basket_id)) {
        $products = Basket::findOrFail($basket_id)->products;
        return view('basket.index', compact('products'));
    } else {
        return view('basket.index', ['products' => '']);
    }
}
```

Рисунок 2.13 – Реалізація функціоналу виводу товару корзини

Як можна побачити з даного рисунку, спершу ми отримує ідентифікаційний номер корзини з куків, після того виконуємо перевірку, чи пусті ці дані чи ні. Коли ж куки не пусті, та вміщують в собі номер корзини, ми отримуємо всі додані продукти у корзину, після чого передаємо їх на сторінку корзини. Коли ж куки пусті, це означає що корзина пуста, і ми на сторінку корзини передаємо пусту змінну у який би мали міститись продукти, та після перевірок вже на самій сторінці, буде виведено напис, що корзина пуста.

Наступний етапом реалізації функціоналу корзини, буде створення логіки, по зміні кількості конкретного продукту, тобто додавання однієї одиниці чи видалення. Спершу розглянемо код на рисунку 2.14, який відповідає за додавання однієї одиниці продукту до корзини.

```php
public function plus(Request $request, $id)
{
    $basket_id = $request->cookie('basket_id');
    if (empty($basket_id)) {
        abort(404);
    }
    $this->change($basket_id, $id, 1);
    return redirect()
        ->route('basket.index')
        ->withCookie(cookie('basket_id', $basket_id, 525600));
}
```

Рисунок 2.14 – Реалізація функціоналу збільшення на одну одиницю товару

Спершу, ми отримуємо дані з куків, а саме ідентифікатор корзини. Тоді робимо перевірку, чи куки пусті чи ні. Якщо ж так, то буде переадресація на сторінку з помилкою. У іншому випадку, ми збільшуємо кількість вибраного товару на одну позицію за допомогою створеної функції change, код якої зображено на рисунку 2.15, та повертаємось назад на сторінку корзини, одразу повертаючи і куки, щоб вони існували далі.

```php
private function change($basket_id, $product_id, $count = 0)
{
    if ($count == 0) {
        return;
    }
    $basket = Basket::findOrFail($basket_id);
    if ($basket->products->contains($product_id)) {
        $pivotRow = $basket->products()->where('product_id', $product_id)->first()->pivot;
        $quantity = $pivotRow->quantity + $count;
        if ($quantity > 0) {
            $pivotRow->update(['quantity' => $quantity]);
            $basket->touch();
        } else {
            $pivotRow->delete();
        }
    }
}
```

Рисунок 2.15 – Код функції change

Дана функція, отримує в собі ідентифікаційні номери корзини та продукту і ще кількість, на яку товар повинен змінитись. Якщо число додатне, то ми збільшуємо кількість продукту, якщо від'ємне то навпаки, зменшуємо. Функція працює таким чином, що спершу, ми перевіряємо, чи існує даний продукт у корзині, тобто чи є запис з даним продуктом у БД. Якщо ж він є, то ми його отримуємо, після чого змінюємо його кількість. Коли кількість змінена, ми проводимо перевірку, чи кількість одиниць товару більша нуля, якщо так, то ми зберігає зміни які стосуються кількості. У зворотному випадку, коли кількість менше або дорівнює нулю, продукт буде видалений з корзини.

Аналогічно до додавання, створюємо і видалення товару по одній позиції за натискання. Код даної функції зображено на рисунку 2.16. З даного рисунку, можна побачити, що код практично ідентичний тому, що на рисунку 2.14. Основна відмінність полягає в тому, що ми замість додатного значення, передаємо від'ємне. Звідси у нас виходить, що кількість товару зменшується, і автоматично відбувається перевірка кількості продукту. Тоді ж коли в корзині один елемент продукта, та кількість стає менша одного, продукт видаляється з корзини.

```php
public function minus(Request $request, $id)
{
    $basket_id = $request->cookie('basket_id');
    if (empty($basket_id)) {
        abort(404);
    }
    $this->change($basket_id, $id, -1);
    return redirect()
        ->route('basket.index')
        ->withCookie(cookie('basket_id', $basket_id, 525600));
}
```

Рисунок 2.16 – Реалізація функціоналу видалення однієї одиниці товару

Ще у корзині реалізований функціонал, який дозволяє відразу видалити продукт, незважаючи на його кількість. Код даної функції наведено на рисунку 2.17.

```
public function remove(Request $request, $id)
{
    $basket_id = $request->cookie('basket_id');
    if (empty($basket_id)) {
        abort(404);
    }
    $this->change($basket_id, $id, - ($request->input('itemQuantity')));
    return redirect()
        ->route('basket.index')
        ->withCookie(cookie('basket_id', $basket_id, 525600));
}
```

Рисунок 2.17 – Реалізація функціоналу видалення товару

Даний функціонал, який зображений на рисунку 2.17, є аналогічний до функцій додавання та видалення однієї одиниці товару. Тут ж відмінність полягає в тому, що коли ми викликаємо метод change, ми передаєм йому повну кількість певного продукти та його інформацію. Слід зауважити, що кількість передається від’ємним значенням, щоб відбувся процес видалення. Далі, вибраний продукт видаляється, хід скрипта вже описаний вище.

Останньою реалізованою функцією корзини буде створення замовлення при натисканні на відповідну кнопку. Код даної функції зображено на рисунку 2.18.

```php
public function saveOrder(Request $request)
{
    $this->validate($request, [
        'name' => 'required|max:255',
        'email' => 'required|email|max:255',
        'phone' => 'required|max:255',
        'address' => 'required|max:255',
    ]);

    $basket_id = request()->cookie('basket_id');
    $basket = Basket::findOrFail($basket_id);
    $user_id = auth()->check() ? auth()->user()->id : null;
    $order = Order::create(
        $request->all() + ['amount' => $basket->getAmount(), 'user_id' => $user_id]
    );

    foreach ($basket->products as $product) {
        $order->items()->create([
            'product_id' => $product->id,
            'name' => $product->name,
            'price' => $product->price,
            'quantity' => $product->pivot->quantity,
            'cost' => $product->price * $product->pivot->quantity,
        ]);
    }
    Session::put('order_id', $order);
    Session::put('order_products', $basket->products);
    $basket->delete();
    return redirect()
        ->route('basket.success')
        ->with('success', 'Ваш заказ успішно оформлений');
}
```

Рисунок 2.18 – Реалізація функціоналу оформлення замовлення

Спершу, при натисканні на кнопку оформлення замовлення, користувача буде перенаправляти на сторінку з формою, у яку він повинен ввести свої дані. Після підтвердження введених даних, вони запукають метод saveOrder, який показано на рисунку 2.18. Введені користувачем дані, ми отримуємо зі змінної request, яка є системною, та проводимо їх перевірку. Після того, як все гараз, ми отримаємо з куків, ідентифікаційний номер корзини. За допомогою нього, ми отримаємо нашу сформовану корзину. Наступним кроком, ми проводимо перевірку, чи користувач, який оформляє замовлення, зареєстрований у системі та увійшов. Тепер створюємо саме замовлення в базу даних та зберігаємо його там. Слід зауважити і ще, що ми використовуємо функцію getAmount, яка розраховує загальну суму замовлення, з моделі Basket. Код даного методу наведено на рисунку 2.19.

```php
public function getAmount()
{
    $amount = 0.0;
    foreach ($this->products as $product) {
        $amount = $amount + $product->price * $product->pivot->quantity;
    }
    return $amount;
}
```

Рисунок 2.19 – Реалізація функціоналу підрахунку загальної вартості замовлення

Наступним кроком, ми створюємо дві сесії. У першу поміщається ідентифікаційний номер замовлення, а у другій усі продукти, з яких було сформоване замовлення. Коли вже замовлення оформлене, запускається метод success. Код даного метода зображено на рисунку 2.20. У ньому ми перевіряємо, чи існує сесія з номером замовлення. Коли ж вона існує то ми створюємо змінні в яких поміщаємо сам номер замовлення та усі продукти які є в ньому. Після цього ми переносимо користувача на сторінку, де можна побачити таблицю з оформленим замовленням, даними які ввів замовник та повідомленням про успішне оформлення. У іншому випадку, коли наприклад користувач випадково попав на цю сторінку, його автоматично буде перенаправляти на сторінку корзини.

```php
public function success(Request $request)
{
    if ($request->session()->exists('order_id')) {
        $order = $request->session()->pull('order_id');
        $products = $request->session()->pull('order_products');
        return view('basket.success', compact('order', 'products'));
    } else {
        return redirect()->route('basket.index');
    }
}
```

Рисунок 2.20 – Реалізація функціоналу виводу підтвердження оформленого замовлення

2.3.4 Реєстрація та вхід користувачів

Після того як ми створили корзину, перейдемо до створення реєстрації та входу в систему користувачів. В фреймворці Laravel, для цього вже практично все готово, залишається лише виконати команду:

– php artisan ui:auth.

Система автоматично створить такі контролери, як:

– RegisterController — відповідає за реєстрацію користувачів.

– LoginController — відповідає за вхід у систему.

– ForgotPasswordController — відправляє лист на відновлення пароля.

– ResetPasswordController — вміщує логіку для скидання пароля.

Ще у системі автоматично створюються шаблони з формами входу та реєстрації, які у подальшому можна використовувати, замість того, щоб створювати власні. Також у системі автоматично створяться роути, які будуть відповідати за перенаправлення на відповідні сторінки аутентифікації.

Тепер, звернемо увагу на адміністратора. Він такий самий користувач як і інші, але його потрібно виокремити. Для цього створимо нову міграцію, яка буде модифікувати таблицю з користувачами, додаючи нове поле за допомогою команди:

– php artisan make:migration alter_users_table --table=users.

Перейдемо до міграцій та прошимо потрібний код, як це показано на рисунку 2.21.

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->boolean('admin')->after('email')->default(false);
    });
}
```

Рисунок 2.21 – Робоча частина файлу 2022_03_28_102810_alter_users_table.php

Як можна побачити з рисунку 2.21, ми створюємо нове поле admin, яке буде розміщуватись після поля email. Новостворене поле буде мати тип boolean, тобто вміщувати в собі значення лише true або false. Після цього запускаємо міграцію також за допомогою командної стрічки.

Отже, коли таблиця модифікована, ми можемо для певних користувачів міняти значення admin на true, і це буде означати, що наш користувач адміністратор. Оскільки він також користувач, то вхід у систему буде відбуватись через форму, якою користуватимуться прості клієнти. Ще створимо за допомогою командної стрічки контролер, який буде відповідати за перенаправлення адміністратора на сторінку панелі керування за допомогою команди:

- php artisan make:controller Admin\IndexController –invokable.

Слід звернути увагу на те, що контролер містить в собі частину --invokable, це означає, що у ньому можна реалізувати лише один метод, і у ньому ми розмістимо код який буде відповідати за перенаправлення адміністратора но головну сторінку пенелі керування, яка буде доступна лише йому. Крім цього, ми проведемо модифікацію файла LoginController.php, у якому створимо метод authenticated, як це показано на рисунку 2.22.

```
protected function authenticated(Request $request, $user)
{
    $route = 'user.index';
    $message = 'Ви успішно увійшли в особистий кабінет';
    if ($user->admin) {
        $route = 'admin.index';
        $message = 'Ви успішно увійшли в кабінет керування';
    }
    return redirect()->route($route)
        ->with('success', $message);
}
```

Рисунок 2.22 – Метод authenticated у файлі LoginController.php

На рисунку 2.22, можна побачити перевірку чи є користувач адміністратором. Якщо перевірка пройдена, то адміністратор успішно увійде у систему та буде перенаправлений до панелі керування, в іншому випадку, коли користувач не володіє спеціальними правами, його перенесе на сторінку користувача.

2.3.5 Керування категоріями

Наступним кроком, буде реалізація функціоналу панелі адміністратора по керуванні категоріями. Весь функціонал буде складатись з перегляду конкретної категорії, додавання нової та редагування вже існуючої.

В першу чергу створимо контролер, який буде реалізовувати у собі такі методи як index, create, store, show, edit, update та destroy. Він буде створений через командну стрічку, за допомогою команди:

- php artisan make:controller Admin/CategoryController --resource --model=Models/Category.

З коду вище, можна побачити, що --resource вказує на те, що контролер буде створений на основі вже існуючої моделі, а --model=Models/Category вказує саме модель.

Після цього, перейдемо до реалізації вищеописаних методів. Першим буде index, який буде відповідати за показ усіх категорій. Код даного методу можна побачити на рисунку 2.23.

```
public function index()
{
    $roots = Category::roots();
    return view('admin.category.index', compact('roots'));
}
```

Рисунок 2.23 – Реалізація методу index

Як можна побачити з рисунку вище, ми отримуємо всі записи з категоріями, поміщаємо їх у змінну та передаємо вже до частини адміністратора, де вони будуть відображатись у таблиці та з ними можна буде взаємодіяти. Далі реалухуємо метод show, який буде реалізувати отримання і вивід тільки однієї категорії, це можна побачити на рисунку 2.24.

```php
public function show(Category $category)
{
    return view('admin.category.show', compact('category'));
}
```

Рисунок 2.24 – Реалізація методу show

Третьою функцією яку ми реалізуємо буде destroy. Вона буде відповідати за видалення обраної адміністратором категорії, яка вже не потрібна або не використовується. Також, буде проходити перевірки, мета якої полягає в тому, що адміністратор не може категорію, товар який належить до до неї. Все це можна побачити на рисунку 2.25.

```php
public function destroy(Category $category)
{
    if ($category->products->count()) {
        $errors[] = 'Неможна видалити категорію яка містить товар';
    }
    if (!empty($errors)) {
        return back()->withErrors($errors);
    }
    $category->delete();
    return redirect()
        ->route('admin.category.index')
        ->with('success', 'Категорія успішно видалена');
}
```

Рисунок 2.25 – Реалізація методу destroy

Далі перейдемо до функції яка реалізує створення нових категорій для нашого веб-ресурсу. Спершу, коли адміністратор натискає на кнопку яка

відповідає за створення нової категорії, активується метод create, який зображений на рисунку 2.26.

```php
public function create()
{
    $parents = Category::roots();
    return view('admin.category.create', compact('parents'));
}
```

Рисунок 2.26 – Реалізація методу destroy

Цей метод, перенаправляє адміністратора на сторінку де вже буде доступна форма, в яку вводяться дані нової категорії. Коли ж усі дані введені та натиснута кнопка для збереження, запускається метод store, код якого зображено на рисунку 2.27.

```php
public function store(Request $request)
{
    $this->validate($request, [
        'parent_id' => 'integer',
        'name' => 'required|max:100',
        'slug' => 'required|max:100|unique:categories,slug|alpha_dash',
        'image' => 'mimes:jpeg,jpg,png|max:5000'
    ]);
    $file = $request->file('image');
    if ($file) {
        $path = $file->store('catalog/category/source', 'public');
        $base = basename($path);
    }
    $data = $request->all();
    $data['image'] = $base ?? null;
    $dates = [
        "name" => $request->name,
        "slug" => $request->slug,
        "parent_id" => '0',
        "content" => $request->content,
        "image" => $base
    ];
    $category = Category::create($dates);
    return redirect()
        ->route('admin.category.show', ['category' => $category->id])
        ->with('success', 'Новая категория успешно создана');
}
```

Рисунок 2.27 – Реалізація методу store

У даному методі спешу відбувається перевірка введених у форму даних. Коли дані введені вірно, то ми поміщуємо їх у масив, який вже відправляємо у базу даних. Після чого адміністратора переносить на сторінку де відображаються усі записи категорій.

