БАКАЛАВРСЬКА РОБОТА

БР. ІП - 15.00.00.000 ПЗ

Група ІП-21-4

Павлів Владислав

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Павлів Владислав Олегович

(прізвище, ім'я, по батькові)

УДК 004

(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка шаблонів генерації дизайн-зображень на одязі

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

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

Здобувач освітнього рівня Павлів В.О.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Процюк Галина Ярославівна, асистент

(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Павліву Владиславу Олеговичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) **“ Розробка шаблонів генерації дизайн-зображень на одязі”**

керівник проекту (роботи) Процюк Г.Я., асистент

затверджені наказом закладу вищої освіти від **“** 28 ” квітня 2025 р. № 264/7

2. Строк подання студентом проекту (роботи) 10 червня 2025 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз використання ІТ для індивідуалізації одягу шляхом генерації дизайн-зображень
2. Аналіз веб-платформ і програмних систем дизайну та кастомізації
3. Алгоритмічна реалізація системи генерації шаблонів дизайн-зображень на одязі
4. Проектування діаграми класів
5. Програмна реалізація основних функцій системи генерації шаблонів дизайн-зображень

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Приклад веб-ресурсу на основі інструмент PrintCommerce від Magento (рис. 1.1)
2. Інтерфейс системи на основі Zakeke (рис. 1.2)
3. Інтерфейс Lumise Product Designer (рис. 1.3)
4. Вигляд плагіну Fancy Product Designer (рис. 1.4)
5. Системна архітектура програмного рішення (рис. 2.4)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **28 квітня 2025 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз використання IT для індивідуалізації одягу шляхом генерації дизайн-зображень	08.05.2025	виконано
2	Аналіз веб-платформ і програмних систем дизайну та кастомізації	18.05.2025	виконано
3	Алгоритмічна реалізація системи генерації шаблонів дизайн-зображень на одязі	25.05.2025	виконано
4	Проектування діаграми класів	01.06.2025	виконано
5	Програмна реалізація основних функцій системи генерації шаблонів дизайн-зображень	07.06.2025	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	10.06.2025	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Бакалаврська робота містить 79 сторінок, 25 рисунків, список використаних джерел із 36 найменуваннями, 1 додаток.

Мета роботи: створення програмної системи генерації шаблонів дизайн-зображень на одязі з використанням сучасних технологій веб-розробки.

Об'єкт дослідження: процес кастомізації текстильних виробів із використанням інформаційних технологій

Предмет дослідження: алгоритми, засоби та програмні компоненти веб-системи для генерації дизайн-зображень на одязі

Результати дослідження: розроблена система може бути інтегрована у виробництво одягу, онлайн-магазини, друкарні та сервіси кастомізації текстильних виробів.

В першому розділі проаналізовано програмні рішення для індивідуалізації дизайну одягу та визначено потребу в новій інтерактивній системі.

В другому розділі описано вибір технологій, бібліотек і архітектури для реалізації системи генерації шаблонів.

В третьому розділі реалізовано програмну систему генерації шаблонів дизайн-зображень з інтерактивним інтерфейсом користувача.

Висновок: розроблено архітектуру та реалізовано програмну систему, що дозволяє користувачам створювати шаблони дизайну одягу з використанням завантажених зображень і текстових елементів

КЛЮЧОВІ СЛОВА: ІНДИВІДУАЛІЗАЦІЯ ОДЯГУ; ГЕНЕРАЦІЯ ДИЗАЙНУ; ВЕБ-ЗАСТОСУНОК; LARAVEL; FABRIC.JS; WEB-TO-PRINT; ШАБЛОНИ ЗОБРАЖЕНЬ; BOOTSTRAP; PHP; MYSQL

ANNOTATION

The Bachelor's thesis consists of 79 pages, 25 figures, a list of 36 references, and 1 appendix.

Purpose of the work: To create a software system for generating design image templates for clothing using modern web development technologies.

Object of research: The process of customizing textile products using information technologies.

Subject of research: Algorithms, tools, and software components of a web system for generating design images on clothing.

Research results: The developed system can be integrated into clothing manufacturing, online stores, printing houses, and textile product customization services.

The first chapter analyzed software solutions for individualizing clothing design and identified the need for a new interactive system.

The second chapter described the selection of technologies, libraries, and architecture for implementing the template generation system.

The third chapter implemented the software system for generating design image templates with an interactive user interface.

Conclusion: An architecture has been developed and a software system has been implemented that allows users to create clothing design templates using uploaded images and text elements.

KEYWORDS: CLOTHING PERSONALIZATION; DESIGN GENERATION; WEB APPLICATION; LARAVEL; FABRIC.JS; WEB-TO-PRINT; DESIGN TEMPLATES; BOOTSTRAP; PHP; MYSQL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AJAX - Asynchronous JavaScript And XML

CSS - Cascading Style Sheets

DBMS - Database management system; an environment for managing databases

HTML - Hyper Text Markup Language

MySQL - My Structured Query Language

PHP - Hypertext Preprocessor

RAM - Random-access memory

RWD - Responsive Web Design

ВСТУП

У сучасних умовах розвитку індустрії моди спостерігається активне впровадження інформаційних технологій для індивідуалізації одягу, що обумовлено зростанням попиту на унікальні та персоналізовані вироби. Зокрема, інтерактивні веб-платформи, які дозволяють користувачам самостійно створювати дизайн-зображення для текстильних виробів, відкривають нові можливості для електронної комерції, виробництва за індивідуальними замовленнями та сталого споживання.

Незважаючи на наявність комерційних рішень у сфері web-to-print, більшість із них є закритими або надмірно складними для адаптації під локальні або нішеві потреби. Це створює потребу у розробці гнучких, масштабованих та технологічно відкритих рішень, що дозволяють реалізовувати генерацію шаблонів дизайну одягу з урахуванням користувацьких уподобань. Тому розробка власної системи генерації дизайн-зображень на одязі з використанням сучасних веб-технологій є актуальним і перспективним напрямом на перетині IT та фешн-індустрії.

Актуальність роботи

Актуальність зумовлена зростаючим попитом на індивідуалізовану продукцію в індустрії моди, що потребує автоматизованих інструментів кастомізації одягу за допомогою веб-застосунків.

У сучасних умовах цифрової трансформації суспільства сфера легкої промисловості, зокрема виробництво одягу, зазнає суттєвих змін. Одним із ключових векторів розвитку є індивідуалізація продукції — орієнтація на потреби конкретного споживача, що проявляється у персоналізованому дизайні, можливості вибору матеріалів, кольорів, стилістики тощо. У цьому контексті дедалі більшої популярності набуває підхід «mass customization», який поєднує принципи масового виробництва з індивідуальним дизайном.

Важливим інструментом реалізації зазначених тенденцій є інформаційні технології, що дозволяють автоматизувати процес взаємодії між замовником та виробником. Зокрема, широкого розповсюдження набули web-to-print платформи, які забезпечують можливість створення та замовлення індивідуальних текстильних виробів із використанням інтерактивного веб-інтерфейсу. Такі системи дають змогу кінцевому користувачу без спеціальних знань у сфері дизайну створити власний макет за допомогою простих інструментів: додавання зображень, тексту, вибору кольорів, тощо.

Проте існуючі комерційні рішення, такі як Zakeke, Lumise Product Designer або Fancy Product Designer, хоча й пропонують широкий функціонал, є переважно закритими, складними у модифікації та часто недоступними для інтеграції у локальні або спеціалізовані системи. Це створює потребу у створенні власних, адаптованих до конкретних потреб, систем, які б поєднували зручність користування, гнучкість налаштувань і відкритість архітектури.

У цьому контексті актуальним є розроблення веб-застосунку, що дозволяє реалізувати індивідуалізацію дизайну одягу шляхом генерації шаблонів зображень безпосередньо у браузері користувача. Такий підхід дає змогу не лише скоротити витрати на комунікацію між клієнтом і дизайнером, а й підвищити залученість користувача до процесу створення продукту.

Об’єкт дослідження - процес кастомізації текстильних виробів із використанням інформаційних технологій.

Предмет дослідження - алгоритми, засоби та програмні компоненти веб-системи для генерації дизайн-зображень на одязі.

Мета цієї роботи полягає у створенні програмної системи генерації шаблонів дизайн-зображень на одязі з використанням сучасних технологій веб-розробки.

Для досягнення цієї мети в роботі виконано наступні завдання:

- здійснено аналіз існуючих рішень у сфері web-to-print та веб-дизайну;

- обґрунтовано вибір інструментів і бібліотек, зокрема фреймворку Laravel, СУБД MySQL, Fabric.js і Bootstrap;

- спроєктовано архітектуру системи та реалізовано її ключові компоненти;

- розроблено інтерактивний інтерфейс для кінцевого користувача.

Методи дослідження

- Аналіз існуючих програмних рішень;

- Проєктування архітектури програмної системи;

- Моделювання бази даних;

- Програмування веб-функціоналу;

- Методи UX/UI дизайну інтерфейсів.

Наукова новизна

У роботі запропоновано інтеграційне рішення для генерації дизайн-зображень із використанням бібліотеки Fabric.js у середовищі Laravel, що дозволяє створювати індивідуальні шаблони одягу з можливістю збереження та обробки.

Практичне застосування

Розроблена система може бути інтегрована у виробництво одягу, онлайн-магазини, друкарні та сервіси кастомізації текстильних виробів.

Отримані результати мають практичне значення та можуть бути використані у створенні комерційних і навчальних веб-платформ для кастомізації одягу, а також як основа для подальших досліджень у галузі цифрового дизайну та персоналізованого виробництва.

Бакалаврська робота містить 79 сторінок, 25 рисунків, 3 розділи список використаних джерел із 36 найменуванням, 1 додаток.

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ІНДИВІДУАЛІЗАЦІЇ ОДЯГУ ШЛЯХОМ ГЕНЕРАЦІЇ ДИЗАЙН-ЗОБРАЖЕНЬ

1.1. Веб-застосунок для індивідуалізації та замовлення текстильних виробів

Даний підрозділ описує архітектуру та функціональні можливості розроблюваного веб- застосунку, призначеного для індивідуалізації дизайнів футболок та їх подальшого електронного комерційного замовлення. Основною метою проекту є створення інтерактивної веб-платформи, що надає користувачам розширений інструментарій для розробки унікальних дизайнів та зручний механізм їх придбання.

Ключовий функціонал системи охоплює комплекс можливостей для створення дизайн-макетів. Користувачі мають можливість:

- Вибирати базовий колір виробу.

- Інтегрувати текстові елементи з широкими опціями налаштування (вибір шрифту, кольору тексту).

- Завантажувати власні графічні зображення.

- Використовувати функціонал генерації дизайн-зображень для створення нових візуальних елементів на основі заданих параметрів або запитів.

- Застосовувати цифрові фільтри до завантажених або згенерованих зображень.

- Здійснювати трансформації (масштабування, обертання, переміщення) текстових та графічних об'єктів у межах дизайн-макету.

- Визначати необхідний розмір футболки.

- Зберігати створені дизайн-макети для подальшого використання або редагування.

Модуль електронної комерції дозволяє користувачам додавати фіналізовані дизайни до віртуального кошика, здійснювати управління його вмістом та проводити процедуру оформлення замовлення. Система інтегрована з платіжним шлюзом для забезпечення безпечної обробки транзакцій за допомогою кредитних карток. Користувачам також надається можливість перегляду історії своїх попередніх замовлень.

Технічна реалізація проекту базується на парадигмі об'єктно-орієнтованого програмування (ООП). Використано сучасний PHP-фреймворк Laravel у поєднанні з ключовими фронтенд-технологіями: JavaScript, CSS та HTML.

Для прискорення розробки клієнтської частини та забезпечення адаптивного інтерфейсу застосовано бібліотеки Bootstrap та jQuery. Архітектура вебзастосунку відповідає патерну Model-View-Controller (MVC), що сприяє логічному розподілу коду, полегшує його масштабування та підтримку.

Для автоматизованої комунікації, зокрема надсилання підтверджень замовлень адміністратору та клієнту, використовується сервіс Google SMTP. З метою оптимізації швидкодії та забезпечення динамічності взаємодії між клієнтською та серверною частинами системи впроваджено технологію AJAX (Asynchronous JavaScript and XML), що дозволяє асинхронно обмінюватися частинами даних без необхідності повного перезавантаження сторінки. Це робить веб-застосунок швидким та ефективним інструментом реального часу.

Таким чином, розроблювана платформа являє собою комплексне вебрішення, що інтегрує функціонал потужного дизайн-редактора, включаючи можливість генерації зображень, з повноцінним модулем електронної комерції, реалізоване з використанням сучасних веб-технологій та архітектурних підходів для забезпечення стабільності, безпеки та високої продуктивності.

1.2. Опис проекту розробки системи генерації дизайн-зображень на одязі

Програмна система генерації дизайн-зображень на одязі являє собою спеціалізований веб-застосунок, що функціонує як платформа для індивідуалізації та електронного комерційного придбання текстильних виробів (футболок). Система реалізує два основні функціональні модулі для кінцевих користувачів.

По-перше, надається інтегроване середовище для створення, модифікації та збереження користувацьких дизайн-макетів.

По-друге, забезпечується спрощений та ефективний процес оформлення замовлень на футболки з розробленим дизайном через систему віртуального кошика та онлайн-оплати.

Технічна основа застосунку побудована на використанні PHP-фреймворку Laravel, який був обраний завдяки своїй архітектурній виразності, простоті розробки та високій читабельності коду. Laravel надає низку вбудованих компонентів, що сприяють ефективній розробці, включаючи інтерфейс командного рядка Artisan, об'єктно-реляційне відображення (ORM) Eloquent для взаємодії з базою даних, механізми міграції баз даних, інструменти для побудови схем (schema builder) та стандартизований, безпечний механізм автентифікації користувачів.

1.2.1. Цілі та завдання проекту

Аналіз існуючих рішень на ринку онлайн-дизайну та продажу текстильних виробів, таких як плагіни для платформи Magento, наприклад, "Онлайн-інструмент для дизайну футболок “PrintCommerce" з орієнтовною вартістю впровадження близько 1990 доларів США, вказує на значний економічний бар'єр для входу на ринок, зокрема для представників малого

бізнесу. Висока вартість ліцензування та інтеграції таких систем обмежує їх доступність.

Інструмент PrintCommerce функціонує як розширення або плагін для платформи електронної комерції Magento. PrintCommerce, у випадку його реалізації як програмного розширення для платформи Magento, являє собою спеціалізований модуль, що інтегрує функціонал "web-to-print" (від вебу до друку) та інструменти персоналізації продукції безпосередньо в структуру онлайн-магазину, побудованого на базі Magento.

Основне призначення даного інструменту полягає у наданні суб'єктам господарювання, які використовують Magento для своєї діяльності в сфері електронної комерції (особливо тих, що займаються друком на вимогу, виробництвом сувенірної продукції або індивідуалізованих товарів), можливості запропонувати своїм клієнтам функціональність онлайн-дизайну та кастомізації товарів.

Типовий інструмент PrintCommerce для Magento включає в себе інтерфейс інтерактивного онлайн-редактора, що дозволяє кінцевим користувачам:

- Вибирати базові продукти (наприклад, футболки, чашки, візитівки).
- Завантажувати власні графічні зображення.
- Додавати та форматувати текстові елементи (вибір шрифту, кольору, розміру, вирівнювання, трансформації).
- Використовувати готові шаблони дизайнів, кліпарти або графічні елементи з бібліотеки.
- Переглядати попередній вигляд (прев'ю) створюваного дизайну на обраному продукті.

Більш просунуті версії можуть також включати функції застосування фільтрів до зображень, інструменти для роботи з шарами дизайну, можливості вибору різних варіантів продукту (розмір, колір основи) в межах інтерфейсу дизайнера.

Рисунок 1.1 – Приклад веб-ресурсу на основі інструмент PrintCommerce від Magento

Як розширення Magento, PrintCommerce інтегрується в бекенд та фронтенд платформи. Він взаємодіє з каталогом продуктів Magento, системою управління замовленнями та платіжними шлюзами. Готові дизайн-макети, створені користувачами, зберігаються та асоціюються з відповідними позиціями замовленнями, що спрощує процеси підготовки до виробництва та виконання замовлень для адміністратора магазину.

Зазвичай, PrintCommerce для Magento є комерційним програмним продуктом, що потребує придбання ліцензії для використання. Вартість таких рішень може варіюватися залежно від постачальника, набору функцій та моделі ліцензування, і, як було зазначено раніше, може становити значну інвестицію для бізнесу.

Таким чином, PrintCommerce у контексті платформи Magento є прикладом комплексного програмного рішення, що розширює стандартні можливості електронної комерції шляхом інтеграції інструментів онлайн-

дизайну, дозволяючи бізнесу ефективно функціонувати в сегменті персоналізованої продукції.

Основною метою даного проекту є розробка та впровадження економічно доцільного альтернативного рішення, що потребує мінімальних початкових інвестицій. Розроблювана програмна система покликана надати власникам малого бізнесу готову платформу, яка дозволить їхнім клієнтам самостійно створювати дизайни футболок відповідно до індивідуальних вподобань. Додатковим завданням проекту, що вирізняє його серед багатьох аналогів, є інтеграція функціоналу застосування цифрових фільтрів до графічних елементів дизайну. Реалізовано набір фільтрів (таких як Чорно-білий, Інверсія, Сепія, Ембос, Різкість), які користувач може обирати та застосовувати для модифікації завантажених або генерованих дизайн-зображень, розширюючи таким чином креативні можливості.

1.2.2. Обсяг та межі проекту

Обсяг даного проекту визначається розробкою вебзастосунку, який забезпечує повний цикл взаємодії між кінцевим користувачем та бізнесом, що пропонує послуги персоналізації та продажу футболок. Ключові функціональні можливості, включені в обсяг проекту, охоплюють:

- Вибір базового кольору виробу.
- Додавання та повне налаштування текстових елементів (колір, розмір шрифту, трансформація, включаючи обертання).
- Завантаження користувацьких графічних зображень.
- Функціонал генерації дизайн-зображень для створення унікальних графічних елементів безпосередньо в інтерфейсі редактора.
- Застосування попередньо визначених цифрових фільтрів до графічних елементів.
- Збереження створених дизайн-макетів.

- Додавання дизайнів до віртуального кошика та управління його вмістом.

- Процедура оформлення та оплати замовлення за допомогою інтегрованої системи обробки платежів кредитними картками.

- Перегляд історії попередніх замовлень користувача.

Цей застосунок розробляється для широкого кола користувачів, незалежно від вікової групи, які мають потребу в створенні персоналізованих футболок. Доступ до функціоналу редагування та замовлення передбачає безкоштовну реєстрацію. Система реалізована як адаптивний вебзастосунок (RWD - Responsive Web Design), що забезпечує його доступність та коректне відображення на різноманітних пристроях, включаючи мобільні, завдяки динамічній адаптації інтерфейсу до параметрів екрана, платформи та орієнтації пристрою користувача.

Адаптивний вебдизайн являє собою парадигму веброзробки, що передбачає створення вебсайтів, які динамічно змінюють свій макет, вміст та інтерактивні елементи відповідно до характеристик кінцевого пристрою користувача. Ці характеристики включають розмір екрана (ширина, висота), роздільну здатність, орієнтацію (портретна чи альбомна) та тип пристрою (стаціонарний комп'ютер, ноутбук, планшет, смартфон).

Головною метою RWD є забезпечення оптимального користувацького досвіду (User Experience, UX) шляхом надання зручного доступу до інформації та функціоналу вебсайту незалежно від пристрою, що використовується для перегляду. Це вирішує проблему фрагментації екосистеми цифрових пристроїв, кожен з яких має свої унікальні технічні параметри.

Таким чином, проект забезпечує високий рівень доступності та зручності використання на різних кінцевих пристроях, що підтримують веб-браузери.

1.3. Аналіз веб-платформ і програмних систем дизайну та кастомізації (Web-to-Print Solutions)

Веб-орієнтовані платформи дизайну - це програмні системи, які надають онлайн-інтерфейс для клієнтів або внутрішніх потреб компанії для створення дизайнів безпосередньо на моделі продукту. Часто інтегруються з платформами електронної комерції.

Zakeke - відомий багатофункціональний 2D/3D конфігуратор продукції та дизайнер, що інтегрується з багатьма e-commerce платформами (Shopify, Magento, WooCommerce тощо). Дозволяє клієнтам візуалізувати та створювати дизайни на об'ємних моделях одягу.

Lumise Product Designer - популярний плагін-дизайнер продукції, що часто використовується для платформ на базі WordPress/WooCommerce.

Fancy Product Designer - ще один широко відомий плагін для дизайну продуктів, що підтримує різні e-commerce системи.

Inkybay - комплексне W2P рішення для онлайн-кастомізації.

Вбудовані інструменти PoD сервісів - дизайнерські інтерфейси, що надаються такими компаніями, як Printful (наприклад, Design Maker) або Printify, для створення дизайнів безпосередньо перед замовленням друку через їхні платформи.

Програмне забезпечення для підготовки до друку (RIP Software) - ці програми не є інструментами створення дизайну, але є критично важливими для перетворення створеного дизайну у формат, зрозумілий конкретному друкувальному обладнанню (особливо для DTG, широкоформатного друку), керування кольорами, оптимізації процесу друку.

AcroRIP - широко використовується з багатьма принтерами прямого друку на тканині (DTG), особливо початкового та середнього рівня.

PrintFactory - комплексне RIP-рішення для широкоформатного та текстильного друку.

Onyx Graphics Software - серія RIP-продуктів для професійного друку.

CalderaRIP - ще один відомий пакет RIP для цифрового друку.

Розглянемо декілька програмних систем дизайну і друку на товарах.

1.3.1. Опис Zakeke - багатофункціональний 2D/3D конфігуратор продукції та дизайнер

Zakeke є багатофункціональною комерційною програмною платформою, що функціонує за моделлю Програмне забезпечення як послуга (SaaS - Software as a Service). Вона розроблена з метою надання можливості онлайн-конфігурації та персоналізації продуктів безпосередньо на платформах електронної комерції.

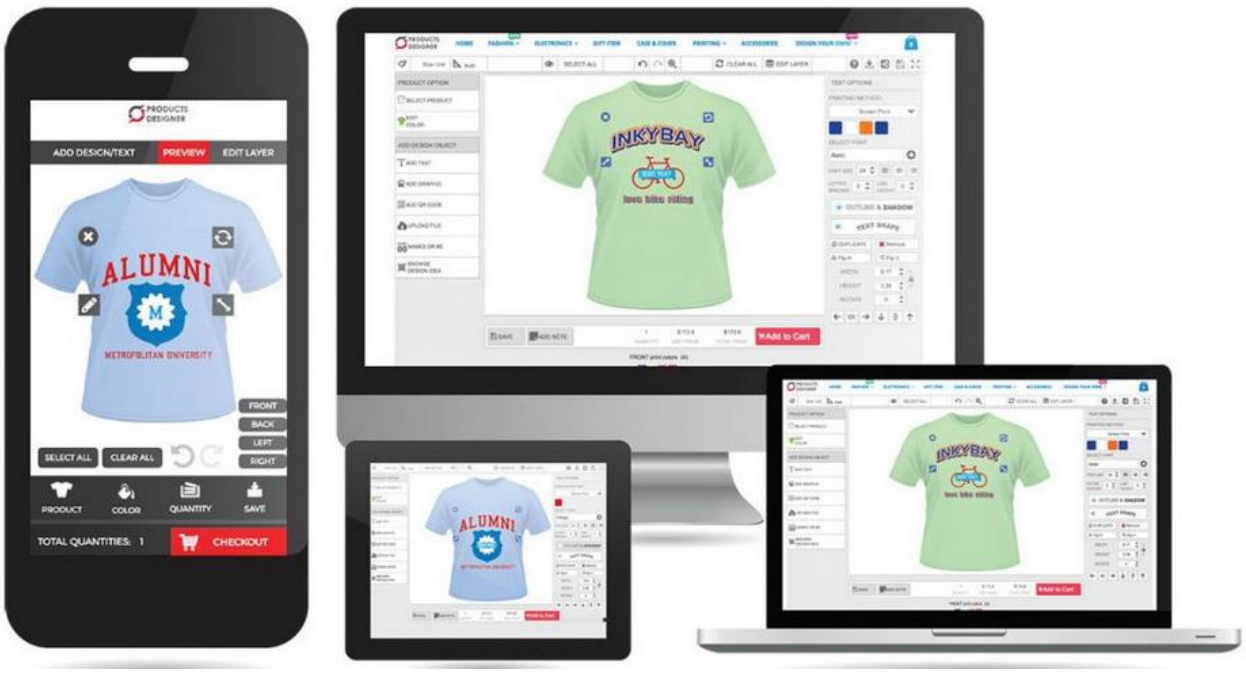


Рисунок 1.2 – Інтерфейс системи на основі Zakeke

Основне призначення Zakeke полягає в інтеграції високоякісного інтерактивного інструментарію безпосередньо у вебсайт онлайн-магазину. Це дозволяє кінцевим користувачам (покупцям) самостійно та візуально створювати унікальні дизайни для різноманітних продуктів, включаючи одяг, взуття, аксесуари, ювелірні вироби, сувенірну продукцію, друковану

продукцію тощо, перед додаванням товару до кошика та оформленням замовлення. Для бізнесу Zakeke виступає як рішення, що автоматизує процес прийому замовлень на кастомізовану продукцію та генерацію відповідних виробничих файлів.

Ключові функціональні можливості:

- Надання візуального редактора з широким набором інструментів для кастомізації. Користувачі можуть:

- Додавати текстові елементи з можливістю вибору шрифтів, кольорів, розмірів, вирівнювання та застосування різних ефектів і трансформацій.

- Завантажувати власні зображення, логотипи або фотографії.

- Використовувати бібліотеку готових графічних елементів (кліпартів), форм та шаблонів, наданих адміністратором магазину.

- Регулювати розміщення, масштабування, обертання та інші параметри всіх елементів дизайну в межах визначених областей кастомізації на продукті.

Одна з ключових переваг Zakeke – можливість переглядати процес створення дизайну та кінцевий результат як на стандартному 2D зображенні продукту, так і на інтерактивній 3D моделі. 3D візуалізація дозволяє покупцям отримати більш реалістичне уявлення про те, як дизайн буде виглядати на фізичному продукті з різних ракурсів.

Панель адміністратора дозволяє власнику магазину налаштовувати, які продукти доступні для кастомізації, визначати дозволені області дизайну, завантажувати власні шрифти та графічні бібліотеки, встановлювати правила ціноутворення залежно від обраних опцій.

Система дозволяє налаштовувати автоматичний перерахунок вартості продукту в реальному часі на основі обраних користувачем опцій кастомізації (наприклад, додавання тексту, зображень, вибір кольору або специфічних опцій).

Після оформлення замовлення Zakeke автоматично генерує готові до друку або виробництва файли у різних форматах (наприклад, PDF, PNG, JPG, SVG) з фінальним дизайном користувача. Це спрощує та автоматизує процес передачі інформації до виробничого відділу або постачальника послуг друку.

Zakeke розроблено як мультиплатформне рішення з широкими можливостями інтеграції. Існують готові модулі та API для підключення до найпопулярніших платформ електронної комерції, включаючи Shopify, Magento, WooCommerce, BigCommerce, Salesforce Commerce Cloud та інші.