Аналогічно працює редагування категорії. Відмінності полягають в тому, що при натисканні на кнопку зміни, на сторінку з формою передаються дані обраної категорії які вже і поміщуються у відповідні їм поля. Далі адміністратор може проводити потрібні йому зміни та натиснути на кнопку зберегти. За збереження відповідає метод update, його код є практично аналогічний до коду метода store, який зображений на рисунку 2.24, окрім однієї стрічки, замість збереження, проводиться оновлення.

2.3.6 Робота з замовленнями та зміна його статуса

Далі, буде реалізований функціонал керування оформленими замовленнями та зміна їхнього статуса. Попередньо, створимо у таблиці orders, нове поле яке буде вміщувати в собі статус замовлення. Це робиться за допомогою командної стрічки та команди:

- php artisan make:migration alter_orders_table --table=orders.

Після виконання команди вище, буде створена міграція яка відповідає за оновлення таблиці з замовленнями, а саме створення нового, потрібного нам поля. У ній ми прописуємо усе що буде відноситись до даного нього, а саме тип, назву та де воно має знаходитись по порядку. Після чого запустимо міграцію для успішного проведення модифікації таблиці. Далі, додаємо в модель Order перелік статусів замовлень які будуть існувати, так як це показано на рисунку 2.28.

```
public const STATUSES = [
    0 => 'Нове',
    1 => 'Оброблено',
    2 => 'Оплачено',
    3 => 'Доставлено',
    4 => 'Завершено',
    5 => 'Відмінено',
];
```

Рисунок 2.28 – Список категорій у моделі Order

Наступним кроком, перейдемо у сам контроллер який відповідає за реалізацію функціоналу керування замовленнями. Першою реалізованою функцією буде отримання та передача замовлень з бази даних до частини адміністратора, де її можна буде побачити. Це зображено на рисунку 2.29.

```
public function index()
{
    $orders = Order::orderBy('id', 'asc')->paginate(5);
    $statuses = Order::STATUSES;
    return view('admin.order.index', compact('orders', 'statuses'));
}
```

Рисунок 2.29 – Отримання та передача замовлень

Як можна побачити з рисунка вище, у змінну orders ми отримуємо відсортований масив усіх замовлень та їхні статуси. Тоді ж усі ці підготовлені дані передаємо на сторінку, де вони будуть виведені.

Також, реалізуємо функціонал, який дозволить детально переглядати певне замовлення, яке було обрано адміністратором. Це можна побачити на рисунку 2.30.

```php
public function show(Order $order)
{
    $statuses = Order::STATUSES;
    return view('admin.order.show', compact('order', 'statuses'));
}
```

Рисунок 2.30 – Вивід усієї інформації замовлення

З вище зображеного рисунку можна побачити, що ми одразу отримуємо в змінній order масив з усім замовленням, а вже у змінну statuses отримуємо категорію і всі ці дані передаються на візуальну сторону проекту.

Останнім що буде реалізовано в даному контролері, це методи для редагування замовлення. Для цього буде створено дві функції, edit та update. У першій відбувається отримання редагованого замовлення та поміщення у форму для редагування на спеціальній для цього сторінці. У другій ж буде отримання всіх даних з форми та оновлення запису у базі даних. Код функції update можна побачити на рисунку 2.31.

```php
public function update(Request $request, Order $order)
{
    $order->update($request->all());
    return redirect()
        ->route('admin.order.show', ['order' => $order->id])
        ->with('success', 'Заказ был успешно обновлен');
}
```

Рисунок 2.31 – Оновлення даних замовлення

При розгляді рисунку 2.31 можна побачити, що спершу ми отримуємо дані з форми, а саме новий статус, дані користувача та адреса доставки товару. Після чого при натисканні на кнопку зберегти, дані будуть отримані в методі та оновлені в базі даних з переадресацією на детальний перегляд даного замовлення.

2.3.7 Керування користувачами

Далі перейдемо до реалізації функціоналу по керуванні зареєстрованими користувачами. У першу чергу створимо новий контролер у якому і буде прописуватись увесь код. Як і раніше, це буде реалізовано за допомогою командної стрічки та команди:

– php artisan make:controller Admin/UserController --resource --model=User.

При розгляді команди вище, можна зрозуміти, що створюється новий контролер для сторони адміністратора, на основі вже існуючої моделі user. Однією з перших реалізованих функцій буде вивід усіх користувачів на сторінці, код реалізації можна побачити на рисунку 2.32.

```
public function index()
{
    $orders = Order::all();
    $users = User::paginate(5);
    return view('admin.user.index', compact('users', 'orders'));
}
```

Рисунок 2.32 – Реалізація виводу усіх користувачів

Як можна побачити з рисунка вище, ми отримуємо усіх користувачів, далі розбиваємо їх таким чином, щоб на одній сторінці розміщувалось не більше п’яти та передаємо усі ці підготовлені дані на сторінку. А саме буде виведено індивідуальний номер користувача, ім’я та фамілія, електронна пошта і кількість оформлених користувачем заказів.

Далі перейдемо до редагування користувача. Для цього нам потрібно створити три функції, а саме edit, update та validator. Перша використовується для переадресації на форму зміни імені, електронної пошти та пароля. Слід зауважити що у поля з іменем та електронною адресою попередні дані

вставляються автоматично, а поля для зміни пароля будуть доступні лише тоді коли активований відповідний чекбокс.

Коли ж усі дані для оновлення введені, активовано чекбокс для пароля і введено новий пароль та його підтвердження, можна перейти до оновлення. У цей момент активуватиметься метод update. Спершу у ньому відбувається відправка даних на перевірку у метод validator, до якого ми повернемося згодом. Коли перевірка пройдена успішно, ми перевіряємо чи потрібно змінювати пароль. У випадку коли пароль змінюється на новий, то першим чином ми створюємо хеш пароля, далі усі введені дані та отриманий хеш оновлюємо для потрібного записа у базі даних. Коли пароль не потрібно змінювати, то до БД відправляється запит на оновлення всіх даних окрім самого пароля. Після цього відбувається відправка адміністратора на сторінку з усіма користувачами. Код даного методу наведено на рисунку 2.33.

```
public function update(Request $request, User $user)
{
    $this->validator($request->all(), $user->id)->validate();
    if ($request->change) {
        $request->merge(['password' => Hash::make($request->password)]);
        $user->update($request->all());
    } else {
        $user->update($request->except('password'));
    }
    return redirect()
        ->route('admin.user.index')
        ->with('success', 'Данные пользователя успешно обновлены');
}
```

Рисунок 2.33 – Оновлення даних користувача

Повернемося до методу, у якому відбувається перевірка даних. У ньому проходить перевірка імені та електронного адреса за відповідними правила, також чи зареєстрований користувач з такою самою електронною поштою та якщо відбувається зміна пароля, також його перевірка за відповідними критеріями. Після чого повертається результат перевірки. Спосіб реалізації валідатора можна побачити на рисунку 2.34.

```php
private function validator(array $data, int $id)
{
    $rules = [
        'name' => [
            'required',
            'string',
            'max:255'
        ],
        'email' => [
            'required',
            'string',
            'email',
            'max:255',
            'unique:users,email,' . $id . ',id',
        ],
    ];
    if (isset($data['change_password'])) {
        $rules['password'] = ['required', 'string', 'min:8', 'confirmed'];
    }
    return Validator::make($data, $rules);
}
```

Рисунок 2.34 – Реалізація валідатора

Як можна побачити з рисунку вище, код валідатора виконує перевірку імені, електронної пошти та паролю.

2.3.8 Створення функціоналу адаптивного банера

Останньою функцією яка буде реалізована – це створення функціоналу для того, щоб банер на головній сторінці сайту був адаптивний, тобто адміністратор міг динамічно змінювати його фотографії. Для цього, створимо нову модель та міграцію для неї. Як і раніше, це виконується за допомогою командної стрічки, а команда буде виглядати таким чином:

- php artisan make:model -m Slider.

Після цього перейдемо до самого міграції та пропишемо потрібні поля як це показано на рисунку 2.35.

```php
public function up()
{
    Schema::create('sliders', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
}
```

Рисунок 2.35 – Міграція для таблиці sliders

Коли міграція готова, запускаємо її за допомогою командної стрічки та відповідної команди, аналогічно до того як ми виконували раніше. Наступним кроком буде створення контролера у якому вже і буде реалізований сам функціонал. Це виконується за допомогою відповідної команди:

- php artisan make:controller Admin/SliderController.

Сам функціонал буде працювати таким чином, що адміністратор буде мати змогу обирати потрібні файли з власного комп'ютера, після чого завантажувати їх у файлову систему проекту а назви файлів будуть зберігатись у таблиці бази даних. У даному контролері буде реалізовано три метода. У першому ми отримуємо усі назви завантажених файлів з БД, за допомогою яких ми реалізуємо посилання на сам файл у системі. Саму реалізацію даного метода можна побачити на рисунку 2.36.

```php
public function index()
{
    $images = Slider::all();
    return view('admin.slider.index', ['images' => $images]);
}
```

Рисунок 2.36 – Отримання даних про завантажені фото

Наступним кроком буде реалізація збереження самих фотографій. Це буде зроблено за допомогою форми. Спершу адміністратор буде обирати потрібне зображення, після чого натиснути на кнопку для збереження. Тоді буде запускатись метод store, реалізація якого показана на рисунку 2.37.

```php
public function store(Request $request)
{
    $file = $request->file('image');
    if ($file) {
        $dest = public_path('/storage/slider');
        $image_name = time() . '.' . $file->getClientOriginalName();
        $file->move($dest, $image_name);

        $datas = [
            'name' => $image_name
        ];
        $slider = Slider::create($datas);
    }
    $images = Slider::all();
    return view('admin.slider.index', ['images' => $images]);
}
```

Рисунок 2.37 – Реалізація збереження фотографій

Розглянувши зображення 2.37 можна побачити, що першим кроком ми отримує сам файл який завантажив адміністратор та поміщаємо його у змінну. Тоді ми перевіряємо чи дана змінна пуста. У випадку якщо змінна пуста, це може означати що натискання на кнопку збереження було випадковим і одразу відбудеться повернення на сторінку з усіма завантаженими фотографіями. Коли ж, зображення було завантажене, ми прописуємо шлях де повинен зберігатись файл. Після цього ми генеруємо індивідуальну назву файла та зберігаємо файл. Коли сам файл вже збережений, ми створює новий запис в таблиці slider, у якому буде зберігатись інформація про завантажене зображення.

Останнім методом який буде реалізований – це видалення вже завантаженого зображеного. За це буде відповідати метод destroy, код якого можна побачити на рисунку 2.38.

```php
public function destroy(Slider $image, Request $request)
{
    $image_path = "storage/slider/" . $request->name;
    if (File::exists($image_path)) {
        File::delete($image_path);
    }
    $image->where('id', $request->id)->delete();
    return redirect()
        ->route('admin.slider.index');
}
```

Рисунок 2.38 – Реалізація видалення зображення

Як можна побачити з зображення вище, при натисканні на кнопку видалення, спершу ми отримуємо розміщення файлу та поміщуємо цю інформацію у змінну. Тоді ми перевіряємо чи даний файл у певній директорії існує та видаляємо його. В кінці вже відбувається видалення інформації з бази даних про зображення та повернення на сторінку керування слайдером у панелі адміністратора.

2.4 Модернізація дизайну інтерфейсу

Після проведеного аналізу технічного завдання модернізація починається зі створення макетів дизайну інтерфейсів у вибраному середовищі редизайну Figma. Розробка починається з редизайну головної сторінки, яка буде відображатися клієнту при першому вході на сайт, а також основних елементів, які будуть відображатися на більшості сторінках [23].

2.4.1 Редизайн головної сторінки та основних елементів.

Аналізуючи теперішню головну сторінку (Див. рисунок 2.39) прийнято рішення спростити її дизайн, зменшити кількість елементів, які такі, як “Новини” у нижньому лівому куті, модернізувати хедер, зменшивши його розмір і видалити зі сторінки зображення, які не несуть ніякої важливої інформації про продукцію компанії [24].