Платформа орієнтована на онлайн-бізнеси різного масштабу – від малих стартапів до великих підприємств, які продають або планують продавати персоналізовану продукцію і прагнуть надати своїм клієнтам зручний та сучасний інструмент для самостійного створення дизайнів.

Таким чином, Zakeke є комплексним хмарним програмним забезпеченням, що значно розширює можливості електронної комерції у сфері кастомізації продуктів, покращуючи взаємодію з клієнтом та оптимізуючи внутрішні бізнес-процеси, пов'язані з обробкою індивідуальних замовлень.

1.3.2. Плагін-дизайнер продукції Lumise Product Designer

Lumise Product Designer є програмним розширенням, що функціонує як плагін, розроблений для інтеграції передусім з екосистемою електронної комерції, що базується на системі управління контентом WordPress та плагіні WooCommerce.

Основне призначення Lumise полягає у вбудовуванні функціоналу онлайн-дизайну та персоналізації безпосередньо у вебсайт онлайн-магазину, що використовує WooCommerce. Це дозволяє власникам такого магазину надати своїм клієнтам можливість візуально створювати та модифікувати дизайн товарів (таких як одяг, друкована продукція, сувеніри, аксесуари тощо) у режимі реального часу через веббраузер, безпосередньо на сторінці

продукту. Система сприяє підвищенню залученості клієнтів та спрощенню процесу замовлення індивідуалізованих товарів.

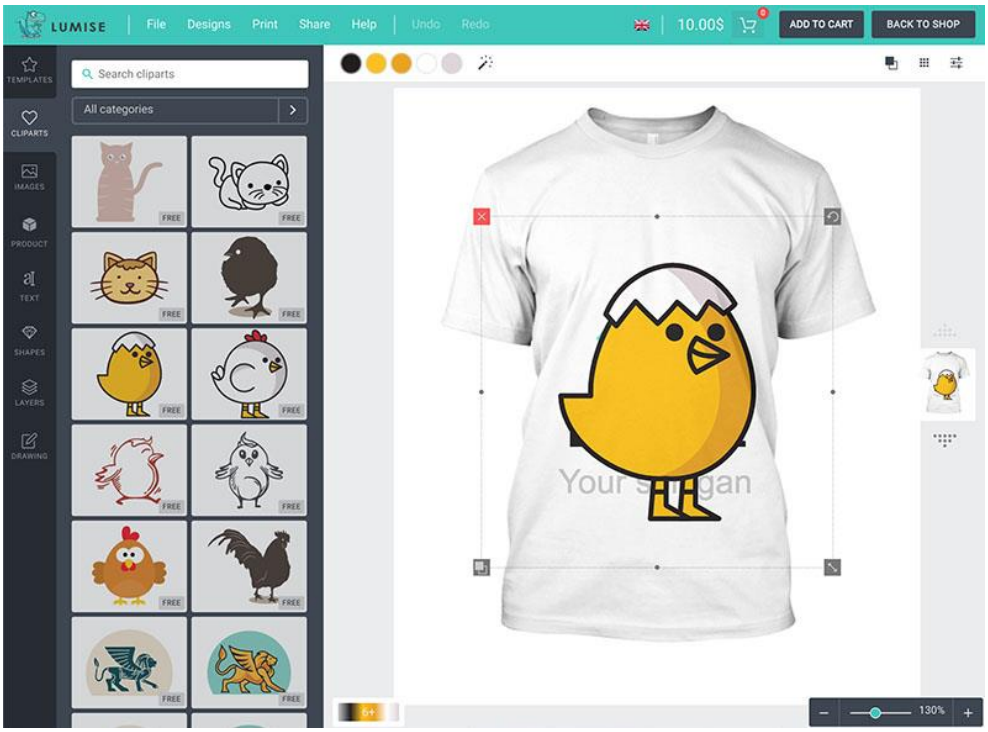


Рисунок 1.3 – Інтерфейс Lumise Product Designer

Розглянемо основні функції:

- Надання користувачам інтерактивного інтерфейсу для створення дизайнів.

- Додавання та розширене форматування текстових елементів (вибір шрифтів, кольорів, розмірів, вирівнювання, застосування ефектів, трансформації).

- Завантаження користувацьких зображень та логотипів.

- Використання багатої бібліотеки попередньо завантажених графічних елементів (кліпартів), форм та іконок.

- Застосування готових шаблонів дизайну як основи для створення власного макету.

- Інструменти для роботи з шарами (переміщення, копіювання, видалення, зміна порядку).

- Можливості трансформації елементів дизайну (переміщення, масштабування, обертання, віддзеркалення, зміна прозорості).

- Застосування різноманітних фільтрів та ефектів до зображень.

- Використання масок для створення фігурних зображень.

- Додавання QR-кодів.

Адміністративна панель Lumise дозволяє власнику магазину налаштовувати конфігурацію продуктів для кастомізації. Це включає: визначення конкретних областей на продукті, де дозволено розміщувати дизайн (наприклад, передня/задня сторона футболки), встановлення розмірів цих областей, завантаження зображень продукту з різних ракурсів, керування доступними шрифтами, графікою та шаблонами.

Плагін тісно інтегрується з функціоналом WooCommerce, включаючи підтримку варіацій продуктів (наприклад, клієнт може обрати колір або розмір футболки, і ціна та вигляд продукту в редакторі будуть відповідно оновлені) та систему замовлень.

Надається функціонал для встановлення правил, за якими ціна продукту автоматично коригується в залежності від обраних клієнтом опцій кастомізації (наприклад, додавання додаткових елементів, вибір специфічних опцій).

Після завершення процесу дизайну та оформлення замовлення система Lumise автоматично генерує високоякісні файли у форматах, придатних для друку або виробництва (таких як PDF, PNG, JPG, SVG), що містять кінцевий дизайн, створений користувачем.

Lumise Product Designer орієнтований на власників онлайн-магазинів, що функціонують на базі WordPress та WooCommerce, які прагнуть інтегрувати функціонал персоналізації товарів та надати своїм клієнтам сучасний та зручний інструмент для самостійного створення дизайнів.

Таким чином, Lumise Product Designer є спеціалізованим програмним рішенням у вигляді плагіна, що розширює стандартні можливості платформи

WooCommerce, додаючи потужний функціонал онлайн-дизайну продукції, що сприяє оптимізації продажів персоналізованих товарів.

1.3.3. Плагін для дизайну продуктів Fancy Product Designer

Функціонально, система Fancy Product Designer розгортає на сторінці продукту візуальний інтерфейс, що слугує середовищем для творчої діяльності користувача. Після активації даного інтерфейсу користувач отримує доступ до представлення обраного продукту (наприклад, футболки) та набору інструментів, розташованих, як правило, на бічних панелях або в верхній частині вікна.

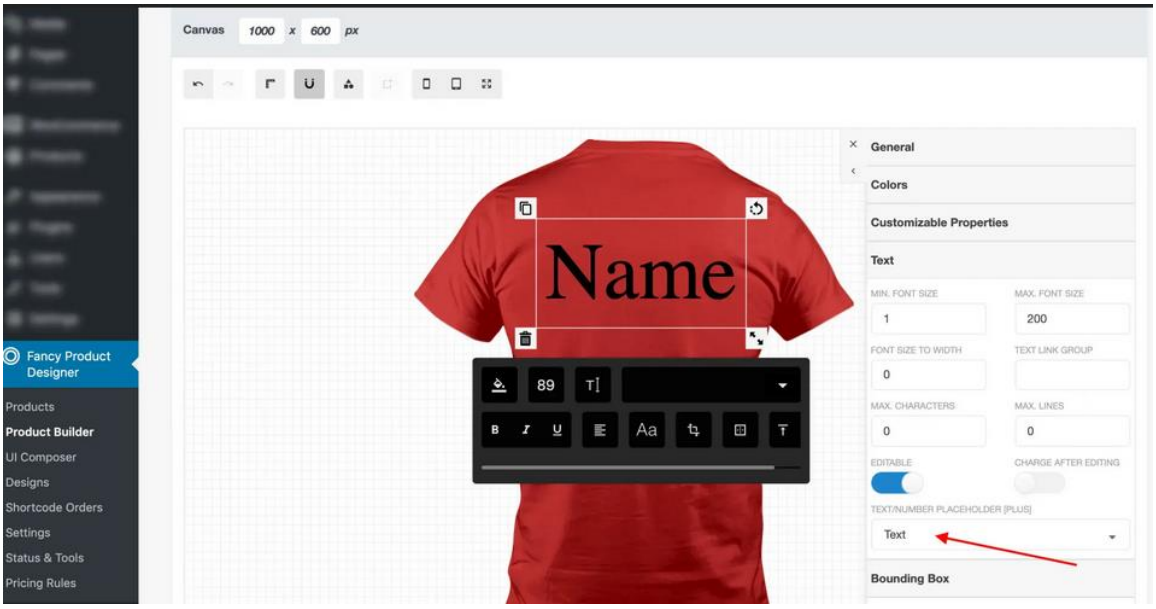


Рисунок 1.4 – Вигляд плагіну Fancy Product Designer

Процес дизайну являє собою послідовність операцій з маніпулювання об'єктами на віртуальному полотні продукту. Користувач може ініціювати додавання елементів різного типу. Модуль додавання тексту дозволяє ввести довільний текстовий рядок, після чого доступна його параметризація: вибір шрифту зі списку дозволених (що визначається адміністратором системи), налаштування кольору, розміру, вирівнювання, а також застосування геометричних трансформацій, таких як масштабування та обертання.

Інший ключовий модуль надає функціональність завантаження зображень. Користувач може імпортувати растрові або векторні графічні файли зі свого локального пристрою. Ці зображення автоматично інтегруються в робочу область дизайнера та можуть бути модифіковані за допомогою інструментів трансформації, аналогічних текстовим об'єктам. Додатково, система може надавати доступ до попередньо завантаженої адміністратором бібліотеки стандартних графічних елементів (кліпартів) або шаблонів, які користувач може використовувати як основу або доповнення до свого дизайну.

Суттєвим аспектом функціонування Fancy Product Designer є система шарів. Подібно до професійних графічних редакторів, кожен доданий елемент (текст, зображення, кліпарт) розташовується на окремому віртуальному шарі. Це дозволяє користувачеві керувати порядком відображення елементів (які об'єкти знаходяться зверху інших), а також здійснювати цільові операції над конкретним елементом без впливу на інші. До елементів можуть застосовуватися додаткові властивості, такі як зміна прозорості.

Під час всього процесу створення дизайну, Fancy Product Designer забезпечує візуалізацію кінцевого результату в реальному часі. Користувач бачить, як саме елементи розташовуються на продукті, що є критично важливим для прийняття рішень та коригування макету. Система також може динамічно оновлювати вартість продукту, враховуючи складність дизайну або додані елементи, згідно з правилами, попередньо встановленими адміністратором магазину.

На стороні адміністратора, Fancy Product Designer надає інтерфейс для конфігурації: визначення, які продукти підлягають кастомізації, на яких сторонах або в яких областях продукту можливе розміщення дизайну, завантаження доступних шрифтів та графічних ресурсів, встановлення

цінових правил, керування замовленнями та доступ до фіналізованих дизайн-макетів.

Після завершення користувачем процесу дизайну та підтвердження замовлення, система виконує функцію генерації вихідних файлів. Ці файли (зазвичай у форматах, таких як PDF, PNG, JPG, SVG) містять остаточний дизайн користувача у форматі, придатному для безпосередньої передачі на виробництво або до друкарського обладнання. Дані про створений дизайн інтегруються із відповідним замовленням у системі WooCommerce.

Таким чином, Fancy Product Designer функціонує як інтегрована програмна система, що реалізує цикл онлайн-персоналізації продукції шляхом надання користувачеві інструментарію для візуального дизайну, забезпечення зворотного зв'язку через візуалізацію та автоматизації процесу генерації виробничих даних, тим самим оптимізуючи взаємодію між споживачем та постачальником послуг кастомізації в середовищі електронної комерції.

РОЗДІЛ 2. АЛГОРИТМІЧНА РЕАЛІЗАЦІЯ СИСТЕМИ ГЕНЕРАЦІЇ ШАБЛОНІВ ДИЗАЙН-ЗОБРАЖЕНЬ НА ОДЯЗІ

2.1. Вибір інструментів та середовища розробки

2.1.1. Мова JavaScript

JavaScript (JS) — це легка інтерпретована або JIT-компільована мова програмування з функціями першого класу. JavaScript — це прототипна, багатопарадигмальна, динамічна мова, яка підтримує об'єктно-орієнтоване, імперативне та декларативне (наприклад, функціональне програмування) стилі. JavaScript — це мова програмування, яка дозволяє створювати динамічно оновлюваний контент, керувати мультимедіа, анімувати зображення та багато іншого.

З точки зору парадигм програмування, JavaScript є багатопарадигмальною мовою. Вона підтримує імперативне програмування, об'єктно-орієнтоване програмування (реалізоване через механізм прототипів, що відрізняється від класичної моделі наслідування), функціональне програмування (завдяки підтримці функцій як об'єктів першого класу, замикань) та, що особливо важливо для веброзробки, подієво-орієнтовану парадигму. Модель виконання JavaScript, особливо у браузері та Node.js, значною мірою базується на асинхронності та механізмі циклу подій (Event Loop), що дозволяє ефективно обробляти операції вводу/виводу та взаємодію з мережею, не блокуючи основний потік виконання програми. Це критично для створення швидких та відгукливих інтерфейсів та серверних застосунків.

У сучасній архітектурі вебзастосунків JavaScript відіграє центральну роль у створенні складних, динамічних інтерфейсів користувача та односторінкових застосунків (SPA). Розробка таких систем часто здійснюється з використанням розвинених фреймворків та бібліотек (наприклад, React, Angular, Vue.js), які надають архітектурні патерни та

готові компоненти для ефективної розробки. Виконання JavaScript коду у браузерах забезпечується високооптимізованими рушіями (V8, SpiderMonkey, JavaScriptCore), що використовують технології компіляції "на льоту" (Just-In-Time, JIT) для досягнення високої продуктивності.

Навколо JavaScript сформувалася надзвичайно масштабна та динамічна екосистема програмних бібліотек, фреймворків, інструментів розробки та систем керування пакетами (таких як npm, yarn), що суттєво прискорює процес розробки та дозволяє вирішувати широкий спектр завдань – від створення простих вебскриптів до складних корпоративних систем та мобільних застосунків.

2.1.2 Фреймворк Laravel

Laravel — це фреймворк вебзастосунків з виразною та елегантною синтаксичною структурою. Laravel має простий і швидкий маршрутизатор, потужні контейнери для ін'єкції залежностей, кілька бекендів для зберігання сесій і кешу, ORM бази даних, міграції схем, надійну обробку фонових завдань та реальну трансляцію подій [3].

Фундаментом архітектури Laravel є реалізація патерну Model-View-Controller (MVC). Цей архітектурний патерн передбачає логічне розділення застосунку на три взаємопов'язані компоненти, кожен з яких має чітко визначену відповідальність:

- Модель (Model) - відповідає за представлення даних, бізнес-логіку та взаємодію з джерелом даних (зазвичай, базою даних). У Laravel ця функціональність значною мірою реалізується через систему Eloquent ORM (Object-Relational Mapping).

- Представлення (View) - відповідає за логіку відображення даних та формування інтерфейсу користувача. У Laravel для цього використовується шаблонізатор Blade.

- Контролер (Controller) -діє як посередник, обробляючи вхідні запити від користувача, взаємодіючи з Моделями для отримання або модифікації даних та обираючи відповідне Представлення для формування відповіді.

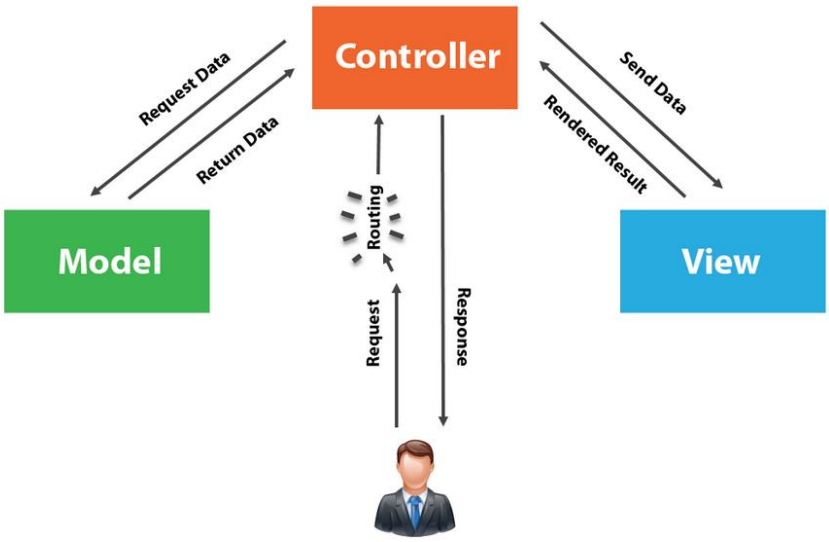


Рисунок 2.1 – Патерн MVC

Дотримання патерну MVC сприяє високій модульності програмного коду, полегшує його структурування, підвищує читабельність та значно спрощує процеси тестування, модифікації та масштабування застосунку.

Laravel інтегрує в собі низку потужних компонентів та сервісів, які покривають більшість типових завдань, що виникають при розробці вебзастосунків:

- Eloquent ORM - уадає об'єктно-орієнтований підхід до взаємодії з базами даних, що дозволяє працювати з таблицями як з класами та записами як з об'єктами, використовуючи виразний синтаксис PHP, замість написання сирого SQL коду.

- Artisan Console - інтерфейс командного рядка, що є невід'ємною частиною розробки на Laravel. Artisan автоматизує виконання численних рутинних завдань, таких як генерація boilerplate-коду (контролерів, моделей, міграцій), управління міграціями бази даних, запуск команд черг, виконання тестування та інші операції, що значно прискорює розробку.

- Blade Templating Engine - легкий та інтуїтивно зрозумілий шаблонізатор, що спрощує створення динамічних HTML-шаблонів. Blade дозволяє використовувати чистий PHP-код в шаблонах, а також надає власний набір директив для реалізації умовних операторів, циклів, виведення даних тощо.

- Система маршрутизації (Routing) - гнучкий механізм для визначення відповідності між вхідними HTTP-запитами (URL-адресами) та відповідними діями в застосунку (методами контролерів або анонімними функціями).

- Вбудовані механізми автентифікації та авторизації - надає готові, легко конфігуровані рішення для керування процесами реєстрації користувачів, входу в систему, виходу та контролю доступу до певних ресурсів на основі ролей чи дозволів.

- Міграції бази даних (Database Migrations) - система для управління версіями схеми бази даних. Дозволяє описувати зміни структури бази даних за допомогою PHP-коду, що робить процес зміни схеми контрольованим та полегшує розгортання на різних середовищах та співпрацю в команді.

- Eloquent Seeder - інструмент для заповнення бази даних тестовими або початковими даними.

2.1.3. Система управління БД MySQL

MySQL, найпопулярніша система керування реляційними базами даних з відкритим кодом, розроблена, розповсюджувана та підтримувана корпорацією Oracle. MySQL — це швидка, надійна, масштабована система керування реляційними базами даних, яка працює в клієнт-серверних або вбудованих системах з великою кількістю внесеного програмного забезпечення.

Архітектурно MySQL реалізована за класичною моделлю клієнт-сервер. Серверна частина (демон mysqld) відповідає за зберігання даних, обробку запитів, управління користувачами та забезпечення цілісності даних.

Клієнтські застосунки (включаючи консольні утиліти, графічні інтерфейси або конектори для різних мов програмування) встановлюють з'єднання із сервером для надсилання SQL-запитів та отримання відповідей.

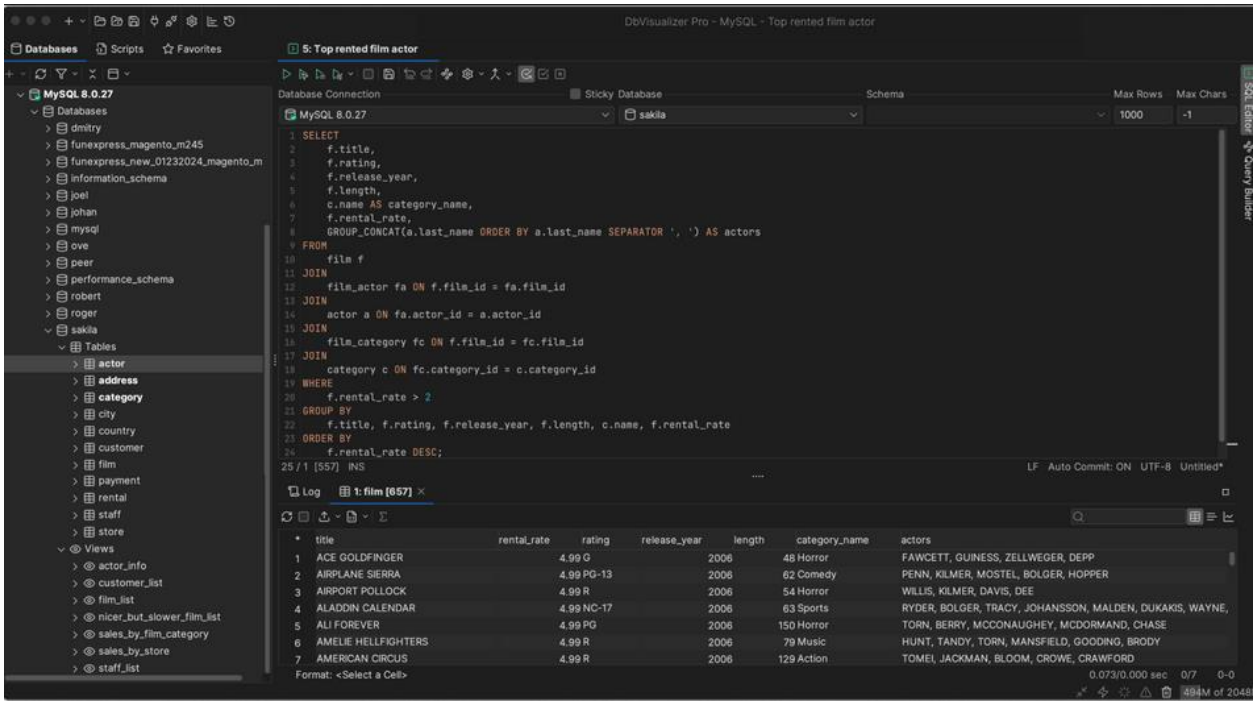


Рисунок 2.2 – Візуалізація MySQL

Ключовою архітектурною особливістю MySQL, що вирізняє її серед багатьох інших СУБД, є підтримка множинних рушіїв зберігання (Storage Engines). Це компонент, який безпосередньо відповідає за те, як дані зберігаються на диску, як реалізуються індекси, як виконуються блокування та чи підтримуються транзакції. Різні рушії мають різні характеристики продуктивності, надійності та функціональності, дозволяючи оптимізувати роботу бази даних під специфічні потреби застосунку. Наприклад, InnoDB є стандартним рушієм за замовчуванням у сучасних версіях MySQL, який повністю підтримує транзакції з властивостями ACID (Атомарність, Узгодженість, Ізольованість, Довговічність), що є критично важливим для застосунків, де необхідна висока цілісність даних при одночасних операціях.

Рушій MyISAM, хоча і не підтримує транзакції ACID на рівні рядків, може бути ефективним для сценаріїв з переважно операціями читання.

MySQL розроблена з акцентом на високу продуктивність, особливо при виконанні запитів на вибірку даних. Це досягається, зокрема, за рахунок підтримки різноманітних стратегій індексації, що прискорюють пошук даних. Надійність системи забезпечується механізмами журналювання та відновлення після збоїв. Масштабованість реалізується як вертикально (через використання потужнішого обладнання), так і горизонтально, зокрема за допомогою механізмів реплікації, що дозволяють створювати копії бази даних на інших серверах для розподілу навантаження читання та забезпечення високої доступності.

Завдяки своїй відкритості, високій продуктивності та надійності, MySQL здобула надзвичайне поширення, ставши де-факто стандартом для багатьох вебзастосунків. Вона є центральним компонентом класичних стеків розробки LAMP (Linux, Apache, MySQL, PHP/Python/Perl) та LEMP (Linux, Nginx, MySQL, PHP/Python/Perl), що використовуються для розгортання більшості динамічних вебсайтів та сервісів.

2.2. Вибір бібліотек для розробки

2.2.1. Fabric.js

Fabric.js — це бібліотека JavaScript HTML5 Canvas, яка забезпечує парсинг SVG-to-canvas та vice-versa. Створення об'єктів, складних форм, зображень стає набагато простішим з fabric.js. Вона забезпечує масштабування, переміщення, обертання, забарвлення для тексту та зображень.

Фундаментальною концепцією Fabric.js є об'єктна модель. Бібліотека представляє базові графічні примітиви (такі як лінії, кола, прямокутники), складніші елементи (текстові блоки, зображення) та їхні композиції (групи

об'єктів) як повноцінні об'єкти з власним набором властивостей (позиція, розмір, колір заливки, колір обведення, кут обертання, прозорість тощо) та методів для їх програмної модифікації.

Ця об'єктна модель уможливлює реалізацію багатого інтерактивного функціоналу. Fabric.js автоматично обробляє події введення (такі як кліки миші або дотики на сенсорних екранах) та визначає, який об'єкт на полотні був взаємодіяний. Бібліотека надає вбудовані механізми для візуального управління об'єктами: виділення об'єкта за допомогою кліка, його переміщення, масштабування (як пропорційне, так і непропорційне), обертання та нахил за допомогою спеціальних інтерактивних маркерів (handles), що відображаються навколо виділеного елемента.

Крім базових форм, Fabric.js має спеціалізовані об'єкти для роботи з текстом, що підтримують налаштування шрифтів, стилів, вирівнювання та обробку багаторядкового тексту. Об'єкти зображень дозволяють легко завантажувати растрові зображення на полотно та застосовувати до них різноманітні цифрові фільтри.

Важливим функціоналом є можливість серіалізації та десеріалізації стану полотна. Весь набір об'єктів, що знаходиться на полотні, разом з усіма їхніми властивостями, може бути збережений у форматі JSON. Це дозволяє легко зберігати поточний дизайн користувача (наприклад, у базі даних або локальному сховищі браузера) та відновлювати його пізніше.

Слід зазначити, що Fabric.js не є самостійним рушієм малювання, а виступає як надбудова над нативним Canvas API. Вона обробляє логіку об'єктної моделі, інтерактивність та керування станом, а для безпосереднього рендерингу фінальної графіки на полотні використовує низькорівневі методи нативного Canvas контексту.

Завдяки своєму функціоналу та об'єктній моделі, Fabric.js знайшла широке застосування у розробці різноманітних вебзастосунків, що вимагають інтерактивної роботи з графікою, зокрема: онлайн-редакторів фотографій,

конструкторів дизайнів для персоналізованої продукції (де користувач розміщує текст, зображення, кліпарти на моделі товару), інструментів для створення діаграм та схем, простих веб-ігор та інших застосунків, де необхідне динамічне управління графічними елементами на базі HTML5 Canvas.