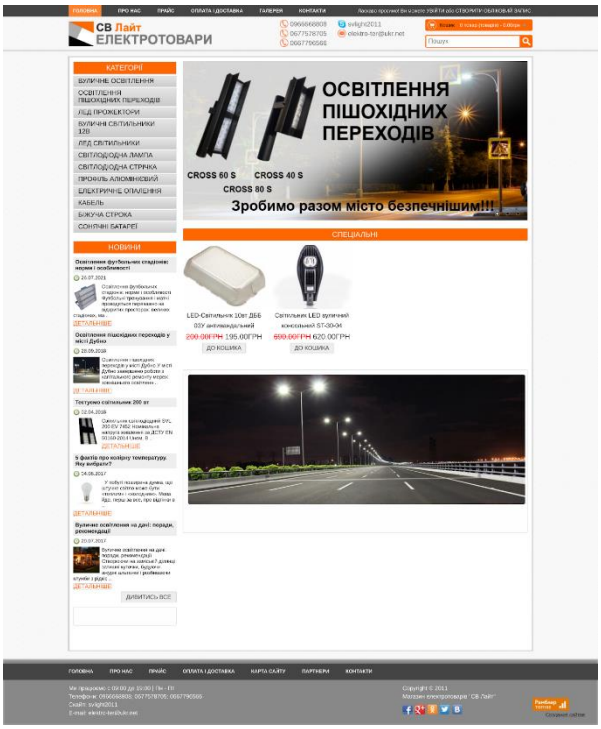

Рисунок 2.39 – Теперішня версія головної сторінки

Перед тим, як спроектувати повноцінно оновлену головну сторінку, необхідно поелементно розібрати теперішню версію, виділити основні елементи, які будуть повторюватися і оновити їхній вигляд. Серед таких елементів є:

- блок header;
- блок footer;
- блок з відображенням списку категорій.

Теперішня версія блоку header (Див. рисунок 2.40) перевантажена інформацією, яка дублюється у блоці footer, а також містить контраст у кольорах, що може негативно вплинути на враження клієнтів [25].

Рисунок 2.40 – Теперішня версія блоку header

Оновлена версія боку header (Див. рисунок 2.41) дотримується принципів мінімалізму і кольорову палітру попереднього дизайну.

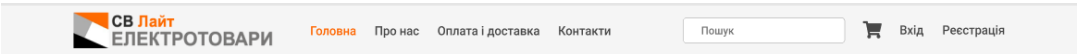

Рисунок 2.41 – Оновлена версія блоку header.

Блок footer також потребує значної модернізації, оскільки його вміст перевантажений елементами (Див. рисунок 2.42), які вважаються застарілими у сучасному веб-дизайні, а саме: карта сайту і лічильник відвідувань. Також іконки соцмереж є різнокольоровими, що порушує правило збереження кольорової палітри.

Рисунок 2.42 – Теперішня версія блоку footer.

Модернізований блок footer (Див. рисунок 2.43) зберіг основну інформацію про компанію, а також важливі елементи. Після оновлення лічильник відвідувань не відображається на сторінках, а іконки соціальних мереж дотримуються візуальної палітри всього блоку.

Рисунок 2.43 – Оновлена версія блоку footer

Блок зі списком категорій (Див. рисунок 2.44) у теперішній версії не є мінімалістично оформленим, і містить значний контраст, що впливає на враження клієнтів веб-сторінки, а також не відповідає сучасним принципам веб-дизайну.

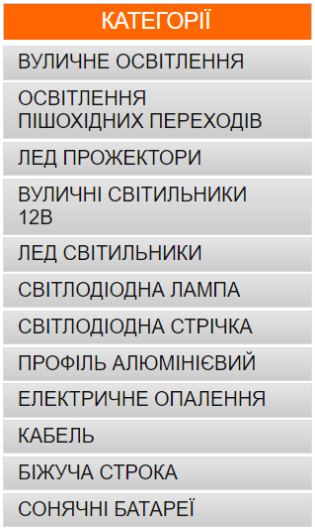

Рисунок 2.44 – Теперішня версія блоку зі списком категорій

Оновлена версія блоку категорій (Див. рисунок 2.45), не містить заголовку “Категорії”, який в даному блоці є лишнім елементом, а також відповідає мінімалістичному вигляду.

Рисунок 2.45 – Оновлена версія блоку зі списком категорій

Після редизайну всіх основних елементів, спроектовано макет головної сторінки в середовищі Figma (Див. рисунок 2.46). Вигляд головної сторінки змінився на мінімалістичний та сучасний. Збереглися основна палітра кольорів (оранжевий, темно-сірий та світло-сірий), розміщення елементів, а також логотип.

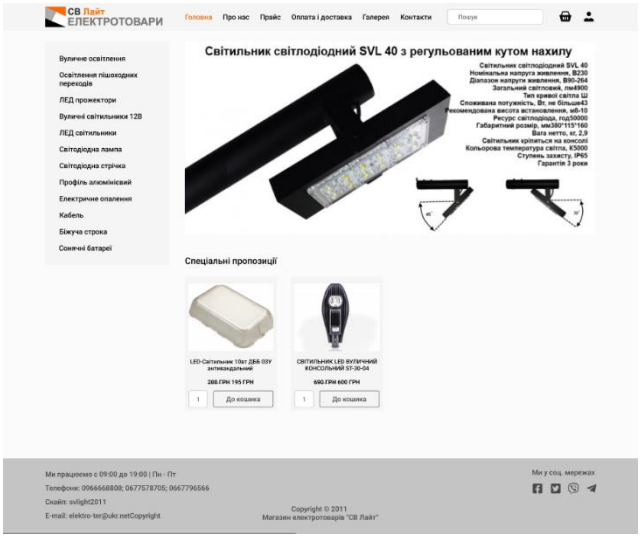

Рисунок 2.46 – Оновлена версія головної сторінки

Зі значних відмінностей є те, що видалено елементи, які не відповідають сучасним вимогам дизайну веб-сторінок, а також застосовані зміни до основних елементів, які описані вище. Також на головні сторінці змінилися також такі елементи, як карусель, товари зі спеціальними пропозиціями, а також в цілому сторінка стала менш навантаженню.

2.4.2 Редизайн сторінки товару

Сторінка товару в теперішньому варіанті (Див. рисунок 2.47) не потребує великих змін, але за рахунок того, що секція “Характеристики” не використовувалася часто (більшість товарів мають не заповнені дані з характеристиками і всі дані про товар вказувалися у описі), прийнято рішення видалити з функціоналу дану секцію і зосередитися на інших елементах сторінки. Загалом сторінка з товаром зберегла основний стиль головної сторінки.

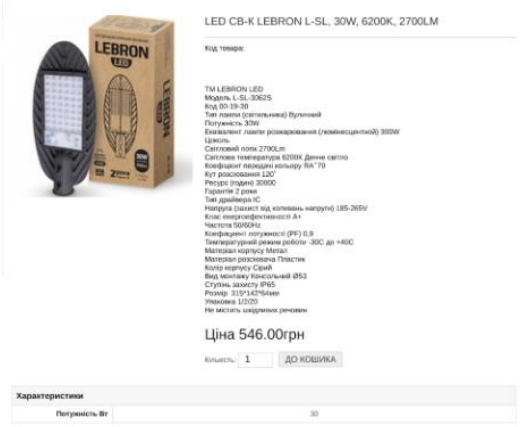

Рисунок 2.47 – Теперішня сторінка з товаром

В оновленій версії сторінка товару (Див. рисунок 2.48) запозичує кольорову палітру сайту додаючи світліші тона.

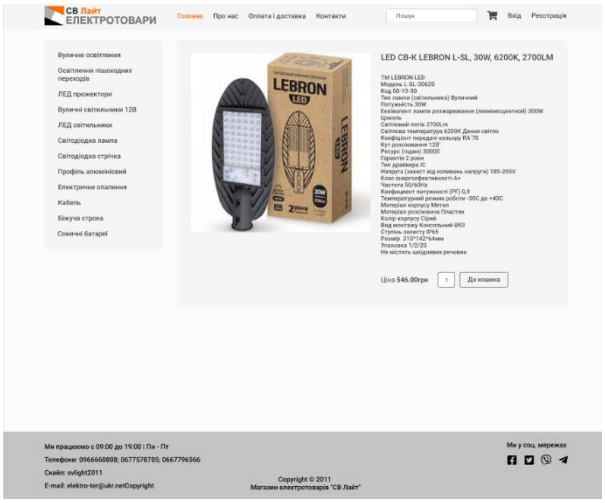

Рисунок 2.48 – Оновлена сторінка з товаром

А також змінився стиль форми для додавання продукту у кошик і шрифти опису продукту.

2.2.3 Редизайн сторінки зі списком товарів

Перед оновленням всієї сторінки зі списком товарів, необхідно спроектувати макет з виведенням товару на даній сторінці. Для теперішньої версії виведення товару на сторінці зі списком товарів (Див. рисунок 2.49) необхідно модернізувати дизайн.

Рисунок 2.49 – Теперішній вигляд елементу з товаром

В оновленій версії (Див. рисунок 2.50) прийнято рішення удосконалити форму додавання до кошика добавивши кількість товарів.

Рисунок 2.50 – Оновлений вигляд елементу з товаром

Теперішня сторінка зі списком товарів (Див. рисунок 2.51) при переході на обрану категорію містить велику кількість елементів, які необхідно вирізати з дизайну у оновленій версії. Прийнято рішення не реалізовувати функціонал картинки для категорії, оскільки картинка займає велику кількість простору на сторінці, тим самим надати більшого розміру для товарів.

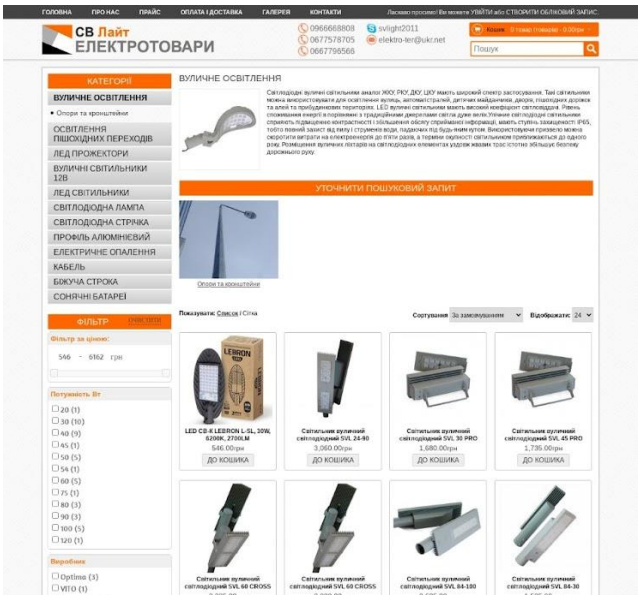

Рисунок 2.51 – Теперішня версія вигляду сторінки зі списком товарів

В оновленій версії зі списком товарів застосовуємо зміни до вигляду товарів у списку, які описані вище (Див. рисунок 2.52).

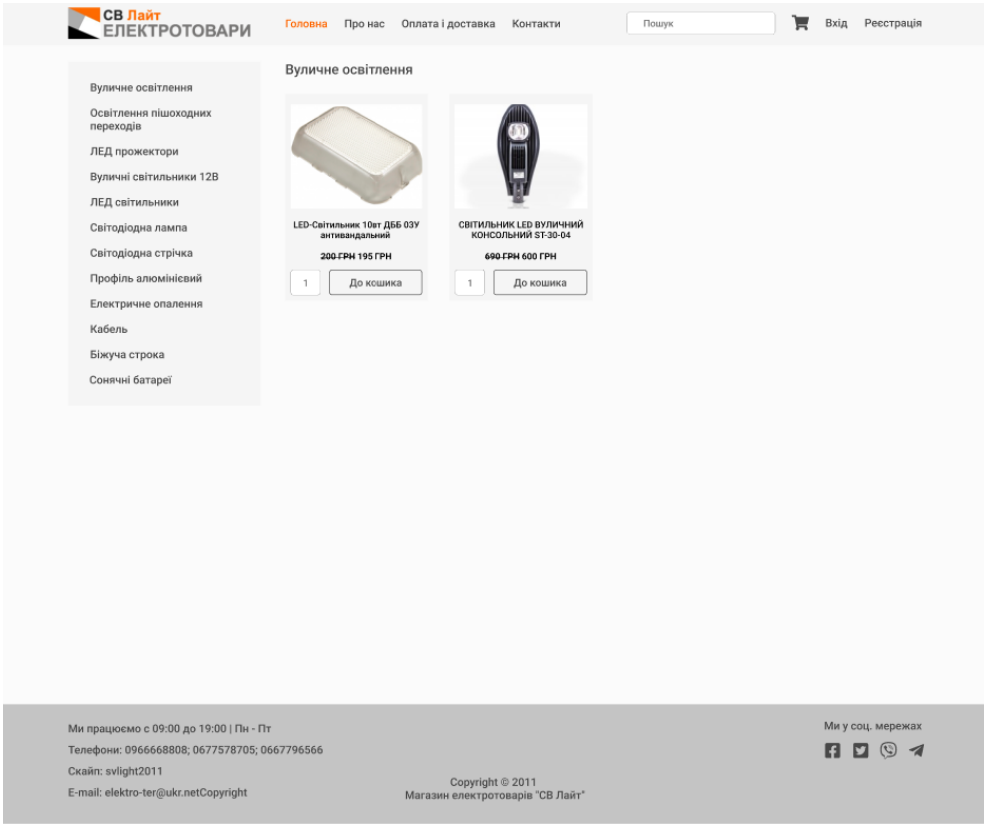

Рисунок 2.52 – Оновлена версія вигляду сторінки зі списком товарів

Також сторінка стала більш візуально простора та збереглася кольорова палітра.

2.4.4 Редизайн сторінки “Про нас”

Теперішня версія сторінки “Про нас” (Див. рисунок 2.53) також потребує модернізації, а саме спрощення. На даний момент сторінка містить велику кількість фотографій, які не демонструють інформативності про компанію. Прийнято рішення залишити інформацію про місцезнаходження магазину компанії з фотографією і текст.