2.2.2. Bootstrap

Bootstrap — це безкоштовний та відкритий фреймворк для фронтенду вебсайтів та вебзастосунків. Він містить шаблони дизайну на основі HTML та CSS для типографіки, форм, кнопок, навігації та інших інтерфейсних компонентів, а також необов'язкові розширення JavaScript.

Фундаментальним принципом розробки з використанням Bootstrap є підхід "mobile-first" (спочатку для мобільних пристроїв). Це означає, що процес проєктування та кодування починається з визначення макету та стилів для найменших екранів, а потім, за допомогою CSS-медіа-запитів, функціональність та візуальне представлення масштабуються для більших дисплеїв (планшетів, ноутбуків, стаціонарних комп'ютерів). Така методологія забезпечує пріоритетну оптимізацію для мобільних пристроїв, враховуючи обмежені ресурси та специфіку мобільного UX.

Bootstrap надає розробнику набір взаємопов'язаних компонентів та утиліт:

1. Адаптивна сіткова система (Responsive Grid System). Є основою для побудови макетів сторінок. Вона реалізована на базі CSS Flexbox (у сучасних версіях) і надає гнучку систему колонок, що дозволяє створювати складні адаптивні розташування елементів, які автоматично перебудовуються (змінюють кількість колонок, їх ширину) залежно від розміру екрана, визначеного через систему контрольних точок (breakpoints).

2. CSS Компоненти. Велика колекція попередньо стилізованих та готових до використання UI-елементів, що покривають більшість

стандартних компонентів вебінтерфейсів. До них відносяться типографія, форми, кнопки, навігаційні панелі (navbars), вкладки (tabs), модальні вікна (modals), спливаючі підказки (tooltips), картки (cards) та багато інших. Використання цих компонентів забезпечує візуальну узгодженість та прискорює розробку.

3. Утилітарні Класи (Utility Classes). Набір допоміжних CSS-класів, що дозволяють швидко застосовувати специфічні стилі (такі як внутрішні/зовнішні відступи, колір тексту/фону, відображення елементів, Flexbox/Grid властивості) без написання великого обсягу власного CSS.

4. JavaScript Компоненти. Традиційно базувалися на бібліотеці jQuery, у сучасних версіях реалізовані з використанням чистого JavaScript або мають можливість інтеграції з іншими бібліотеками/фреймворками. Надають інтерактивну поведінку для деяких UI-компонентів, таких як керування випадаючими меню, активація модальних вікон, функціонал каруселей, перемикання вкладок тощо.

Рисунок 2.3 – Приклад застосування Bootstrap

Інтеграція Bootstrap у веб-проект значно підвищує швидкість розробки, надаючи готовий "скелет" та набір UI-елементів. Розробник може швидко конструювати макет сторінки, використовуючи сіткову систему та готові CSS-класи, а потім доповнювати їх власними стилями або перевизначати

стандартні для досягнення унікального дизайну. Фреймворк бере на себе значну частину роботи з забезпечення адаптивності та кросбраузерної сумісності, що зменшує ймовірність виникнення помилок відображення на різних пристроях та у різних браузерах.

2.3. Представлення системної архітектури

Системна архітектура системи дизайну одягу побудована на основі класичної клієнт-серверної парадигми. Ця модель передбачає розподіл функціональності між двома основними компонентами: клієнтською частиною та серверною частиною.

У даній архітектурі клієнтами виступають пристрої кінцевих користувачів (персональні комп'ютери, планшети, смартфони), які оснащені веббраузерами. Веббраузер відповідає за відображення користувацького інтерфейсу (сформованого за допомогою HTML, CSS) та виконання клієнтського коду (JavaScript), що забезпечує інтерактивність, зокрема у візуальному редакторі дизайнів.

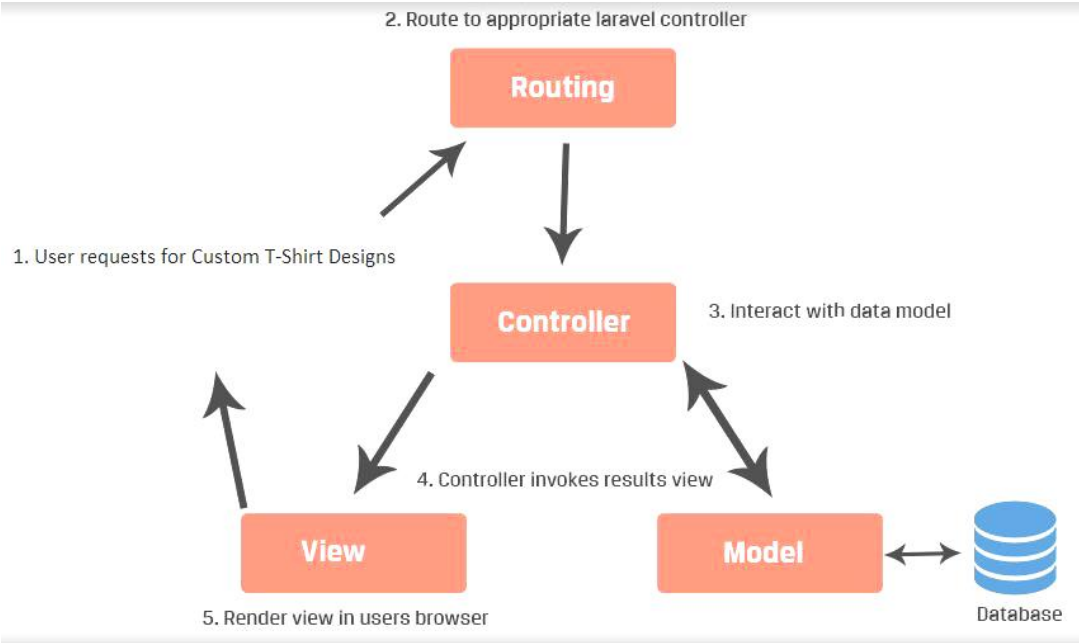


Рисунок 2.4 – Системна архітектура програмного рішення

Серверна частина розміщує основну логіку застосунку, управління даними та обробку запитів. Реалізація бекенду здійснюється з використанням потужного PHP-фреймворку Laravel. Laravel виступає як основний обробник усіх вхідних запитів від клієнтів, забезпечуючи структуроване та ефективне виконання серверної логіки.

На серверній стороні архітектура застосунку організована відповідно до архітектурного патерну Model-View-Controller (MVC), який реалізовано засобами фреймворку Laravel. Розподіл відповідальності між компонентами відбувається наступним чином:

Model - відповідає за представлення структури даних застосунку та взаємодію з постійним сховищем даних (базою даних MySQL у даному проекті). Моделі містять логіку доступу до даних, їхньої валідації та маніпуляцій.

View - відповідає за формування візуального інтерфейсу користувача, тобто того, що користувач бачить у своєму браузері (HTML, можливо з інтеграцією CSS та JavaScript). Представлення отримує дані від Контролера для відображення.

Controller - діє як координатор. Він приймає вхідні запити від клієнта (через систему маршрутизації Laravel), обробляє їх, взаємодіє з Моделями для виконання необхідних операцій з даними (наприклад, збереження нового дизайну, завантаження існуючого, обробка замовлення) і, зрештою, обирає відповідне Представлення для генерації відповіді, яку буде надіслано клієнту.

Життєвий цикл запиту в цій архітектурі відбувається наступним чином:

1. Користувач ініціює дію в клієнтському інтерфейсі (наприклад, зберігає дизайн, додає товар у кошик), що генерує HTTP-запит.

2. Цей запит надсилається на сервер.

3. Фреймворк Laravel приймає запит. Система маршрутизації Laravel аналізує URL та метод запиту та визначає, який саме Контролер має його обробити.

4. Відповідний Контролер отримує запит, видобуває з нього необхідні дані.

5. Контролер взаємодіє з однією або кількома Моделями для виконання бізнес-логіки (наприклад, збереження даних дизайну у базі даних через Eloquent ORM).

6. Після виконання логіки, Контролер визначає, яке Представлення необхідно відобразити (або які дані повернути у відповідь).

7. Якщо повертається Представлення (наприклад, HTML-сторінка), Контролер передає йому необхідні дані. Представлення формується (наприклад, за допомогою шаблонізатора Blade).

8. Сформована відповідь (HTML, JSON, тощо) повертається через фреймворк Laravel клієнту як HTTP-відповідь.

Взаємодія між клієнтською та серверною частинами для обміну даними без повного перезавантаження сторінки (що критично для динамічного редактора дизайнів) реалізована з використанням технології AJAX. В цьому випадку запити також обробляються Laravel на бекенді відповідними Контролерами, але відповідь повертається у форматі структурованих даних (наприклад, JSON), які потім обробляються JavaScript на клієнтській стороні для динамічного оновлення інтерфейсу.

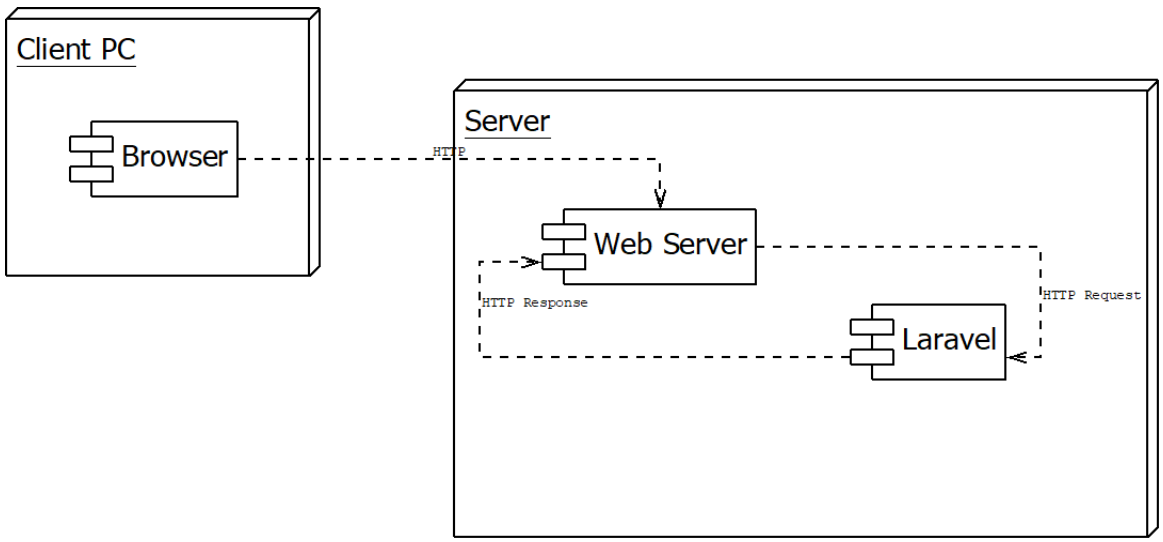


Рисунок 2.5 – Клієнт-серверна архітектура

Таким чином, системна архітектура проекту є розподіленою клієнт-серверною системою, де Laravel забезпечує надійний, структурований та безпечний бекенд, реалізуючи патерн MVC для логічного розподілу компонентів, а клієнтська сторона відповідає за інтерактивне представлення та взаємодію з користувачем, обмінюючись даними із сервером для реалізації функціоналу дизайну та електронної комерції.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ СИСТЕМИ ГЕНЕРАЦІЇ ШАБЛОНІВ ДИЗАЙН-ЗОБРАЖЕНЬ НА ОДЯЗІ

3.1. Проектування діаграми класів

Представлена діаграма класів (рисунок 3.1) ілюструє структуру даних програмної системи дизайну зображень на одязі, зокрема футболок, що відповідає модельній частині архітектури MVC.

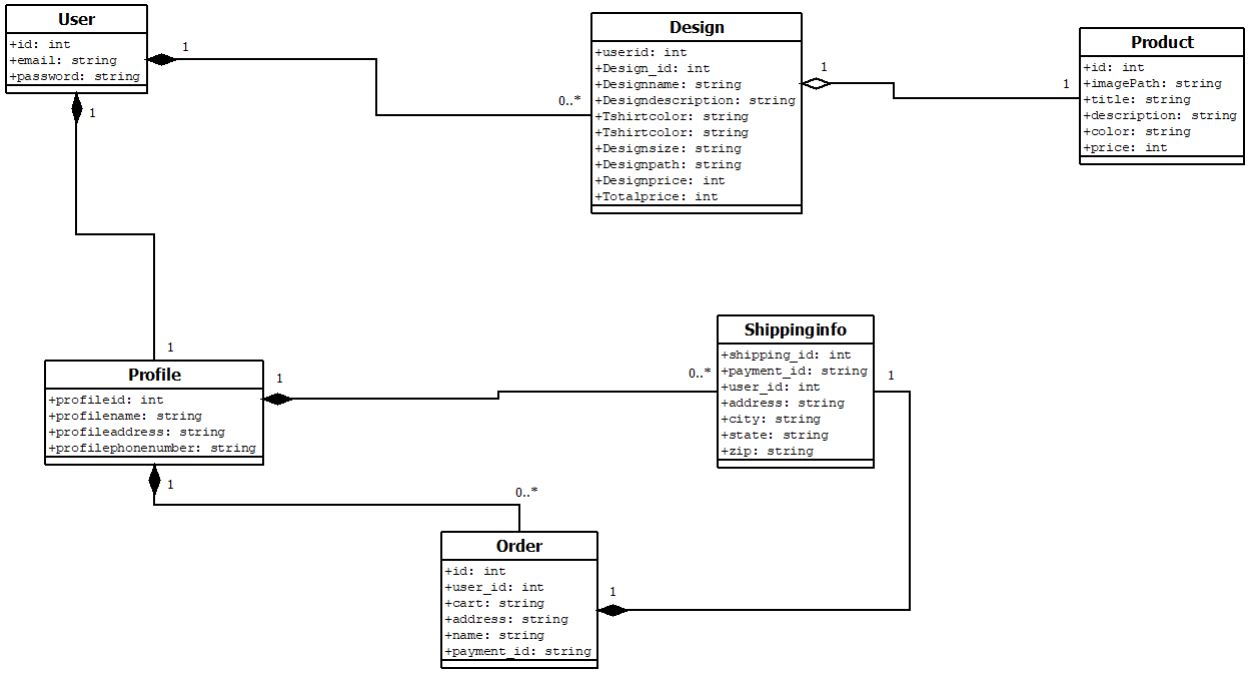


Рисунок 3.1 – Діаграма класів

Представлена на рисунку 3.1 діаграма класів ілюструє структуру даних (модель даних) програмної системи дизайну футболок, що є ключовим компонентом рівня Моделі (Model) в рамках архітектури Model-View-Controller (MVC). Діаграма відображає основні сутності системи, їхні атрибути та зв'язки між ними з вказівкою кардинальності (множинності) асоціацій.

Діаграма включає наступні ключові класи (сутності) та їх атрибути:

1. User: Представляє зареєстрованого користувача системи.

- id: int (унікальний ідентифікатор користувача)

- email: string (адреса електронної пошти користувача)

- password: string (хешований пароль користувача)

2. Profile: Містить додаткову інформацію про користувача, що доповнює базові облікові дані.

- profileid: int (унікальний ідентифікатор профілю)

- profilename: string (ім'я користувача)

- profileaddress: string (загальна адреса профілю, можливо, основна)

- profilephonenumber: string (номер телефону користувача)

3. Design: Представляє збережений користувачем дизайн для продукту.

- id: int (унікальний ідентифікатор запису дизайну)

- DesignId: int (дублюючий або альтернативний ідентифікатор дизайну)

- Designname: string (назва дизайну, надана користувачем)

- Designdescription: string (опис дизайну)

- Tshirtcolor: string (колір футболки, на якій зроблено дизайн)

- Tshirtsize: string (розмір футболки, на якій зроблено дизайн)

- Designsize: string (розміри самого дизайн-зображення або області друку)

- Designpath: string (шлях до файлів, що зберігають елементи або фінальне зображення дизайну)

- Designprice: int (вартість самого дизайну або елементів)

- Totalprice: int (загальна вартість продукту з цим дизайном)

4. Product: Представляє базовий фізичний продукт (наприклад, тип футболки), на основі якого створюється дизайн.

- id: int (унікальний ідентифікатор продукту)

- imagepath: string (шлях до зображення продукту)

- title: string (назва продукту)

- description: string (опис продукту)

- color: string (доступний колір продукту)

- price: int (базова вартість продукту)

5. ShippingInfo: Зберігає інформацію про адресу доставки.

- shipping_id: int (унікальний ідентифікатор запису про доставку)

- payment_id: string (ідентифікатор платежу; наявність тут може вказувати на зв'язок з конкретною транзакцією)

- user_id: int (ідентифікатор користувача; дублює зв'язок через Profile)

- address: string (рядок адреси)

- city: string (місто)

- state: string (область/штат)

- zip: int (поштовий індекс)

6. Order: Представляє оформлене замовлення користувача.

- id: int (унікальний ідентифікатор замовлення)

- user_id: int (ідентифікатор користувача, що здійснив замовлення)

- cart: string (серіалізоване представлення вмісту кошика, включаючи обрані дизайни та продукти)

- address: string (адреса доставки для цього замовлення; можливо, дублює або фіксує адресу з ShippingInfo на момент відповідного замовлення в системі)

- name: string (ім'я отримувача замовлення)

- payment_id: string (ідентифікатор платежу, пов'язаний з цим замовленням)

Опишемо зв'язки між сутностями (асоціації), що показані на рисунку 3.1:

- User (1) -- (1) Profile: Зв'язок один-до-одного. Кожен користувач має рівно один пов'язаний профіль, і кожен профіль належить рівно одному користувачеві.

- User (1) -- (0..*) Design: Зв'язок один-до-багатьох. Один користувач може створити та зберігати нуль або більше дизайнів. Кожен дизайн створений рівно одним користувачем.

- User (1) -- (0..*) Order: Зв'язок один-до-багатьох. Один користувач може здійснити нуль або більше замовлень. Кожне замовлення асоційоване рівно з одним користувачем.

- Profile (1) -- (0..*) ShippingInfo: Зв'язок один-до-багатьох. Один профіль користувача може мати нуль або більше збережених адрес доставки. Кожен запис ShippingInfo асоційований рівно з одним профілем.

- Product (1) -- (0..*) Design: Зв'язок один-до-багатьох. Один тип продукту може бути використаний як основа для нуля або більше дизайнів. Кожен дизайн стосується рівно одного продукту.

- Order (1) -- (0..*) ShippingInfo: Зв'язок один-до-одного з точки зору Order до ShippingInfo. Кожне замовлення обов'язково пов'язане рівно з одним записом ShippingInfo, що представляє адресу доставки для цього конкретного замовлення. Однак, запис ShippingInfo (збережена адреса) може бути використаний у нуль або більше замовлень.

Представлена діаграма класів надає високоабстрактне структурне представлення даних системи, відображаючи ключові сутності та їхні взаємовідносини. Вона слугує основою для реалізації модельного рівня в застосунку на базі Laravel, зокрема для роботи з базами даних через Eloquent ORM, де кожен клас на діаграмі, як правило, відповідає моделі Eloquent та таблиці в базі даних.

3.2. Представлення міграції бази даних у PHP-фреймворку Laravel

Цей код (лістинг 3.1) є файлом міграції бази даних у PHP-фреймворку Laravel. Міграції в Laravel – це інструмент для управління схемою бази даних вашого застосунку за допомогою PHP-коду. Вони дозволяють версіонувати

зміни структури бази даних, спрощуючи розгортання застосунку в різних середовищах та спільну роботу над проектом.

Лістинг 3.1. Файл міграції БД

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

class CreateProductsTable extends Migration
{
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->string('imagePath');
            $table->string('title');
            $table->text('description');
            $table->integer('price');
            $table->string('color');
        });
    }

    public function down()
    {
        Schema::dropIfExists('products');
    }
}
```

Представлений клас CreateProductsTable успадковує від базового класу Migration і містить два ключові методи: up() та down().

1. Метод up():

- Цей метод виконується, коли ви застосовуєте (виконуєте) цю конкретну міграцію (наприклад, командою php artisan migrate).

- Schema::create('products', function (Blueprint $table) { ... }); : Цей рядок інструктує Laravel створити нову таблицю в базі даних з назвою products. Другим аргументом є анонімна функція (Closure), яка отримує об'єкт Blueprint. Об'єкт Blueprint надає методи для визначення стовпців та індексів таблиці.

- $table->increments('id'); : Додає стовпець з назвою id. Цей стовпець буде цілочисельним (integer), автоматично збільшуватиметься при додаванні нового запису (auto-increment) і буде встановлений як первинний ключ (primary key) таблиці.

- $table->timestamps(); : Додає два стовпці: created_at та updated_at. Це мітки часу, які Laravel автоматично оновлює при створенні (created_at) та зміні (updated_at) записів у таблиці. Вони використовуються для відстеження часу життя записів.

- $table->string('imagePath'); : Додає стовпець з назвою imagePath типу VARCHAR (рядок тексту) зі стандартною довжиною (зазвичай 255 символів). Призначений для зберігання шляху до зображення продукту.

- $table->string('title'); : Додає стовпець з назвою title типу VARCHAR. Призначений для зберігання заголовка/назви продукту.

- $table->text('description'); : Додає стовпець з назвою description типу TEXT. Цей тип даних використовується для зберігання довших рядків тексту (наприклад, повного опису продукту).

- $table->integer('price'); : Додає стовпець з назвою price типу INT (ціле число). Призначений для зберігання ціни продукту.

- $table->string('color'); : Додає стовпець з назвою color типу VARCHAR. Призначений для зберігання кольору продукту.

2. Метод down():

- Цей метод виконується, коли ви відкочуєте (скасовуєте застосування) цю міграцію (наприклад, командою php artisan migrate:rollback).

- Schema::dropIfExists('products'); : Цей рядок інструктує Laravel видалити таблицю з назвою products з бази даних, якщо вона існує. Це дія, зворотна тій, що виконується в методі up().

Таким чином, весь цей код виконує створення у базі даних таблиці з назвою products зі структурою, що включає автоматичний ID, мітки часу створення/оновлення, а також стовпці для шляху до зображення, назви,

опису, ціни та кольору продукту. Ця таблиця відповідає класу Product з представленої вами діаграми класів і слугує для зберігання інформації про продукти у вашому застосунку дизайну одягу.

Усі таблиці міграції для кожного класу на діаграмі класів створюються аналогічним чином. Laravel надає таблиці міграції, тому нам не потрібно писати запити. Він надає об'єктно-орієнтований шаблон для взаємодії з базою даних. Тепер ми створюємо кожну модель наступним чином.

3.3. Визначення моделі для product у фреймворку Laravel

Код наведений в лістингу 3.2 визначає модель у фреймворку Laravel, використовуючи систему Eloquent ORM (Object-Relational Mapper).

Лістинг 3.2. Визначення моделі для product

```php
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
{
    protected $fillable = ['imagePath', 'title', 'description', 'price', 'color'];
}
```

Розглянемо цей код детальніше:

1. namespace App;: Цей рядок вказує, що клас Product належить до простору імен App. Це стандартна конвенція для основних класів застосунку в Laravel.

2. use Illuminate\Database\Eloquent\Model;: Цей рядок імпортує базовий клас Model з бібліотеки Eloquent ORM, який ми будемо розширювати.

3. class Product extends Model: Це оголошення класу Product. Ключове тут – extends Model. Це означає, що клас Product успадковує всю функціональність від базового класу Eloquent Model. Завдяки цьому, об'єкт Product тепер представлятиме записи в таблиці бази даних. За конвенцією

Laravel/Eloquent, якщо назва класу Моделі в однині та написана з великої літери (Product), Eloquent автоматично пов'яже її з таблицею бази даних, назва якої є формою множини від назви класу і написана в нижньому регістрі (products). Таким чином, цей клас Product представлятиме таблицю products, яку ми бачили у файлі міграції.

4. protected $fillable = ['imagePath', 'title', 'description', 'price', 'color'];: Це оголошення захищеної властивості $fillable. Вона містить масив назв атрибутів (стовпців таблиці). Ця властивість відіграє важливу роль у масовому присвоєнні (mass assignment). Масове присвоєння – це коли ви намагаєтеся заповнити кілька атрибутів моделі одночасно, наприклад, використовуючи масив даних, отриманих з форми ($product->fill($request->all()) або Product::create($request->all())).

- Визначення $fillable вказує Eloquent, які атрибути дозволено заповнювати таким чином.

- Це є важливою мірою безпеки для запобігання вразливостям масового присвоєння, коли зловмисник може спробувати змінити небажані поля (наприклад, змінити роль користувача або інші чутливі дані), передавши їх у масиві даних.

- У цьому випадку, $fillable дозволяє безпечно заповнювати поля imagePath, title, description, price, та color моделі Product за допомогою методів масового присвоєння.

Отде, цей код визначає Eloquent Модель Product. Ця модель слугує об'єктно-орієнтованим інтерфейсом для взаємодії з таблицею products у базі даних. Вона дозволяє вам виконувати типові операції з базою даних (читання, створення, оновлення, видалення записів, визначення зв'язків з іншими таблицями) за допомогою зручного PHP-коду, а не писати SQL-запити вручну. В контексті архітектури MVC, цей клас є реалізацією компоненту Моделі, що представляє сутність "Продукт" у застосунку. Властивість $fillable забезпечує безпеку при заповненні даних моделі.

Усі моделі Design, Order, Product, Profile, Shippinginfo, User створюються аналогічно, щоб їх можна було використовувати в контролері як об'єкт.

3.4. Визначення класу контролера ProductController у фреймворку Laravel

Представлений код (лістинг 3.3) є визначенням класу Контролера (Controller) у фреймворку Laravel, який називається ProductController. В рамках архітектури Model-View-Controller (MVC), контролер відповідає за обробку вхідних запитів від користувача, взаємодію з моделями (для роботи з даними) та вибір відповідного представлення (View) для формування відповіді.

Лістинг 3.3. Визначення класу ProductController

```php
<?php
namespace App\Http\Controllers;
use App\Cart;
use App\Product;
use Illuminate\Http\Request;
use Session;
use Stripe\Stripe;
use Stripe\Charge;
use App\Order;
use Auth;
use App\Shippinginfo;
use Mail;
use Log;
use DB;

class ProductController extends Controller
{
    public function getIndex(){
        $products = Product::all();
        return view('shop.index', ['products' => $products]);
    }

    public function getAddToCart(Request $request, $id){
        $product = Product::find($id);
        $oldCart = $request->session()->has('cart') ? $request->session()->get('cart')
        $cart = new Cart($oldCart);
        $request->session()->put('cart', $cart);
        $size = $request->cookie('size');
        $cart->add($product, $product->id, $size);
        return redirect()->route('product.index');
    }
```