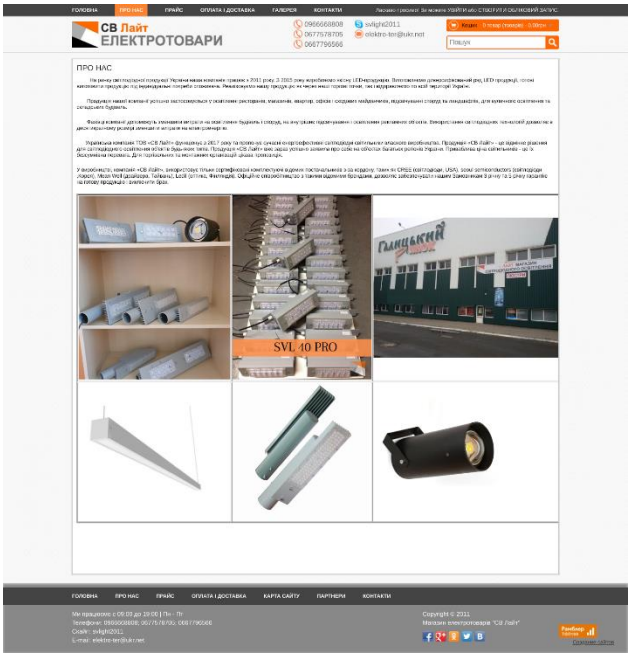

Рисунок 2.53 – Теперішня версія сторінки “Про нас”

Оновлена версія, як описано вище, зберегла у собі основний текст про компанію, її місцезнаходження, графік роботи, а також фотографію з місцезнаходженням магазину (Див. рисунок 2.54).

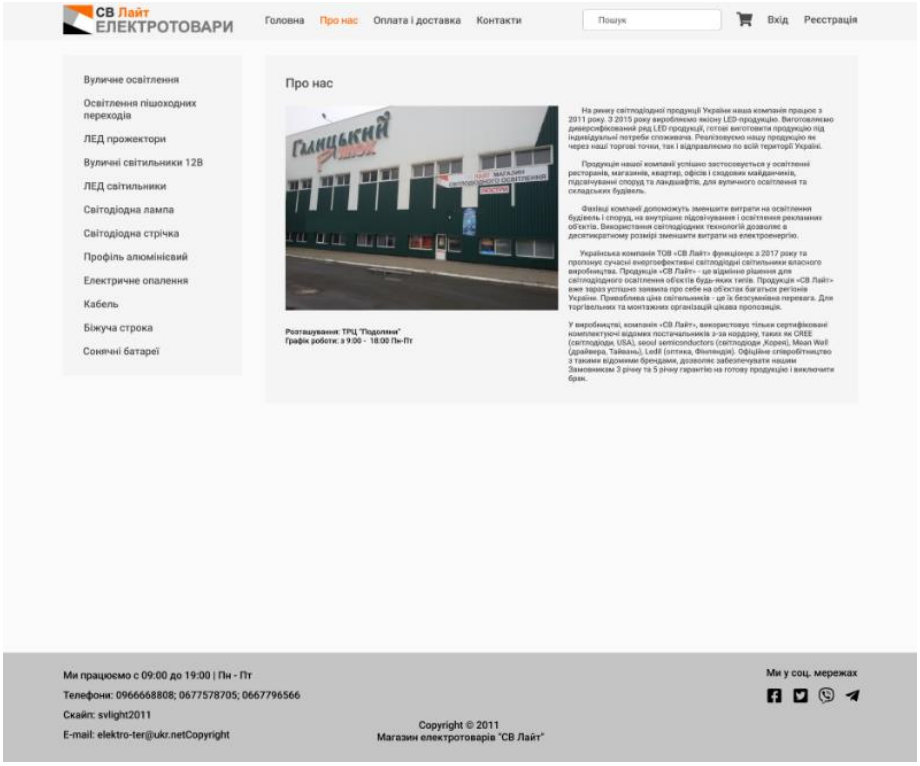

Рисунок 2.54 – Оновлена версія дизайну сторінки “Про нас”

В модернізованому інтерфейсі не містяться зображення, які не несуть інформації саме про компанію.

2.4.5 Редизайн сторінки кошика

Сторінка з відображенням товарів у кошику потребує незначних змін у виводі списку товарів, та покращень в оформленні розміщення елементів на сторінці (Див. рисунок 2.55).

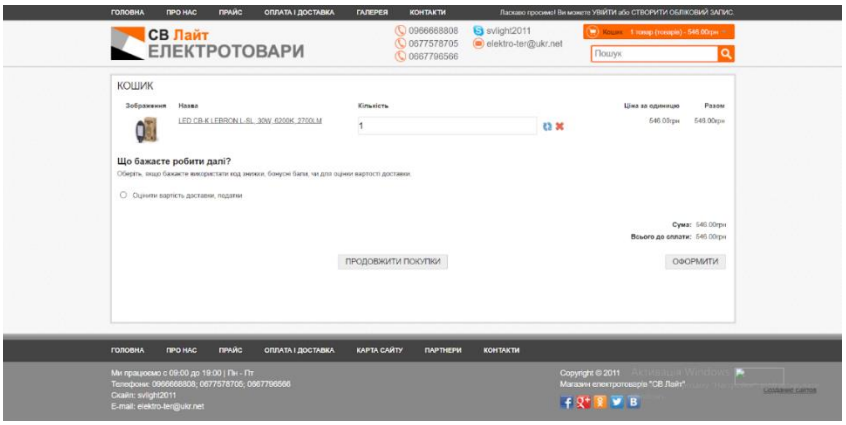

Рисунок 2.55 – Теперішня версія дизайну кошика

В оновленій версії сторінки кошику з товарами прийнято рішення використовувати звичайну таблицю з товарами, та спростити саму структуру оформлення замовлення (Див. рисунок 2.56).

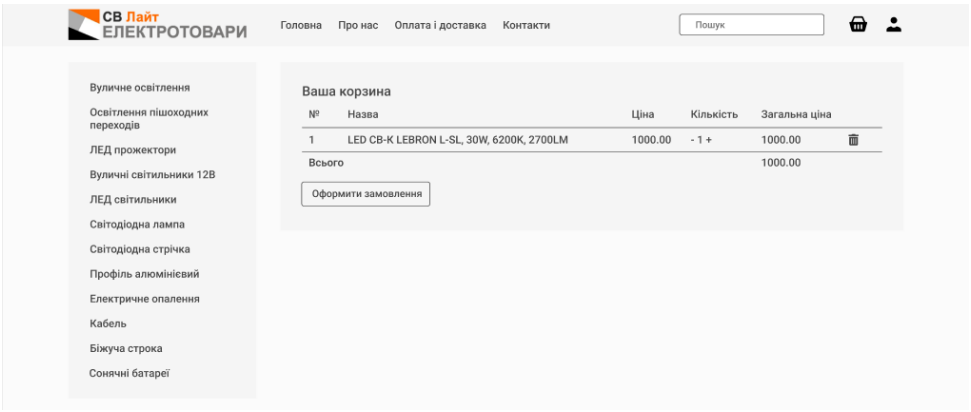

Рисунок 2.56 – Оновлена версія дизайну кошика

Як можна побачити з рисунку вище, сторінка більш простіша та зрозуміліша для користувача.

2.4.6 Розробка фронтенду

Після етапу з редизайном основних сторінок у середовищі редизайну Figma слід перейти до етапу розробки фронтенд-частини. На даному кроці необхідно максимально відтворити вигляд інтерфейсу у коді та сторінці у браузері. Так як всі сторінки у фреймворці Laravel поділені на шаблони, то роботу над безпосередньо розробкою можна також поділити поетапно по шаблонам.

Перед тим як описувати основні елементи, необхідно описани блок head, який буде міститься у кожній html-сторінці, даний блок відповідає здебільшого за інформацію про сторінку для браузера, а саме: назву сторінки, посилання на css-файли та js-файли, іконки, та інші метадані.

У блоці head (Див. рисунок 2.57) задано опис сторінки з імпортом необхідний бібліотек для роботи фреймворка Bootstrap 5, а також кодуванням і назвою сторінки.

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>{{ $title ?? "Інтернет магазин" }}</title>
  <link rel="stylesheet" href="{{ asset('css/app.css') }}">
  <link rel="stylesheet" href="{{ asset('css/style.css') }}">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW6ZMFYlzcLA8Nl
  <script src="{{ asset('js/app.js') }}"></script>
</head>
```

Рисунок 2.57 – Код блоку head

Після опису блоку head слідує опис блоку body, в якому і описуються вигляд сторінки, та її основні елементи. Основні елементи, які відображаються на кожній сторінці записані в головний шаблонний файл, який буде

наслідуватися кожною сторінкою, до таких елементів відносяться вищеописані: блок header, блок footer та блок зі списком категорій, їх слід описати в головному шаблоні.

Спочатку опишемо header, або іншими навігацію (Див. рисунок 2.58). В навігації містяться: логотип посилання на сторінки, форма пошуку, а також посилання на сторінки з входом в особистий кабінет та реєстрації.

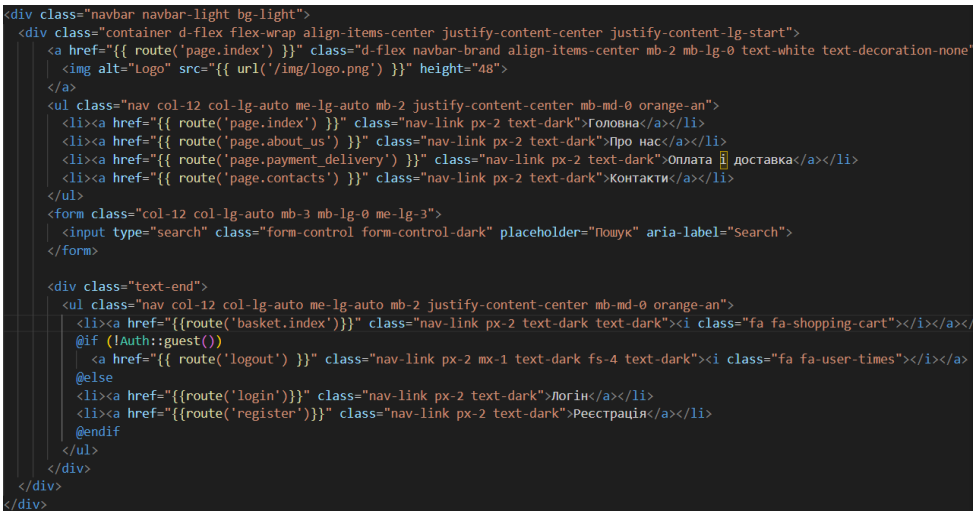

Рисунок 2.58 – Код блоку навігації

Після навігації описано контейнер сайту (Див. рисунок 2.59). Контейнер у фреймворці Bootstrap 5 використовується для відображення на сторінці основного контенту, а також його відображення буде завжди по центру сторінки. У ньому записуємо блок з відображенням категорій та блок який буде відображати основний контент в залежності від заданого роута.

Рисунок 2.59 – Код контейнера

В кінці сторінки відображатиметься footer-блок (Див. рисунок 2.60), який міститиме інформацію про контакти з компанією, копірайт, а також посилання на соціальні мережі.

```
<footer class="text-center text-lg-start bg-light text-muted mt-2">
<section class="d-flex justify-content-center justify-content-lg-between">
  <div class="container text-center text-md-start mt-1">
    <div class="row mt-3">
      <div class="col-md-3 col-lg-4 col-xl-3">
        <p>
          Ми працюємо с 09:00 до 19:00 | Пн - Пт
          Телефони: 0966668808; 0677578705; 0667796566
          Скайп: svlight2011
          E-mail: elektro-ter@ukr.net
        </p>
      </div>
      <div class="offset-md-8 offset-lg-6 offset-xl-6 col-md-4 col-lg-3 col-xl-3">
        <p class="mb-1">
          Ми у соц. мережах
        </p>
        <p>
          <a href="#" class="mx-1"><i class="fa fa-facebook-square text-muted fs-5"></i></a>
          <a href="#" class="mx-1"><i class="fa fa-twitter-square text-muted fs-5"></i></a>
          <a href="#" class="mx-1"><i class="fa fa-telegram text-muted fs-5"></i></a>
        </p>
      </div>
    </div>
  </div>
</section>
<div class="text-center p-2">
  <p>© Copyright © 2011</p>
  <p>Магазин електротоварів "СВ Лайт"</p>
  </div>
</footer>
```

Рисунок 2.60 – Код блоку footer

В подальшому вищеописані елементи будуть відображатися на кожній сторінці у браузері, оскільки фреймворк імпортуватиме їхню структуру разом з тим контентом, який описаний у інших файлах.

Для початку опишемо головну сторінку. Дана сторінка імпортує основні елементи і вставляє секцію з її контентом, в даному випадку це карусель з оголошеннями (Див. рисунок 2.61). Карусель написана за допомогою фреймворка Bootstrap, що спростило її реалізацію. Вона містить елементи перегортання зображень, як здебільшого зроблено на інших інтернет-магазинах.