```
    }

    public function getRemoveItem($id) {
        $oldCart = Session::has('cart') ? Session::get('cart') : null;
        $cart = new Cart($oldCart);
        $cart->removeItem($id);
        if (count($cart->items) > 0) {
            Session::put('cart', $cart);
        } else {
            Session::forget('cart');
        }
        return redirect()->route('product.shopping-cart');
    }

    public function getCart(){
        if(!Session::has('cart'))
            return view('shop.shopping-cart');
        $oldCart = Session::get('cart');
        $cart = new Cart($oldCart);
        return view('shop.shopping-cart', ['products' => $cart->items, 'totalPrice' =>
    }

    public function getCheckout(){
        if(!Session::has('cart')) {
            return view('shop.shopping-cart');
        }
        $oldCart = Session::get('cart');
        $cart = new Cart($oldCart);
        $total = $cart->totalPrice;
        return view('shop.checkout', ['total' => $total]);
    }

    }

    public function postCheckout(Request $request){
        if(!Session::has('cart')) {
            return redirect()->route('shop.shopping-cart');
        }
        $oldCart = Session::get('cart');
        $cart = new Cart($oldCart);
        $shippinginfo = new Shippinginfo();
        $shippinginfo->address = $request->input('shippingaddress');
        $shippinginfo->city = $request->input('city');
        $shippinginfo->state = $request->input('state');
        $shippinginfo->zip = $request->input('zipcode');
        Stripe::setApiKey('sk_test_wUvdtaSqyINKJlZbcPcJUmjo');

        try {
            $charge = Charge::create(array(
                "amount" => $cart->totalPrice * 100,
                "currency" => "usd",
                "source" => $request->input('stripeToken'),
                "description" => "Test Charge"
            ));
            $order->cart = serialize($cart);
            $order->address = $request->input('address');
            $order->name = $request->input('name');
            $order->payment_id = $charge->id;
            if(Auth::user()->orders()->save($order)) {
                $shippinginfo->payment_id = $order->payment_id;
                Auth::user()->shippinginfos()->save($shippinginfo);
```

Цей конкретний контролер ProductController відповідає за управління логікою, пов'язаною з продуктами, кошиком покупок та процесом оформлення замовлення у вашому застосунку.

Розглянемо функціональність кожного публічного методу цього контролера:

1. getIndex():

- Призначення: Обробляє запит на відображення головної сторінки магазину або списку продуктів.

- Дії: Використовує Eloquent модель Product (Product::all()) для отримання всіх записів продуктів з бази даних. Передає отриману колекцію продуктів ($products) у представлення з назвою 'shop.index' та повертає це представлення як відповідь на запит користувача.

2. getAddToCart(Request $request, $id):

- Призначення: Обробляє запит (ймовірно, GET-запит через маршрут) на додавання продукту до кошика.

- Дії: Знаходить продукт у базі даних за переданим $id (Product::find($id)). Отримує поточний стан кошика з сесії користувача ($request->session()->has('cart') ? ... : null). Створює новий об'єкт Cart, передаючи йому старий стан кошика. Отримує значення розміру товару з cookie ($request->cookie('size')). Викликає метод add() об'єкта кошика, передаючи продукт, його ID та розмір. Зберігає оновлений об'єкт кошика назад у сесію ($request->session()->put('cart', $cart)). Перенаправляє користувача на маршрут з іменем 'product.index' (ймовірно, назад на сторінку зі списком продуктів).

4. getRemoveItem($id):

- Призначення: Обробляє запит на видалення конкретного елемента з кошика за його ідентифікатором.

- Дії: Отримує поточний стан кошика із сесії (Session::has('cart') ? ... : null). Створює об'єкт Cart із цим станом. Викликає метод removeItem() об'єкта

кошика, передаючи ID елемента для видалення. Перевіряє, чи залишилися елементи у кошику. Якщо так, зберігає оновлений об'єкт кошика у сесії. Якщо ні (кошик став порожнім), видаляє запис кошика із сесії (Session::forget('cart')). Перенаправляє користувача на маршрут 'product.shopping-cart' (ймовірно, сторінка самого кошика).

5. getCart():

- Призначення: Обробляє запит на відображення вмісту кошика покупок.

- Дії: Перевіряє, чи існує кошик у сесії (!Session::has('cart')). Якщо ні, повертає представлення 'shop.shopping-cart' без даних про кошик (ймовірно, відображаючи порожній кошик). Якщо кошик існує, отримує його стан із сесії, створює об'єкт Cart і передає список елементів кошика ($cart->items) та загальну вартість ($cart->totalPrice) у представлення 'shop.shopping-cart', повертаючи його.

6. getCheckout():

- Призначення: Обробляє запит на відображення сторінки оформлення замовлення (форми введення даних для доставки та оплати).

- Дії: Перевіряє наявність кошика у сесії. Якщо його немає, перенаправляє користувача на сторінку кошика. Якщо кошик є, отримує його стан, створює об'єкт Cart, визначає загальну вартість ($total) і передає її у представлення 'shop.checkout', повертаючи його.

7. postCheckout(Request $request):

- Призначення: Обробляє дані, надіслані методом POST з форми оформлення замовлення (процес оплати та створення замовлення).

- Дії:

- Перевіряє наявність кошика у сесії. Якщо немає, перенаправляє.

- Отримує дані кошика та загальну вартість.

- Створює новий об'єкт моделі Shippinginfo та заповнює його даними про адресу доставки, отриманими із запиту ($request->input(...)).

- Встановлює секретний ключ API для платіжного шлюзу Stripe.

- Розпочинає блок обробки винятків (try...catch), що важливо для фінансових операцій.

- Використовуючи бібліотеку Stripe, створює запит на списання коштів (Charge::create). Вказує суму (загальна вартість кошика, помножена на 100, оскільки Stripe оперує центами), валюту ("usd"), токен оплати, отриманий з форми ($request->input('stripeToken')), та опис платежу.

- Потенційна помилка в оригінальному коді: Здається, об'єкт $order використовується для присвоєння ($order->cart = ..., $order->address = ... тощо) до того, як він був створений (наприклад, \$order = new Order();). Припускаючи, що створення об'єкта Order мало бути перед цим:

- Серіалізує вміст об'єкта Cart і зберігає його у полі cart об'єкта замовлення.

- Заповнює інші поля об'єкта замовлення (address, name, payment_id - ID транзакції Stripe) даними із запиту та відповіді Stripe.

- Зберігає об'єкт замовлення, пов'язуючи його з поточним автентифікованим користувачем (Auth::user()->orders()->save($order)).

- Якщо збереження замовлення пройшло успішно:

- Пов'язує об'єкт shippinginfo з ідентифікатором платежу (який тепер є ID замовлення).

- Зберігає об'єкт shippinginfo, пов'язуючи його з поточним автентифікованим користувачем (Auth::user()->shippinginfos()->save($shippinginfo)).

- Готує масив $data з інформацією для першого email-повідомлення клієнту.

- Відправляє email-повідомлення клієнту з підтвердженням замовлення, використовуючи представлення 'mail' та підготовлені дані.

- Готує другий масив $data та додаткові дані ($designcolors) для email-повідомлення адміністратору. Примітка: Отримання кольорів та шляхів до дизайнів відбувається шляхом прямих запитів до бази даних

(DB::table(...)), що може бути не найефективнішим способом, особливо у циклі; краще використовувати Eloquent зв'язки, якщо вони налаштовані між моделями Order, Cart (якщо це окрема модель) та Design.

- Відправляє друге email-повідомлення адміністратору/відділу продажів з темою "Ready for Order Design - Attachment". До цього листа прикріплюються файли дизайнів, шляхи до яких були отримані з бази даних.

- У разі виникнення винятку під час спроби обробки платежу (наприклад, відхилення транзакції) – блок catch: Перенаправляє користувача назад на маршрут 'checkout' з повідомленням про помилку, отриманим з винятку.

- Після успішної обробки платежу та збереження замовлення (або у випадку помилки в catch до очищення сесії): Видаляє інформацію про кошик із сесії (Session::forget('cart')).

- Перенаправляє користувача на маршрут 'product.index' з повідомленням про успішне придбання.

Таким чином, цей ProductController є центральним компонентом серверної логіки, який керує основними процесами електронної комерції в застосунку: відображенням товарів, управлінням вмістом кошика за допомогою сесії, обробкою платежів через Stripe, збереженням даних замовлення та доставки у базі даних через Eloquent моделі, а також відправкою підтверджень та сповіщень електронною поштою.

3.5. Визначення маршрутизації для контролерів

Запити дизайнів футболок обробляються контролерами. Ми визначаємо ці маршрути до контролерів наступним чином як показано в лістингу 3.4.

Цей код є частиною системи маршрутизації (Routing) у фреймворку Laravel. Система маршрутизації відповідає за визначення того, як застосунок реагує на вхідні HTTP-запити (GET, POST, PUT, DELETE тощо) на певних

URL-адресах. Вона зіставляє конкретну комбінацію методу запиту та URL із відповідною дією контролера (або іншим кодом), який має обробити цей запит.

Лістинг 3.4. Визначення маршрутизації для контролерів

```php
<?php
Route::get('/', [
    'uses' => 'ProductController@getIndex',
    'as' => 'product.index'
]);
Route::get('/add-to-cart/{id}', [
    'uses' => 'ProductController@getAddToCart',
    'as' => 'product.addtocart'
]);
Route::get('/reduce/{id}', [
    'uses' => 'ProductController@getReduceByOne',
    'as' => 'product.reduceByone'
]);
Route::get('/shopping-cart', [
    'uses' => 'ProductController@getCart',
    'as' => 'product.shopping-cart'
]);
Route::get('/checkout', [
    'uses' => 'ProductController@getCheckout',
    'as' => 'checkout',
    'middleware' => 'auth'
]);
Route::post('/checkout', [
    'uses' => 'ProductController@postCheckout',
    'as' => 'checkout',
    'middleware' => 'auth'
]);
Route::post('/user/savedesignvalues', [
    'uses' => 'UserController@saveDesignvalues',
    'as' => 'user.savedesignvalues'
   Route::group(['middleware' => 'auth'], function(){
       Route::get('/profile', [
           'uses' => 'UserController@getProfile',
           'as' => 'user.profile'
       ]);
       Route::get('/logout', [
           'uses' => 'UserController@getLogout',
           'as' => 'user.logout'
       ]);
       Route::get('/createdesign', [
           'uses' => 'UserController@createDesign',
           'as' => 'user.createdesign'
       ]);
       Route::post('saveprofile', [
           'uses' => 'UserController@saveProfile',
           'as' => 'user.saveprofile'
       ]);
   });
});
```

Більш детально розглянемо роботу коду на лістингу 3.4:

1. Route::get(...), Route::post(...): Це визначення маршрутів для різних HTTP-методів.

- Route::get(...): Визначає маршрут, який відповідає на GET-запити.

- Route::post(...): Визначає маршрут, який відповідає на POST-запити.

2. Перший аргумент ('/', '/add-to-cart/{id}' тощо): Це шаблон URL-адреси, на який реагує маршрут. Частина в фігурних дужках ({id}) є параметром маршруту, значення якого буде передано в метод контролера.

3. Другий аргумент (масив [...]): Це конфігурація маршруту.

'uses' => 'ControllerName@methodName': Вказує, який контролер і який його метод мають бути викликані для обробки запиту, що відповідає цьому маршруту.

'as' => 'routeName': Призначає маршруту унікальне ім'я (аліас). Ці імена зручно використовувати для генерації URL-адрес в коді Laravel (route('routeName')), що робить посилання гнучкішими (якщо URL зміняться, вам потрібно буде змінити їх лише у файлі маршрутів).

'middleware' => 'middlewareName': Вказує на проміжне програмне забезпечення (middleware), яке має бути виконане перед обробкою запиту контролером. Middleware може виконувати різні функції, наприклад, перевірку автентифікації, авторизації, логування, модифікацію запиту/відповіді тощо.

Опис конкретних маршрутів:

Route::get('/', [...]); : Маршрут для головної сторінки застосунку (кореневий URL /). Він обробляється методом getIndex контролера ProductController і має ім'я product.index.

Route::get('/add-to-cart/{id}', [...]); : Маршрут для додавання продукту в кошик. Відповідає на GET-запити до /add-to-cart/ІД_продукту. Обробляється методом getAddToCart контролера ProductController. Ім'я: product.addtocart.

Route::get('/reduce/{id}', [...]); : Маршрут, ймовірно, для зменшення кількості одиниць товару в кошику. Обробляється методом getReduceByOne контролера ProductController. Ім'я: product.reduceByone. (Примітка: метод getReduceByOne не був представлений у попередньому коді ProductController).

Route::get('/shopping-cart', [...]); : Маршрут для відображення сторінки кошика. Обробляється методом getCart контролера ProductController. Ім'я: product.shopping-cart.

Route::get('/checkout', [...]); та Route::post('/checkout', [...]); : Маршрути для сторінки оформлення замовлення.

- GET-запит відображає форму оформлення (getCheckout контролера ProductController).

- POST-запит обробляє дані з форми (платіж, створення замовлення) (postCheckout контролера ProductController).

- Обидва мають middleware 'auth', що означає, що доступ до цих маршрутів дозволений лише автентифікованим (увійшовшим у систему) користувачам.

Route::post('/user/savedesignvalues', [...]); : Маршрут для POST-запиту, що обробляється методом saveDesignvalues контролера UserController. Назва: user.savedesignvalues. Ймовірно, використовується для збереження певних параметрів процесу дизайну.

Route::post('/user/savedesign', [...]); : Маршрут для POST-запиту, що обробляється методом saveDesign контролера UserController. Назва: user.savedesign. Ймовірно, використовується для збереження фінального дизайн-макету.

Route::get('/remove/{id}', [...]); : Маршрут для видалення елемента з кошика за його ID. Обробляється методом getRemoveItem контролера ProductController. Ім'я: product.remove.

Групи маршрутів (Route::group):

Конструкція Route::group використовується для групування маршрутів, які мають спільні атрибути (наприклад, префікс URL, middleware). Атрибути, визначені для групи, автоматично застосовуються до всіх маршрутів всередині цієї групи.

Route::group(['prefix' => 'user'], function(){ ... });: Це основна група. Вона застосовує префікс /user до всіх URL-адрес маршрутів, визначених всередині. Наприклад, маршрут /signup всередині групи фактично відповідатиме URL /user/signup.

Route::group(['middleware' => 'guest'], function(){ ... });: Це вкладена група всередині групи user. Вона застосовує middleware 'guest'. Middleware 'guest' гарантує, що маршрути всередині цієї групи доступні лише для користувачів, які НЕ увійшли в систему. Сюди входять:

Route::get('/signup', [...]); та Route::post('/signup', [...]); : Маршрути для реєстрації користувача (відображення форми та її обробка). Ім'я: user.signup. Фактичний URL: /user/signup.

Route::get('/signin', [...]); : Маршрут для відображення форми входу. Ім'я: user.signin. Фактичний URL: /user/signin. (Примітка: зазвичай тут також був би POST-маршрут для обробки даних форми входу, але він не представлений у цьому фрагменті коду).

Route::group(['middleware' => 'auth'], function(){ ... });: Це інша вкладена група всередині групи user. Вона застосовує middleware 'auth'. Middleware 'auth' гарантує, що маршрути всередині цієї групи доступні лише для користувачів, які увійшли в систему. Сюди входять:

Route::get('/profile', [...]); : Маршрут для відображення профілю користувача. Ім'я: user.profile. Фактичний URL: /user/profile.

Route::get('/logout', [...]); : Маршрут для виходу користувача із системи. Ім'я: user.logout. Фактичний URL: /user/logout.

Route::get('/createdesign', [...]); : Маршрут для відображення сторінки створення дизайну. Ім'я: user.createdesign. Фактичний URL: /user/createdesign.

Route::post('saveprofile', [...]); : Маршрут для POST-запиту, що обробляє збереження даних профілю. Ім'я: user.saveprofile. Фактичний URL: /user/saveprofile.

Цей файл маршрутів визначає вхідні точки (URL-адреси та HTTP-методи), на які реагує Laravel застосунок. Він спрямовує кожен тип запиту до відповідного методу в ProductController (для товарів, кошика, оформлення замовлення) або UserController (для реєстрації, входу, профілю, збереження дизайнів/профілю). Використання груп та middleware (auth, guest) дозволяє організувати маршрути та застосувати правила доступу на рівні групи, що спрощує управління автентифікацією та авторизацією в застосунку.

3.6. Програмна реалізація основних функцій системи генерації шаблонів дизайн-зображень

3.6.1 Функція завантаження зображення

Цей код (лістинг 3.5) є фрагментом JavaScript, призначеним для виконання у веб-браузері. Його основна функція – обробка події вибору графічного файлу користувачем за допомогою елемента вводу типу file і подальше відображення цього зображення на інтерактивному полотні (Canvas), керованому бібліотекою Fabric.js, а також оновлення деяких показників вартості в інтерфейсі.

Представимо детальний опис дій коду:

1. Прив'язка обробника події:

- document.getElementById('files').onchange = function handleImage(e) { ... };

- Цей рядок знаходить HTML-елемент на вебсторінці з ідентифікатором files (це, ймовірно, <input type="file">) і прив'язує до його

події onchange анонімну функцію handleImage. Це означає, що коли користувач обирає один або кілька файлів за допомогою цього елемента вводу, функція handleImage буде автоматично викликана, отримуючи об'єкт події e.

Лістинг 3.5. Код для завантаження зображення