```
@extends('layout.site', ['title' => 'Головна'])

@section('content')

<div class="row">
  <div id="carouselExampleDark" class="carousel carousel-light slide" data-bs-ride="carousel">
    <div class="carousel-indicators">
      @foreach($images as $key => $image)
      <button type="button" data-bs-target="#carouselExampleDark" data-bs-slide-to="{{$key}}" class="{{$key == 0 ? 'active': ''}}" aria

      @endforeach
    </div>
    <div class="carousel-inner">
      @foreach($images as $key => $image)
      <div class="carousel-item {{$key == 0 ? 'active' : '' }}" data-bs-interval="10000">
        <img src="\storage\slider\{{ $image->name }}" class="d-block w-100 h-100" alt="">
      </div>
      @endforeach
    </div>
    <button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleDark" data-bs-slide="prev">
      <span class="carousel-control-prev-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Previous</span>
    </button>
    <button class="carousel-control-next" type="button" data-bs-target="#carouselExampleDark" data-bs-slide="next">
      <span class="carousel-control-next-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Next</span>
    </button>
  </div>
</div>

@endsection
```

Рисунок 2.61 – Код головної сторінки

В результаті головна сторінка виглядатиме в браузері (Див. рисунок 2.62) подібною до того, що спроектовано в середовищі проектування інтерфейсів Figma, але з незначними змінами у масштабі елементів через відмінність у розмірах та розширенні екрану.

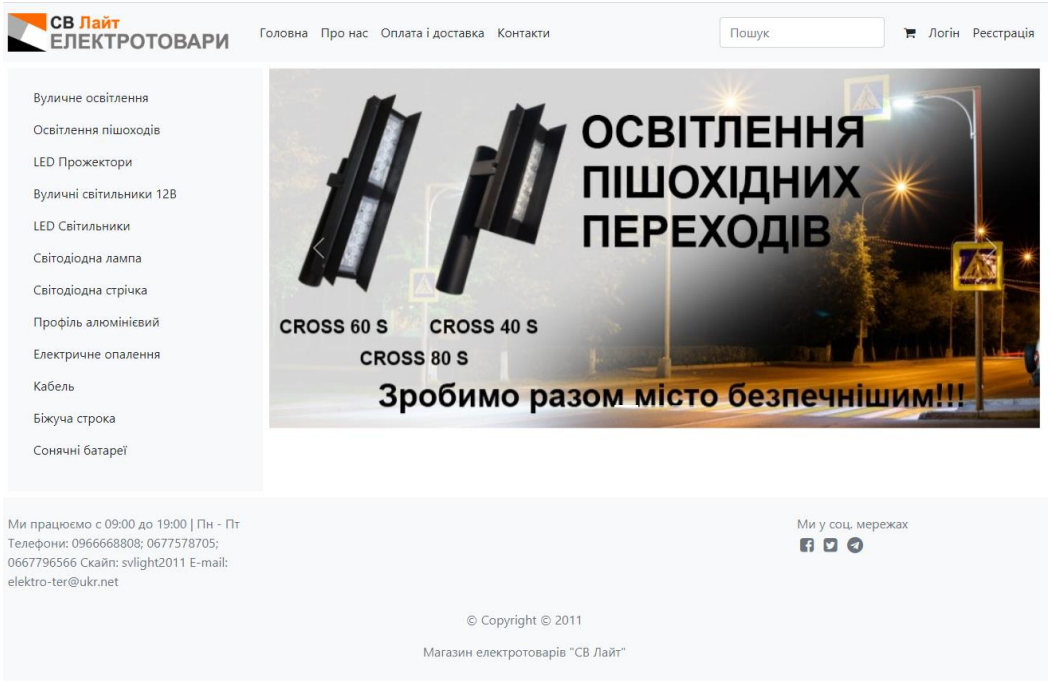

Рисунок 2.62 – Відображення у браузері головної сторінки

Після реалізації головної сторінки переходимо до етапу написання сторінки з виводом товарів категорії. Даний етап слід розпочати з написання структури товару (Див. рисунок 2.63). Для цього створимо шаблон, який буде дублюватися на сторінці з одинаковым дизайном, який реалізується за допомогою інструменту Blade. В шаблоні опишемо основний стиль, а саме: відображення картинки товару, назва товару та форма для замовлення товару.

```
<div class="col-md-3 text-center bg-light">
    <div class="m-2 p-2">
        <div>
            @php
            if ($product->image) {
            $url = url('/storage/product/' . $product->image);
            } else {
            $url = url('\storage\product\dafault.jpg');
            }
            @endphp
            <a href="{{ route('catalog.product', ['slug' => $product->slug]) }}"><img src="{{ $url }}" alt="" class="img-fluid"></a
        </div>
        <div class="p-2">
            <a href="{{ route('catalog.product', ['slug' => $product->slug]) }}" class="text-decoration-none text-dark fs-5">{{ $pro
        </div>
        <div>
            <form action="{{ route('basket.add', ['id' => $product->id]) }}" method="post" class="d-flex gap-2">
                @csrf
                <input type="text" name="quantity" id="input-quantity" value="1" class="form-control" placeholder="Кількість">
                <button type="submit" class="btn col-9 border-dark">У корзину</button>
            </form>
        </div>
    </div>
</div>
```

Рисунок 2.63 – Код з виводом товару у списку

Після опису структури товару, реалізовуємо відображення списку з товарами (Див. рисунок 2.64), в чому також допомагає інструмент Blade, на вхід якого приходить список товарів із бази даних з вибраної категорії, після чого Blade циклічно відображає на сторінці всі товари у вищенаведеному шаблоні.

```
@extends('layout.site')

@section('content')
<div class="row">
    <h2 class="m-3">{{ $category->name }}</h2>
    <div class="d-flex gap-2">
        @foreach ($category->products as $product)
            @include('catalog.part.product')
        @endforeach
    </div>
</div>
@endsection
```

Рисунок 2.64 – Код сторінки з відображенням списку товарів

В результаті браузер рендерить сторінку з товарами вибраної категорії (Див. рисунок 2.65).

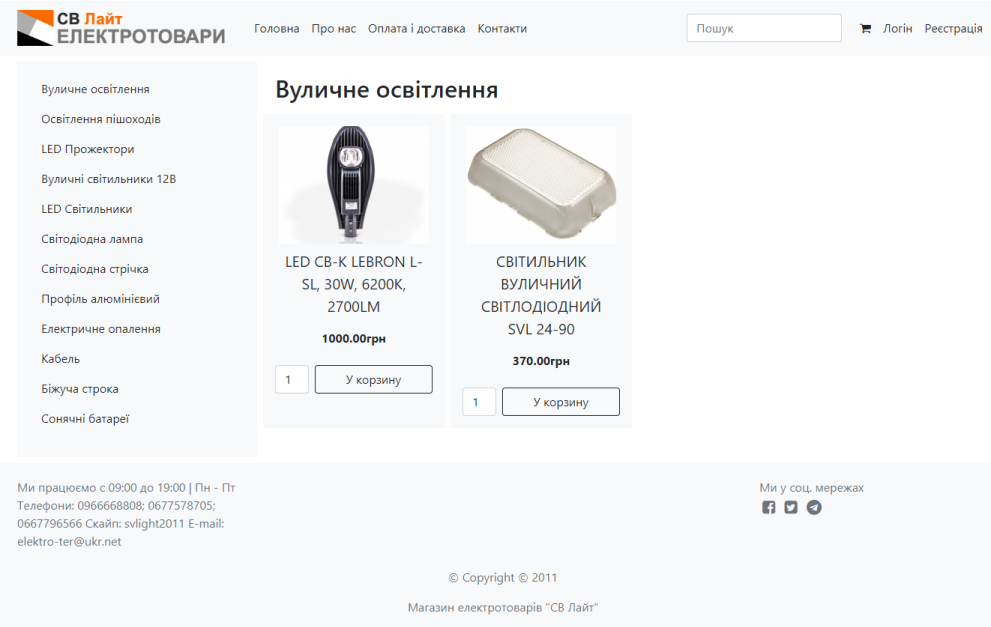

Рисунок 2.65 – Відображення у браузері списку товарів вибраної категорії

Дальше реалізовуємо сторінку детальною інформацією про товар (Див. рисунок 2.66). Вона також наслідує всю основну структуру за допомогою шаблонізатора Blade.

```
@extends('layout.site', ['title' => $product->name])

@section('content')
<div class="bg-light">
    <div class="row p-3">
        <div class="col-md-6">
            @php
            if ($product->image) {
                $url = url('/storage/product/' . $product->image);
            } else {
                $url = url('\storage\product\dafault.jpg');
            }
            @endphp
            <img src="{{ $url }}" class="img-fluid col-12">
        </div>
        <div class="col-md-6">
            <p class="fs-3">{{ $product->name }}</p>
            <span class="mt-4 mb-4 fs-7" style="white-space: pre">{{ $product->content }}</span>
            <form action="{{ route('basket.add', ['id' => $product->id]) }}" method="post" class="d-inline-flex gap-3 align-items-cente
                <span class="">ціна: {{ number_format($product->price, 2, '.', '') }}</span>
                @csrf
                <input type="text" name="quantity" id="input-quantity" value="1" placeholder="Кількість" class="form-control w-25">
                <button type="submit" class="btn border-dark">До кошика</button>
            </form>
        </div>
    </div>
</div>
</div>
@endsection
```

Рисунок 2.66 – Код з відображенням інформації про товар

В браузері сторінка з детальною інформацією про товар виглядатиме подібною до того, що спроектовано у середовищі Figma (Див. рисунок 2.67).

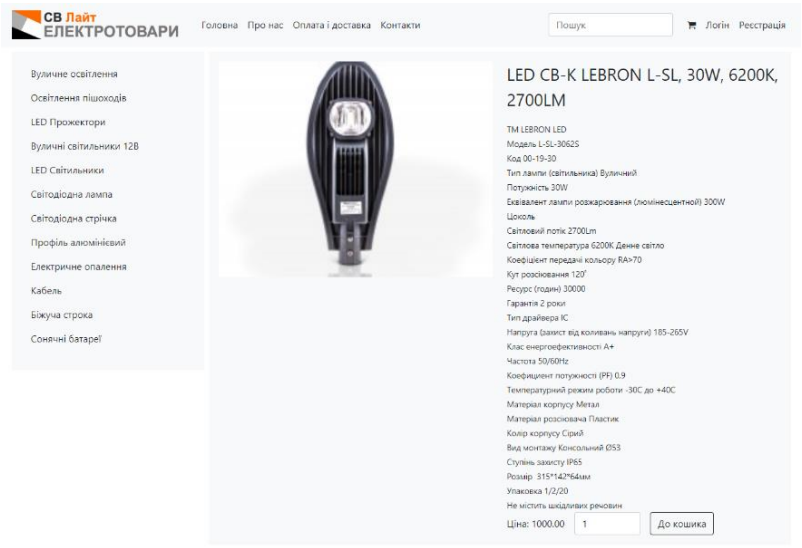

Рисунок 2.67 – Відображення у браузері інформації про вибраний товар

Дальше реалізовуємо сторінку на якій міститься детальна інформація про компанію (Див. рисунок 2.68). В даному випадку запишемо інформацію про компанію у коді.

```
@extends('layout.site', ['title' => 'Про нас'])

@section('content')

<div class="bg-light">
    <div class="row p-2">
        <div class="col-12 m-2">
            <h2 class="text-dark">Про нас</h2>
        </div>
        <div class="col-6">
            <img src="{{ url('/img/about_us.jpg') }}" class="col-12">
            <div class="mt-2 text-dark">
                <b class="fs-7">
                    Розташування: ТРЦ “Подоляни”<br>
                    Графік роботи: з 9:00 -  18:00 Пн-Пт
                </b>
            </div>
        </div>
        <div class="col-6 fs-7 text-dark" style="white-space: pre-wrap">
            На ринку світлодіодної продукції України наша компанія працює з 2011 року. З 2015 року виробляємо якісну LED-продукцію. Виг

            Продукція нашої компанії успішно застосовується у освітленні ресторанів, магазинів, квартир, офісів і сходових майданчиків,

            Фахівці компанії допоможуть зменшити витрати на освітлення будівель і споруд, на внутрішнє підсвічування і освітлення рекла

            Українська компанія ТОВ «СВ Лайт» функціонує з 2017 року та пропонує сучасні енергоефективні світлодіодні світильники власн

            У виробництві, компанія «СВ Лайт», використовує тільки сертифіковані комплектуючі відомих постачальників з-за кордону, таки
        </div>
    </div>
</div>

@endsection
```

Рисунок 2.68 – Код сторінки з інформацію про компанію

При відкритті у браузері сторінки з інформацією про компанією відображається та інформація, яка описана у коді, що є подібним до того, що спроектовано у середовищі Figma (Див. рисунок 2.69).

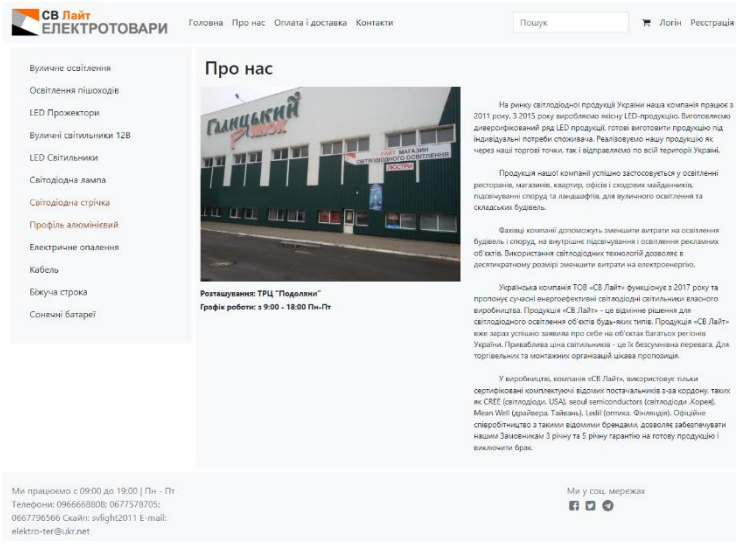

Рисунок 2.69 – Вигляд сторінки у браузері інформації про компанію

У результаті оновлення даної сторінки, користувачу представлена інформація про компанію у кращому вигляді.

2.5 Висновок до другого розділу

В даному розділі завершено оновлення інтерфейсу та розробку програмних компонент веб-сайту ТОВ ”СВ Лайт”. Серед основного:

- Оновлено ПЗ для роботи веб-сайту.
- Модернізовано дизайн інтерфейсу.
- Покращено роботу сайту.
- Переписано основний функціонал.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Роль центральної нервової системи в трудовій діяльності людини

Нервова система відіграє важливу роль у життєдіяльності людини, та її організму. Від неї залежить робота всіх внутрішніх органів, а також зв’язок із зовнішнім середовищем.

Центральна нервова система людини складається із центральної, яка включає головний і спинний мозок. На головний мозок діють різноманітні подразники із внутрішнього та зовнішнього середовища [26].

За функціями нервову систему поділяють на соматичну і вегетативну. Соматична нервова система регулює опорно-руховий апарат і всі органи чуття, а вегетативна - процес обміну речовин та роботу всіх внутрішніх органів (серця, нирок, легенів). Найпростіші рухи регулює спинний мозок. Довгастий мозок керує процесами травлення, дихання, кровообігу та іншими життєво важливими функціями. Підкіркова і кіркова частини головного мозку керують всією психічною діяльністю людини.

Центральна нервова система виконує рефлекторну, інтегративну та координаційну функції.

Рефлекторна діяльність мозку зумовлена безумовними та умовними рефлексами. Безумовні рефлекси є вродженими, мають велику стійкість і забезпечують пристосування організму до зовнішнього середовища. Умовні рефлекси набуваються залежно від обставин, розширюють діапазон пристосувальницьких можливостей організму і згасають, якщо потреби в них немає.

Стійка і злагоджена система умовних рефлексів формується у процесі навчання і забезпечує виконання певного виробничого завдання. Стійкість

системи умовних рефлексів може бути порушена при відхиленні трудової діяльності від програми, а надійність - під впливом несприятливих виробничих чинників. Такі порушення, якщо не вжити належних заходів, можуть призвести до зниження працездатності, травм або нещасних випадків.

Виконуючи інтегративну функцію, ЦНС забезпечує злагоджену взаємодію всіх органів і систем організму, підтримує його стійкий внутрішній стан. Несприятливі умови праці можуть призвести до стомлення нервової системи, що послаблює її інтегративну функцію і може спровокувати розлад ряду фізіологічних систем: серцево-судинної, шлунково-кишкової, дихальної тощо або призвести до різних захворювань (інфаркти, інсульти, виразкові хвороби).

Завдяки координаційній функції ЦНС здійснює підпорядкування багатьох рефлексів одному, який має на даний час найважливіше значення для організму.

Усі функції центральної нервової системи реалізуються в кожній конкретній реакції організму, забезпечуючи ефект найбільшого пристосування до мінливих умов зовнішнього середовища і підвищуючи фізіологічну опірність організму шкідливим зовнішнім впливам.

Вища нервова діяльність людини заснована на функціях двох сигнальних систем. Анатомічною основою першої сигнальної системи є аналізатори (зоровий, слуховий). Аналізатор - це система нервових клітин, які сприймають і переробляють інформацію, що надходить до них із зовнішнього та внутрішнього середовища організму.

Анатомічною основою другої сигнальної системи, яка властива тільки людині, є мовно-руховий апарат, тісно пов'язаний із зоровим та слуховим аналізаторами, а її подразником є слово. Мова, в усіх її видах, являє собою найбагатше джерело подразників. За допомогою слова передаються сигнали про конкретні подразники, і в цьому випадку слово служить принциповим подразником - сигналом сигналів, є пусковим механізмом дій і вчинків людей.

Мова підвищує здатність мозку відображати дійсність, забезпечує аналіз і синтез, абстрактне мислення, створює можливість для спілкування, використання і передачі життєвого досвіду, досягнень культури і мистецтва. Але в деяких випадках слово може бути негативним подразником і може призвести до розладів нервової системи, порушень функціонування всіх систем організму і, таким чином, стати небезпечним виробничим фактором.

Центральна нервова система бере участь у прийманні, обробці та аналізі будь-якої інформації, що надходить із зовнішнього і внутрішнього середовищ. При виникненні перенавантажень на організм людини нервова система визначає ступінь їхнього впливу і формує адаптаційно-захисну реакцію.

Отже, під час любої роботи, яка потребує розумової діяльності, необхідно робити перерви для відпочинку, а також слідкувати за власним становищем нервової системи, щоб в подальшому не мати проблем з ментальним здоров’ям. Також слід відпочивати достатню кількість часу у вільний від роботи час і уникати перепрацювань на роботі.

3.2 Надзвичайні ситуації екологічного характеру

Надзвичайні ситуації екологічного характеру різних видів є доволі небезпечними для людини та негативним чином впливають як на її фізичне так і на моральне здоров’я. Сама ж надзвичайна ситуація – це процес виникнення протягом короткого періоду часу екстремальних умов для людини, подолання яких вимагає високого персонального рівня фізичної, фізіологічної, психічної та моральної адаптованості [27].

Радіаційно-небезпечними об’єктами є атомні електростанції та реактори, радіохімічні підприємства, споруди для обробки та захоронення радіоактивних відходів. У разі аварії або катастрофи з радіаційно небезпечним об'єктом утворюється осередок радіоактивного забруднення. Залежно від характеру і тривалості небезпеки радіоактивне зараження місцевості можна

розділити на два види: те, що відбувається під час вибуху ядерного боєприпасу, і те, що відбувається під час аварії на атомній електростанції. До радіаційно-небезпечних об’єктів належать атомні електростанції та реактори, радіохімічні підприємства, споруди для обробки та захоронення радіоактивних відходів.

У разі аварії або катастрофи з радіаційно небезпечним об'єктом утворюється осередок радіоактивного забруднення. Залежно від характеру і тривалості небезпеки радіоактивне забруднення місцевості можна розділити на два види: те, що відбувається під час вибуху ядерного боєприпасу, і те, що відбувається під час аварії на атомній електростанції. У ядерних вибухах переважають радіонукліди з коротким періодом напіврозпаду. Як наслідок, рівень радіації стрімко падає. Характеристиками аварій на АЕС є, по-перше, радіоактивне забруднення атмосфери та місцевості летючими радіонуклідами, по-друге, тривалий період напіврозпаду цезію та стронцію. Тому рівень радіації різко не знизився. При ядерному вибуху основну небезпеку становить зовнішнє опромінення близько 90 - 95% загальної дози а при аваріях на АЕС доза зовнішнього опромінення складає 15%, а внутрішнього – 85% [28].

При виникненні таких небезпечних ситуацій, лунає спеціальний сигнал який сповіщає про небезпеку радіоактивного зараження, почувши який, необхідно:

– прийняти протирадіаційний препарат з індивідуальної аптечки;

– надіти засоби захисту органів дихання, такі як протигази, респіратори, ватно-марлеві пов'язки дорослим і дітям;

– загерметизувати квартиру, наприклад заклеїти вікна, вентиляційні отвори та ущільнити стики;

– одягти куртки, брюки, комбінезони, плащі з прогумованої або щільної тканини;

– укрити продукти харчування в герметичній тарі;

– автобуси і інші криті машини подавати безпосередньо до під'їздів.

Оксиди сірки та азоту, що викидаються в атмосферу тепловими електростанціями та автомобільними двигунами, з'єднуючись з атмосферною вологою, утворюють невеликі крапельки сірчаної та азотної кислоти, які розносяться вітром у вигляді кислотного туману і проходять крізь краплі повітря, випадаючи на землю у вигляді кислотний дощ [29].

Взимку біля ТЕС і металургійних комбінатів іноді випадає кислотний сніг, який шкідливіший за кислотні дощі через більший вміст кислоти. Ця засніжена місцевість забруднюється по 4-5 місяців, а оскільки цей сніг тане навесні, то концентрація шкідливих речовин в талій воді деколи у десятки разів більше ніж у самому снігу. Отже, при забрудненому повітря не можна ходити під дощем із непокритою головою, це може призвести до випадіння волосся, також не можна допускати, щоб такий дощ потрапляв на шкіру, на очі. Якщо це все таки відбувається, потрібно промити уражену ділянку водою, хоча опіки кислотний дощ зазвичай не спричиняє.

Смог – аерозоль, який складається з диму, туману і пилу, один з видів забруднення повітря в атмосфері у великих містах і промислових центрах. Він викликає різні алергічні реакції, подразнення слизової оболонки, приступи бронхіальної астми, пошкодження рослинності, будівель та споруд.

Всі ці хімічні речовини, як правило, мають високу хімічну активність і легко окислюються, тому фотохімічний смог вважається однією з головних проблем сучасної цивілізації. Спекотна безвітряна погода сприяла утворенню смогу. Смог особливо небезпечний для дітей, літніх людей, а також людей із захворюваннями серця і легенів, бронхітом, астмою, емфіземою. Сам дим викликати задишку, утруднення та зупинку дихання, головний біль та кашель. Він також викликає запалення слизових оболонок очей, носа і горла, знижуючи імунітет. Ще смог часто збільшує кількість госпіталізацій, ремісії та смертності від респіраторних та серцевих захворювань.

Щоб уникнути негативних наслідків від смогу, слід:

- обмежити перебування на вулиці у ранковий час, оскільки вранці концентрація смогу у повітрі максимальна;
- вразливим групам населення варто відмовитись від тривалих прогулянок;
- утриматись від фізичного навантаження на вулиці;
- щільно зачиняти вікна та двері;
- увімкнути системи кондиціонування та очищення повітря;
- проводити вологе прибирання у житлових приміщеннях, використовувати зволожувачі повітря;
- не курити і не вживати алкоголь, тому, що це підвищує ризик розвитку серцево-судинних захворювань та хвороб дихальних шляхів;
- збільшити споживання рідини до 2-3 літрів на день;
- слід промивати ніс і горло, вживати кисломолочні продукти, підсолену й лужну воду;
- обмежити використання автомобіля, надавати перевагу громадському транспорту.

У разі, коли отруєння смогом відбулось і з’явились ознаки першіння в горлі, подразнення слизових носа, очей чи утруднене дихання, слід прополощіть горло сольовим розчином. У разі погіршення стану потрібно звернутися до лікаря.

Отже, знаючи цю інформацію, можна спасти життя людині та надати їй першу допомогу у різних випадках, та знаючи детальніше про надзвичайні ситуації є можливість забезпечити безпеку робітників від вищеописаних чинників та створити безпечні умови для їхньої праці в офісному приміщенні.

3.3. Вимоги безпеки до робочих місць для виконання робіт

При проектуванні робочого місця враховувано, що при виконанні роботи з фізичним навантаженням бажана поза "стоячи", а при малих зусиллях – "сидячи".

При проектуванні робочого місця необхідно брати до уваги: якщо при прямій позі "сидячи" роботу м'язів умовно взяти за 1.0, то при прямій позі "стоячи" м'язова робота підвищується у 1.6 раза, а при похилій позі "сидячи" — у 4 рази, при похилій позі "стоячи" – у 10 разів [30].

Робота в позі "стоячи" призводить до більшого стомлення. Бажано передбачати можливість зміни пози під час роботи. Це попереджує виникнення таких профзахворювань, як варикозне розширення вен при роботі стоячи або геморою при роботі сидячи. Крісло має відповідати вимогам ДСТУ ГОСТ 12.2.032-78 "Робоче місце при виконанні робіт сидячи. Загальні ергономічні вимоги" а робочі місця "стоячи" мають відповідати ДСТУ ГОСТ 12.2.033-78 "Робоче місце при виконанні робіт стоячи. Загальні ергономічні вимоги".

Антропометричні дані, наведені в ДСТУ ГОСТ 12.2.049-80 "Обладнання виробниче. Загальні ергономічні вимоги", допомагають правильно спроектувати робоче місце, форми і розміри обладнання органів управління з урахуванням анатомічної структури, фізіологічних можливостей і особливості людини. Ручні органи управління розташовуються на висоті 1000—1600 мм над рівнем підлоги при обслуговуванні стоячи й на висоті 600—1200 мм при обслуговуванні сидячи. Сидіння на постійному робочому місці за пультом або щитом управління повинно мати пристрій для регулювання його як по вертикалі, так і по горизонталі. Конструкція сидіння повинна мати підлокітники, опори для спини та підставку для ніг згідно з антропометричними даними. Органи управління на пульті або щиті

розташовуються на відстані не більш як 800 мм від вертикальної осі сидіння. Загальні ергономічні вимоги наведені в ДСТУ ГОСТ 12.2.049-80.

Критерії оцінки умов праці на виробництві викладені в офіційному документі, який має назву "Гігієнічна класифікація праці за показниками шкідливості і небезпечності факторів виробничого середовища, тяжкості й напруженості трудового процесу" (наказ МОЗ України від 27.12.01р. № 528). Цей документ призначений для гігієнічної оцінки умов і характеру праці, для встановлення пріоритетності при проведенні оздоровчих заходів; розробки рекомендацій щодо профвідбору і профпридатності прогнозування професійної захворюваності; створення банку даних про умови праці на рівні підприємства, району, міста, країни в цілому; атестації робочих місць за умовами праці; впровадження диференційованих внесків у Фонд соціального страхування від нещасних випадків на виробництві та професійних захворювань України; визначення економічних санкцій у зв'язку з несприятливими умовами праці.

Отже, до основних вимог до робочого місця відносяться зручність працівників під час роботи, робочим місцем є стіл, за яким працівник буде працювати за комп’ютером, відносять можливість працювати у положеннях “сидячи” та “стоячи” для меншого навантаження у першому положенні та уникненню ризиків отримання профзахворювань у другому положенні. Також важливим є гігієна на робочому місці.

3.4 Загальні вимоги безпеки з охорони праці для користувачів ПК

Освітлювальні установки повинні забезпечувати рівномірне освітлення і не повинні утворювати засліплюючих відблисків на клавіатурі, а також на екрані монітора за напрямом очей. Також воно повинне бути змішаним, тобто природним та штучним. При роботі з комп'ютером не допускається розташування робочого місця в приміщеннях без природного освітлення, без

наявності природної або штучної вентиляції. Саме ж робоче місце з комп'ютером повинно розміщуватися на відстані не менше одного метра від стіни, а від стіни з віконними отворами – на відстані не менше 1,5 м. Кут нахилу екрана монітора або ноутбука по відношенню до вертикалі повинен складати 10-15 градусів, а відстань до екрана – 500-600 міліметрів, а кут зору екрана повинен бути прямим і становити 90 градусів. У приміщенні кабінету і на робочому місці необхідно підтримувати чистоту і порядок, проводити систематичне провітрювання. Про всі виявлені під час роботи несправності обладнання необхідно доповісти керівнику, у випадку поломки необхідно припинити роботу до усунення аварійних обставин. При виявленні можливої небезпеки, попередити оточуючих та негайно повідомити керівнику [31].

Перед початком роботи, слід оглянути і переконатися у справності обладнання та електропроводки. У разі виявлення несправностей, до роботи не приступати. Потрібно повідомити про це керівника і, тільки після усунення несправностей і його дозволу, приступити до роботи. Також потрібно перевірити освітлення робочого місця, за необхідності, вжити заходів до його нормалізації, ще перевірити наявність та надійність захисного заземлення устаткування, стан електричного шнура і вилки, справність вимикачів та інших органів управління персональним комп'ютером.

Під час роботи з ПК, вмикати і вимикати його слід тільки вимикачем, забороняється проводити вимкнення витягуванням вилки з розетки. Також забороняється переміщати та переносити системний блок, монітор чи будь-яке обладнання, що знаходиться під напругою, пити будь-які напої та приймати їжу, залишати включене обладнання без нагляду чи класти предмети на комп'ютерне обладнання. Сумарний час безпосередньої роботи з персональним комп'ютером протягом робочого дня має бути не більше 6 годин. Тривалість безперервної роботи з персональним комп'ютером без регламентованої перерви не повинна перевищувати двох годин. Через кожну годину роботи слід робити перерву тривалістю 15 хв. Під час регламентованих

перерв, з метою зниження нервово-емоційного напруження, стомлення зорового аналізатора, усунення впливу гіподинамії та гіпокінезії, запобігання розвитку познотонічного стомлення, слід виконувати комплекси вправ для очей або організовувати фізкультурні паузи.

При закінченні роботи з ПК вимикаємо його від електромережі, для чого необхідно вимкнути тумблери, а потім акуратно витягнути штепсельні вилки з розетки. Протерти зовнішню поверхню комп'ютера чистою вологою тканиною, при цьому не допускати використання розчинників, одеколону, препаратів в аерозольній упаковці. Після чого прибрати робоче місце та ретельно провітрити приміщення з персональним комп'ютером.

ВИСНОВКИ

Отже, оновлення будь-якої веб-сторінки є важливим етапом у життєвому циклі веб-ресурсів, адже технології у сучасному світі не стоять на місці і необхідно вчасно вдосконалювати вигляд свого продукту в інтернеті, оскільки це є важливим як для маркетингу, так і для покращення роботоспроможності. В даному випадку веб-сторінка зазнала значних змін, як у вигляді зі сторони фронтенду, так і в роботоспроможності зі сторони бекенду.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

– Розглянуто технічне завдання, яке містить основні критерії та аспекти, яким повинна відповідати оновлена версія.

– Подано теоретичну інформацію про інструментарій для написання веб-сайту.

– Висвітлено графіки та діаграми зі структурою сайту та його елементів.

В другому розділі кваліфікаційної роботи:

– Спроектовано оновлений дизайн веб-сторінок, який відповідає сучасним вимогам до веб-дизайну.

– Оновлено програмне забезпечення, на якому працює веб-сайт, що покращує роботу веб-сайту.

– Розроблено бекенд частину, що вдосконалює модульність коду веб-сайту.

У розділі «Безпека життєдіяльності, основи охорони праці» висвітлено роль нервової системи під час трудової діяльності, основні застереження та поради щодо надзвичайних ситуацій екологічного характеру. Подано інформацію про вимоги безпеки робочих місць для виконання робіт та загальні вимоги щодо безпеки з охорони праці для користувачів ПК.

ПЕРЕЛІК ДЖЕРЕЛ

1. A Brief History of PHP [Електронний ресурс] – Режим доступу до ресурсу: https://www.oreilly.com/library/view/programming-php-3rd/9781449361068/ch01s02.html.

2. PHP Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.php.net/docs.php.

3. OpenCart Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.opencart.com/.

4. Laravel vs OpenCart: выбираем лучшую платформу для создания интернет-магазина [Електронний ресурс] – Режим доступу до ресурсу: https://it-rating.ua/news-3184.

5. Laravel 8 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/docs/8.x/readme.

6. Frain B. Responsive Web Design with HTML5 and CSS: Develop future-proof responsive websites using the latest HTML5 and CSS techniques, 3rd Edition / Ben Frain., 2020. – 410 с.

7. Історія розвитку Html [Електронний ресурс] – Режим доступу до ресурсу: https://markup-ua.com/istoriya-razvitiya-html/.

8. Our Review of Figma, the Google Docs for Designers! [Електронний ресурс] – Режим доступу до ресурсу: https://usersnap.com/blog/review-figma/.

9. Adobe XD - The Ultimate Design Tool [Електронний ресурс] – Режим доступу до ресурсу: https://webtribunal.net/graphics/adobe-xd/#gref.

10. Sketch 2021 review [Електронний ресурс] – Режим доступу до ресурсу: https://www.creativebloq.com/reviews/sketch.

11. Create a Clean and Professional Web Design in Photoshop [Електронний ресурс] – Режим доступу до ресурсу: https://www.webfx.com/blog/web-design/create-a-clean-and-professional-web-design-in-photoshop/.

12. Adobe XD vs Figma [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@sannanmalikofficial/adobe-xd-vs-figma-e62fe3af60f5.

13. Bootstrap 5 Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

14. Foundation vs. Bootstrap: Which front end framework to use? [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@davegenge/bootstrap-vs-foundation-which-front-end-framework-to-use-e85319258b88.

15. Material design introduction [Електронний ресурс] – Режим доступу до ресурсу: https://material.io/design/introduction.

16. SKELETON FRAMEWORK TUTORIALS [Електронний ресурс] – Режим доступу до ресурсу: https://ieatcss.com/skeleton-tutorial.html.

17. Ларман К. Застосування UML 2.0 і шаблонів проектування. 3-тє вид / Крег Ларман., 2019. – 736 с.

18. Скляр Д. Изучаем PHP 7. Руководство по созданию интерактивных веб-сайтов / Давид Скляр., 2017. – 464 с. – (O'Reilly Media). – (O`Reilly).

19. Зандстра М. PHP: объекты, шаблоны и методики программирования. 5-е издание / Мэтт Зандстра., 2019. – 736 с. – (Expert's Voice APress).

20. Stauffer M. Laravel: Up & Running: A Framework for Building Modern PHP Apps 2nd Edition / Matt Stauffer., 2019. – 554 с.

21. McGrath M. PHP & MySQL in easy steps, 2nd Edition / Mike McGrath., 2018. – 340 с.McGrath M. PHP & MySQL in easy steps, 2nd Edition / Mike McGrath., 2018. – 340 с.McGrath M. PHP & MySQL in easy steps, 2nd Edition / Mike McGrath., 2018. – 340 с.

22. Laravel Eloquent ORM Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://linuxhint.com/laravel_eloquent_orm_tutorial/.

23. Емброуз Г. Книга Основи. Графічний дизайн 01. Підхід і мова / Г. Емброуз, Н. Леонард., 2019. – 192 с. – (Basic Graphic Design).

24. Лаптон Е. Графічний дизайн. Нові основи / Е. Лаптон, Д. Коул., 2020. – 264 с. – (ArtHuss).

25. DuRocher D. HTML and CSS QuickStart Guide: The Simplified Beginners Guide to Developing a Strong Coding Foundation, Building Responsive Websites, and Mastering ... of Modern Web Design (QuickStart Guides) / David DuRocher., 2021. – 361 с.

26. Роль центральної нервової системи в трудовій діяльності людини [Електронний ресурс] – Режим доступу до ресурсу: https://library.if.ua/book/9/920.html.

27. Надзвичайні ситуації екологічного характеру [Електронний ресурс] – Режим доступу до ресурсу: http://www.ukrainereferat.org/uaref-3871-2.html.

28. Надзвичайні ситуації соціального, природного й техногенного походження, причини їхнього виникнення [Електронне джерело] – Режим доступу до ресурсу: http://www.lit.dp.ua/oz/2.html.

29. Як убезпечитися від негативних наслідків смогу – поради МОЗ [Електронний ресурс] – Режим доступу до ресурсу: https://www.ukrinform.ua/rubric-society/3434350-ak-ubezpecitisa-vid-negativnih-naslidkiv-smogu-poradi-moz.html.

30. Вимоги безпеки до робочого місця. Ергономічні вимоги [Електронний ресурс] – Режим доступу до ресурсу: https://pidru4niki.com/16850303/bzhd/vimogi_bezpeki_robochogo_mistsya_ergonomichni_vimogi.

31. Інструкція з охорони праці при роботі на персональному комп’ютері [Електронний ресурс] – Режим доступу до ресурсу: https://www.sop.com.ua/article/485-nstruktsya-z-ohoroni-prats-pri-robot-na-personalnomu-kompyuter.

ДОДАТКИ

Додаток А

Програмний код файлу CategoryController.php

```
<?php
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Category;
use Illuminate\Support\Facades\Storage;
class CategoryController extends Controller
{
    public function index ()
    {
        $roots = Category::roots ();
        return view ('admin.category.index', compact ('roots'));
    }
    public function create ()
    {
        $parents = Category::roots ();
        return view ('admin.category.create', compact
('parents'));
    }
    public function store (Request $request)
    {
        $this->validate ($request, [
            'parent_id' => 'integer',
            'name' => 'required|max:100',
            'slug' =>
'required|max:100|unique:categories,slug|alpha_dash',
        ]);
        $data = $request->all ();
        $dates = [
            "name" => $request->name,
```

```
            "slug" => $request->slug,
            "parent_id" => '0',
            "content" => $request->content,
        ];
        $category = Category::create ($dates);
        return redirect ()
            ->route ('admin.category.show', ['category' =>
$category->id])
            ->with ('success', 'Нова категорія додана');
    }
    public function show (Category $category)
    {
        return view ('admin.category.show', compact
('category'));
    }
    public function edit (Category $category)
    {
        $parents = Category::roots ();
        return view ('admin.category.edit', compact ('category',
'parents'));
    }
    public function update (Request $request, Category
$category)
    {
        $id = $category->id;
        $this->validate ($request, [
            'parent_id' => 'integer',
            'name' => 'required|max:100',
            'slug' => 'required|max:100|unique:categories,slug,'
. $id . ',id|alpha_dash',
        ]);
        $dates = [
            "name" => $request->name,
            "slug" => $request->slug,
            "parent_id" => '0',
```

```
            "content" => $request->content,
        ];
        $category->update ($dates);
        return redirect ()
            ->route ('admin.category.show', ['category' =>
$category->id])
            ->with ('success', 'Кетегорія успішно оновлена');
    }
    public function destroy (Category $category)
    {
        if  ($category->products->count ()) {
            $errors[] = 'Неможливо видалити категорію;
        }
        if  (!empty ($errors)) {
            return back ()->withErrors ($errors);
        }
        $category->delete ();
        return redirect ()
            ->route ('admin.category.index')
            ->with ('success', 'Категорія успішно видалена');
    }
}
```

Програмний код файлу OrderController.php

```
<?php
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use App\Models\Order;
use Illuminate\Http\Request;
class OrderController extends Controller
{
    public function index ()
    {
```

```
        $orders = Order::orderBy ('id', 'asc')->paginate (5);
        $statuses = Order::STATUSES;
        return view ('admin.order.index', compact ('orders',
'statuses'));
    }
    public function create ()
    {
        //
    }
    public function store (Request $request)
    {
        //
    }
    public function show (Order $order)
    {
        $statuses = Order::STATUSES;
        return view ('admin.order.show', compact ('order',
'statuses'));
    }
    public function edit (Order $order)
    {
        $statuses = Order::STATUSES;
        return view ('admin.order.edit', compact ('order',
'statuses'));
    }
    public function update (Request $request, Order $order)
    {
        $order->update ($request->all ());
        return redirect ()
            ->route ('admin.order.show', ['order' => $order-
>id])
            ->with ('success', 'Заказ успішно оновлено');
    }
    public function destroy (Order $order)
```

```
    {
        //
    }
}
```

Програмний код файлу ProductController.php

```
<?php
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use App\Models\Category;
use App\Models\Product;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\File;
class ProductController extends Controller
{
    public function index ()
    {
        $roots = Category::where ('parent_id', 0)->get ();
        $products = Product::paginate (5);
        return view ('admin.product.index', compact ('products',
'roots'));
    }
    public function category (Category $category)
    {
        $id = $category->id;
        $products = $category->products ()->where ('id', $id)-
>paginate (5);
        return view ('admin.product.category', compact
('category', 'products'));
    }
    public function create ()
    {
```

```
        $items = Category::all ();
        return view ('admin.product.create', compact ('items'));
    }
    public function store (Request $request)
    {
        $file = $request->file ('image');
        if  ($file) {
            $dest = public_path ('/storage/product');
            $image_name = time () . '.' . $file-
>getClientOriginalName ();
            $file->move ($dest, $image_name);
        }
        $data = $request->all ();
        $data['image'] = $image_name ?? null;
        $datas = [
            "category_id" => $data['category_id'],
            "name" => $data['name'],
            "content" => $data['content'],
            "slug" => $data['slug'],
            "image" => $data['image']
        ];
        $product = Product::create ($datas);
        return redirect ()
            ->route ('admin.product.show', ['product' =>
$product->id])
            ->with ('success', 'Новий товар успішно доданий');
    }
    public function show (Product $product)
    {
        return view ('admin.product.show', compact ('product'));
    }
    public function edit (Product $product)
    {
```

```
        $items = Category::all ();
        return view ('admin.product.edit', compact ('product',
'items'));
    }
    public function update (Request $request, Product $product)
    {
        if  ($request->remove) {
            $image_path = "storage/product/" . $product->image;
            if  (File::exists ($image_path)) {
                File::delete ($image_path);
            }
        }
        $file = $request->file ('image');
        if  ($file) {
            $dest = public_path ('/storage/product');
            $image_name = time () . '.' . $file-
>getClientOriginalName ();
            $file->move ($dest, $image_name);
        }
        $data = $request->all ();
        $data['image'] = $image_name ?? null;
        if  ($request->remove == NULL) {
            $image = $data['image'];
        } else {
            $image = NULL;
        }
        $datas = [
            "category_id" => $data['category_id'],
            "name" => $data['name'],
            "content" => $data['content'],
            "slug" => $data['slug'],
            "image" => $image
        ];
```

```
        $product->update ($datas);
        return redirect ()
            ->route ('admin.product.show', ['product' =>
$product->id])
            ->with ('success', 'Товар успішно оновлено');
    }
    public function destroy (Product $product)
    {
        $image_path = "storage/product/" . $product->image;
        if  (File::exists ($image_path)) {
            File::delete ($image_path);
        }
        $product->delete ();
        return redirect ()
            ->route ('admin.product.index')
            ->with ('success', 'Товар успішно видалений');
    }
}
```

Програмний код файлу UserController.php

```
<?php
namespace App\Http\Controllers\Admin;
use App\Http\Controllers\Controller;
use App\Models\User;
use App\Models\Order;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
class UserController extends Controller
{
    public function index ()
    {
        $orders = Order::all ();
```

```
        $users = User::paginate (5);
        return view ('admin.user.index', compact ('users',
'orders'));
    }
    public function create ()
    {
        //
    }
    public function store (Request $request)
    {
        //
    }
    public function show (User $user)
    {
        //
    }
    public function edit (User $user)
    {
        return view ('admin.user.edit', compact ('user'));
    }
    public function update (Request $request, User $user)
    {
        $this->validator ($request->all (), $user->id)->validate
();
        if  ($request->change) {
            $request->merge (['password' => Hash::make
($request->password)]);
            $user->update ($request->all ());
        } else {
            $user->update ($request->except ('password'));
        }
        return redirect ()
            ->route ('admin.user.index')
            ->with ('success', 'Дані користувача оновлено');
```

```
    }
    public function destroy (User $user)
    {
        //
    }
    private function validator (array $data, int $id)
    {
        $rules = [
            'name' => [
                'required',
                'string',
                'max:255'
            ],
            'email' => [
                'required',
                'string',
                'email',
                'max:255',
                'unique:users,email,' . $id . ',id',
            ],
        ];
        if  (isset ($data['change_password'])) {
            $rules['password'] = ['required', 'string', 'min:8',
'confirmed'];
        }
        return Validator::make ($data, $rules);
    }
}
```

Програмний код файлу 2022_02_16_213240_create_products_table.php

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
```

```
use Illuminate\Support\Facades\Schema;
class CreateProductsTable extends Migration
{
    public function up ()
    {
        Schema::create ('products', function  (Blueprint $table)
{
            $table->id ();
            $table->bigInteger ('category_id')->unsigned ()-
>nullable ();
            $table->bigInteger ('brand_id')->unsigned ()-
>nullable ();
            $table->string ('name', 100);
            $table->text ('content')->nullable ();
            $table->string ('slug', 100)->unique ();
            $table->string ('image', 50)->nullable ();
            $table->decimal ('price', 10, 2, true)->default (0);
            $table->timestamps ();
            $table->foreign ('category_id')
                ->references ('id')
                ->on ('categories')
                ->nullOnDelete ();
            $table->foreign ('brand_id')
                ->references ('id')
                ->on ('brands')
                ->nullOnDelete ();
        });
    }
    public function down ()
    {
        Schema::dropIfExists ('products');
    }
}
```

Програмний код файлу 2022_03_28_171409_create_orders_table.php

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
class CreateOrdersTable extends Migration
{
    public function up ()
    {
        Schema::create ('orders', function  (Blueprint $table) {
            $table->id ();
            $table->bigInteger ('user_id')->unsigned ()-
>nullable ();
            $table->string ('name');
            $table->string ('email');
            $table->string ('phone');
            $table->string ('address');
            $table->string ('comment')->nullable ();
            $table->decimal ('amount', 10, 2)->unsigned ();
            $table->timestamps ();
            $table->foreign ('user_id')
                ->references ('id')
                ->on ('users')
                ->nullOnDelete ();
        });
    }
    public function down ()
    {
        Schema::dropIfExists ('orders');
    }
}
```

Програмний код файлу web.php

```
<?php

use Illuminate\Support\Facades\Route;

Route::get ('/', 'App\Http\Controllers\IndexController')->name
('index');

Route::get ('/catalog/index',
'App\Http\Controllers\CatalogController@index')->name
('catalog.index');

Route::get ('/catalog/category/{slug}',
'App\Http\Controllers\CatalogController@category')->name
('catalog.category');

Route::get ('/catalog/brand/{slug}',
'App\Http\Controllers\CatalogController@brand')->name
('catalog.brand');

Route::get ('/catalog/product/{slug}',
'App\Http\Controllers\CatalogController@product')->name
('catalog.product');

Route::get ('/basket/index',
'App\Http\Controllers\BasketController@index')->name
('basket.index');

Route::get ('/basket/checkout',
'App\Http\Controllers\BasketController@checkout')->name
('basket.checkout');

Route::post ('/basket/add/{id}',
'App\Http\Controllers\BasketController@add')

    ->where ('id', '[0-9]+')

    ->name ('basket.add');

Route::post ('/basket/plus/{id}',
'App\Http\Controllers\BasketController@plus')

    ->where ('id', '[0-9]+')

    ->name ('basket.plus');

Route::post ('/basket/minus/{id}',
'App\Http\Controllers\BasketController@minus')

    ->where ('id', '[0-9]+')

    ->name ('basket.minus');

Route::post ('/basket/remove/{id}',
'App\Http\Controllers\BasketController@remove')

    ->where ('id', '[0-9]+')

    ->name ('basket.remove');
```

```
Route::post ('/basket/clear',
'App\Http\Controllers\BasketController@clear')->name
('basket.clear');

Route::prefix ('user')->group (function  () {

    Auth::routes ();

});

Route::get ('/home',
[App\Http\Controllers\UserController::class, 'index'])->name
('user.index');

Route::post ('/basket/saveorder',
'App\Http\Controllers\BasketController@saveOrder')->name
('basket.saveorder');

Route::get ('/basket/success',
'App\Http\Controllers\BasketController@success')->name
('basket.success');

Route::group ([

    'as' => 'admin.',

    'prefix' => 'admin',

    'middleware' => ['auth', 'admin']

], function  () {

    Route::get ('index',
'App\Http\Controllers\Admin\IndexController')->name ('index');

    Route::resource ('category',
'App\Http\Controllers\Admin\CategoryController');

    Route::resource ('product',
'App\Http\Controllers\Admin\ProductController');

    Route::get ('product/category/{category}',
'App\Http\Controllers\Admin\ProductController@category')

        ->name ('product.category');

    Route::resource ('order',
'App\Http\Controllers\Admin\OrderController', ['except' => [

        'create', 'store', 'destroy'

    ]]);

    Route::resource ('user',
'App\Http\Controllers\Admin\UserController', ['except' => [

        'create', 'store', 'show', 'destroy'

    ]]);
```

```
    Route::resource ('page',
'App\Http\Controllers\Admin\PageController');

    Route::get ('slider/index',
'App\Http\Controllers\Admin\SliderController@index')->name
('slider.index');

    Route::post ('slider/store',
'App\Http\Controllers\Admin\SliderController@store')->name
('slider.store');

    Route::post ('slider/destroy',
'App\Http\Controllers\Admin\SliderController@destroy')->name
('slider.destroy');

    Route::get ('gallery/index',
'App\Http\Controllers\Admin\GalleryController@index')->name
('gallery.index');

    Route::post ('gallery/store',
'App\Http\Controllers\Admin\GalleryController@store')->name
('gallery.store');

    Route::post ('gallery/destroy',
'App\Http\Controllers\Admin\GalleryController@destroy')->name
('gallery.destroy');

});

Route::get ('/page',
'App\Http\Controllers\PageController@index')->name
('page.index');

Route::get ('/about-us',
'App\Http\Controllers\PageController@about_us')->name
('page.about_us');

Route::get ('/payment-delivery',
'App\Http\Controllers\PageController@payment_delivery')->name
('page.payment_delivery');

Route::get ('/gallery',
'App\Http\Controllers\PageController@gallery')->name
('page.gallery');

Route::get ('/contacts',
'App\Http\Controllers\PageController@contacts')->name
('page.contacts');

Route::get ('/about-us',
'App\Http\Controllers\PageController@about_us')->name
('page.about_us');

Route::get ('/page/{slug}',
'App\Http\Controllers\PageController@show')->name ('page.show');
```

Додаток Б

Програмний код файлу home.blade.php

```
@extends ('layouts.app')
@section ('content')
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-8">
            <div class="card">
                <div class="card-header">{{ __ ('Dashboard')
}}</div>
                <div class="card-body">
                    @if  (session ('status'))
                        <div class="alert alert-success"
role="alert">
                            {{ session ('status') }}
                        </div>
                    @endif
                    {{ __ ('You are logged in!') }}
                </div>
            </div>
        </div>
    </div>
</div>
@endsection
```

Програмний код файлу app.blade.php

```
<!doctype html>
<html lang="{{ str_replace ('_', '-', app ()->getLocale ()) }}">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
```

```
    <!-- CSRF Token -->
    <meta name="csrf-token" content="{{ csrf_token () }}">
    <title>{{ config ('app.name', 'Laravel') }}</title>
    <!-- Scripts -->
    <script src="{{ asset ('js/app.js') }}" defer></script>
    <!-- Fonts -->
    <link rel="dns-prefetch" href="//fonts.gstatic.com">
    <link href="https://fonts.googleapis.com/css?family=Nunito"
rel="stylesheet">
    <!-- Styles -->
    <link href="{{ asset ('css/app.css') }}" rel="stylesheet">
</head>
<body>
    <div id="app">
        <nav class="navbar navbar-expand-md navbar-light bg-
white shadow-sm">
            <div class="container">
                <a class="navbar-brand" href="{{ url ('/') }}">
                    {{ config ('app.name', 'Laravel') }}
                </a>
                <button class="navbar-toggler" type="button"
data-bs-toggle="collapse" data-bs-
target="#navbarSupportedContent" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-
label="{{ __ ('Toggle navigation') }}">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="collapse navbar-collapse"
id="navbarSupportedContent">
                    <!-- Left Side Of Navbar -->
                    <ul class="navbar-nav me-auto">
                    </ul>
                    <!-- Right Side Of Navbar -->
                    <ul class="navbar-nav ms-auto">
                        <!-- Authentication Links -->
```

```
                        @guest
                            @if  (Route::has ('login'))
                                <li class="nav-item">
                                    <a class="nav-link" href="{{
route ('login') }}">{{ __ ('Login') }}</a>
                                </li>
                            @endif

                            @if  (Route::has ('register'))
                                <li class="nav-item">
                                    <a class="nav-link" href="{{
route ('register') }}">{{ __ ('Register') }}</a>
                                </li>
                            @endif
                        @else
                            <li class="nav-item dropdown">
                                <a id="navbarDropdown"
class="nav-link dropdown-toggle" href="#" role="button" data-bs-
toggle="dropdown" aria-haspopup="true" aria-expanded="false" v-
pre>
                                    {{ Auth::user ()->name }}
                                </a>
                                <div class="dropdown-menu
dropdown-menu-end" aria-labelledby="navbarDropdown">
                                    <a class="dropdown-item"
href="{{ route ('logout') }}"

onclick="event.preventDefault ();

document.getElementById ('logout-form').submit ();">
                                        {{ __ ('Logout') }}
                                    </a>
                                    <form id="logout-form"
action="{{ route ('logout') }}" method="POST" class="d-none">
                                        @csrf
                                    </form>
```

```
                                </div>
                            </li>
                        @endguest
                    </ul>
                </div>
            </div>
        </nav>
        <main class="py-4">
            @yield ('content')
        </main>
    </div>
</body>
</html>
```

Програмний код файлу show.blade.php

```
@extends ('layout.site', ['title' => $page->name])
@section ('content')
<div class="card">
    <div class="card-header">
        <h1>{{ $page->name }}</h1>
    </div>
    <div class="card-body">
        {!! $page->content !!}
    </div>
    <div class="card-footer">
        Добавлена: {{ $page->created_at->format ('d.m.Y H:i') }}
    </div>
</div>
@endsection
```

Програмний код файлу index.blade.php

```
@extends ('layout.site', ['title' => 'Головна'])
```

```
@section ('content')

<div class="row">

  <div id="carouselExampleDark" class="carousel carousel-light
slide" data-bs-ride="carousel">

    <div class="carousel-indicators">

      <button type="button" data-bs-
target="#carouselExampleDark" data-bs-slide-to="0"
class="active" aria-current="true" aria-label="Slide
1"></button>

      <button type="button" data-bs-
target="#carouselExampleDark" data-bs-slide-to="1" aria-
label="Slide 2"></button>

      <button type="button" data-bs-
target="#carouselExampleDark" data-bs-slide-to="2" aria-
label="Slide 3"></button>

    </div>

    <div class="carousel-inner">

      @foreach ($images as $image)

      <div class="carousel-item active" data-bs-
interval="10000">

        <img src="\storage\slider\{{ $image->name }}" class="d-
block w-100 h-100" alt="">

      </div>

      @endforeach

    </div>

    <button class="carousel-control-prev" type="button" data-bs-
target="#carouselExampleDark" data-bs-slide="prev">

      <span class="carousel-control-prev-icon" aria-
hidden="true"></span>

      <span class="visually-hidden">Previous</span>

    </button>

    <button class="carousel-control-next" type="button" data-bs-
target="#carouselExampleDark" data-bs-slide="next">

      <span class="carousel-control-next-icon" aria-
hidden="true"></span>

      <span class="visually-hidden">Next</span>

    </button>

  </div>
```

```
</div>
@endsection
```

Програмний код файлу contacts.blade.php

```
@extends ('layout.site', ['title' => 'Головна'])
@section ('content')
<div class="row">
</div>
@endsection
```