```
document.getElementById('files').onchange = function handleImage(e) {
    var reader = new FileReader();
    reader.onload = function (event) {
        var imgObj = new Image();
        imgObj.src = event.target.result;
        imgObj.onload = function () {
            var image = new fabric.Image(imgObj);
            image.set({
                angle: 0,
                padding: 10,
                cornersize: 10,
                left: canvasobject.width / 2,
                top: canvasobject.height / 2,
                scaleY: canvasobject.height / image.width,
                scaleX: canvasobject.width / image.width,
            });
            canvasobject.centerObject(image);
            canvasobject.add(image);
            canvasobject.renderAll();
            designpricetotal += 5;
            myDiv.innerHTML = "$" + designpricetotal;
            totalprice += 5;
            myDiv2.innerHTML = "$" + totalprice;
        };
    };
    reader.readAsDataURL(e.target.files[0]);
};
```

2. Читання файлу за допомогою FileReader:

- var reader = new FileReader(); : Створюється новий об'єкт FileReader. Це стандартний браузерний API, який дозволяє вебзастосункам асинхронно читати вміст файлів, на які посилаються об'єкти File або Blob.

- reader.onload = function (event) { ... }; : Встановлюється обробник події onload для об'єкта reader. Ця функція буде виконана лише після того, як FileReader успішно прочитає вміст файлу. Результат читання буде доступний через event.target.result.

- reader.readAsDataURL(e.target.files[0]); : Цей рядок, розташований наприкінці функції handleImage (але виконується першим у своєму блоці),

запускає процес читання вмісту першого вибраного користувачем файлу (e.target.files[0]). Метод readAsDataURL читає файл як Data URL, тобто кодує вміст файлу (наприклад, зображення) у рядок Base64, вбудований у URL. Після завершення читання спрацює подія reader.onload.

3. Завантаження зображення в стандартний HTML-об'єкт Image:

- var imgObj = new Image(); : Всередині обробника reader.onload створюється новий стандартний JavaScript/HTML-об'єкт зображення.

- imgObj.src = event.target.result; : Джерелом (src) цього нового об'єкта зображення встановлюється Data URL, отриманий після читання файлу. Браузер починає завантажувати дані зображення в цей об'єкт imgObj.

- imgObj.onload = function () { ... }; : Встановлюється обробник події onload для об'єкта imgObj. Ця функція буде виконана, коли браузер повністю завантажить та декодує дані зображення в об'єкт imgObj.

4. Створення та маніпуляції об'єктом Fabric.js Image:

- var image = new fabric.Image(imgObj); : Всередині обробника imgObj.onload створюється новий об'єкт зображення, але вже з бібліотеки Fabric.js, використовуючи завантажений HTML-об'єкт imgObj. Об'єкти Fabric.js є інтерактивними та можуть бути додані на Fabric.js Canvas.

- image.set({...}); : Встановлюються початкові властивості для цього об'єкта Fabric.js image. Встановлюється кут обертання (0), внутрішній відступ (10 пікселів), розмір маркерів для трансформацій (10 пікселів), а також координати left та top, які розраховуються так, щоб початково розмістити об'єкт по центру канваса. Встановлюються коефіцієнти масштабування по осях Y та X, розраховані на основі розмірів канваса (canvasobject.width, canvasobject.height) та вихідної ширини завантаженого зображення (image.width). (Примітка: Логіка масштабування scaleY: canvasobject.height / image.width виглядає нетиповою для збереження пропорцій зображення, вона масштабує висоту зображення на основі висоти канваса, діленої на ширину

зображення). canvasobject, очевидно, є змінною, яка посилається на об'єкт Fabric.js Canvas, який був ініціалізований раніше.

- canvasobject.centerObject(image); : Викликається метод centerObject для об'єкта канваса canvasobject. Цей метод автоматично центрує об'єкт image на канвасі. (Якщо цей рядок виконується після image.set(), то значення left та top, встановлені в set, будуть перевизначені).

- canvasobject.add(image); : Додає створений та налаштований об'єкт зображення Fabric.js на полотно canvasobject.

- canvasobject.renderAll(); : Викликає повне перемальовування всього полотна Canvas, щоб відобразити щойно доданий об'єкт та будь-які інші зміни.

5. Оновлення вартості в інтерфейсі:

- designpricetotal += 5; та totalprice += 5; : Дві змінні designpricetotal та totalprice (ймовірно, глобальні змінні, що зберігають поточну вартість дизайну та загальну вартість) збільшуються на 5 одиниць (можливо, за додавання зображення).

- myDiv.innerHTML = "$" + designpricetotal; та myDiv2.innerHTML = "$" + totalprice; : HTML-вміст елементів вебсторінки з ідентифікаторами myDiv та myDiv2 (які, ймовірно, є елементами для відображення вартості) оновлюється, показуючи нові значення цих змінних з символом "$". (myDiv та myDiv2 мають бути посиланнями на відповідні DOM-елементи, отримані раніше, наприклад, за допомогою document.getElementById).

Отже, цей код реалізує функціональність завантаження зображення користувачем для використання його у дизайні на Canvas. Він читає файл, перетворює його на об'єкт зображення Fabric.js, додає його на Canvas по центру (з певним масштабуванням), робить його інтерактивним (завдяки Fabric.js), перемальовує Canvas для відображення змін і оновлює значення вартості в інтерфейсі, при цьому, додаючи фіксовану плату за додавання зображення.

3.6.2. Функція додавання тексту

Цей код є визначенням JavaScript-функції під назвою addtext(). Ця функція призначена для виконання на стороні клієнта у веббраузері і, ймовірно, викликається, коли користувач натискає кнопку або виконує іншу дію, що ініціює додавання тексту до дизайну на полотні.

Лістинг 3.6. Код додавання тексту

```
function addtext() {
    var textvalue = document.getElementById('textbox').value;
    var text = new fabric.Text(textvalue, { left: 100, top: 100 });
    var fontcolor = document.getElementById("fontcolors").value;
    text.fontFamily = document.getElementById("fonts").value;
    text.set({ fill: fontcolor });
    canvasobject.add(text);
    if (!textvalue == "") {
        designpricetotal += 4;
        myDiv.innerHTML = "$" + designpricetotal;
        totalprice += 4;
        myDiv2.innerHTML = "$" + totalprice;
    }
    document.getElementById('textbox').value = null;
}
```

Ось детальний опис функції додавання тексту:

1. Отримання введеного тексту:

- var textvalue = document.getElementById('textbox').value;

- Цей рядок отримує доступ до HTML-елемента з ідентифікатором textbox (це, ймовірно, текстове поле вводу <input type="text"> або <textarea>) і зберігає його поточне значення (текст, введений користувачем) у змінній textvalue.

2. Створення об'єкта тексту Fabric.js:

- var text = new fabric.Text(textvalue, { left: 100, top: 100 });

- Створюється новий об'єкт тексту за допомогою бібліотеки Fabric.js (fabric.Text). Як перший аргумент передається отримане значення тексту (textvalue). Другий аргумент – це об'єкт конфігурації, що встановлює початкові властивості розташування: left: 100 та top: 100 пікселів від верхнього лівого кута полотна.

3. Отримання стилів тексту:

- var fontcolor = document.getElementById("fontcolors").value;

- Отримує значення (ймовірно, код кольору, наприклад, HEX) з HTML-елемента з ідентифікатором fontcolors (це може бути елемент <input type="color"> або <select>).

- text.fontFamily = document.getElementById("fonts").value;

- Отримує значення (назву шрифту) з HTML-елемента з ідентифікатором fonts (імовірно, <select>) і встановлює його як властивість fontFamily створеного об'єкта тексту Fabric.js.

4. Встановлення кольору тексту:

- text.set({ fill: fontcolor });

- Встановлює колір заливки (fill) для об'єкта тексту Fabric.js, використовуючи значення кольору, отримане на попередньому кроці. Метод set у Fabric.js використовується для встановлення однієї або кількох властивостей об'єкта.

5. Додавання тексту на Canvas:

- canvasobject.add(text);

- Додає створений та налаштований об'єкт тексту Fabric.js на полотно (Canvas). Змінна canvasobject, імовірно, посилається на раніше ініціалізований екземпляр fabric.Canvas. (Зазвичай після додавання об'єкта на полотно необхідно викликати canvasobject.renderAll() для оновлення візуального відображення, але цього виклику немає безпосередньо в цьому фрагменті; можливо, він викликається в іншому місці або процес оновлення відбувається автоматично в інших частинах коду).

6. Оновлення вартості (за умовою):

- if (!textvalue == "") { ... }

- Цей рядок перевіряє умову: якщо значення textvalue не є порожнім рядком, виконується код у блоці if.

- Якщо текст не порожній:

- designpricetotal += 4; : Збільшує значення змінної designpricetotal (ймовірно, вартості самого дизайну) на 4 (можливо, додаючи плату за додавання текстового елемента).

- myDiv.innerHTML = "$" + designpricetotal; : Оновлює HTML-вміст елемента з ідентифікатором myDiv, відображаючи нове значення designpricetotal з префіксом "$". (myDiv має бути посиланням на DOM-елемент).

- totalprice += 4; : Збільшує значення змінної totalprice (ймовірно, загальної вартості замовлення) на 4.

- myDiv2.innerHTML = "$" + totalprice; : Оновлює HTML-вміст елемента з ідентифікатором myDiv2, відображаючи нове значення totalprice з префіксом "$". (myDiv2 має бути посиланням на DOM-елемент).

7. Очищення поля вводу:

- document.getElementById('textbox').value = null;

- Отримує доступ до поля вводу з ID textbox і встановлює його значення в null. Це очищає вміст текстового поля після того, як текст було додано на Canvas. (Хоча встановлення значення в порожній рядок "" є більш стандартним способом очищення текстових полів у JavaScript).

Функція addtext() реалізує механізм додавання текстового елемента на інтерактивне полотно дизайнера. Вона отримує текст, колір та шрифт з відповідних полів вводу, створює об'єкт тексту Fabric.js з цими властивостями, додає його на Canvas (з початковим розміщенням), збільшує вартість дизайну та загальну вартість, якщо доданий текст не був порожнім, і, нарешті, очищає поле вводу тексту.

3.6.3. Функція збереження дизайну

Цей код (лістинг 3.7) є фрагментом JavaScript, що виконується на стороні клієнта у веббраузері і використовує бібліотеку jQuery для здійснення асинхронних HTTP-запитів (AJAX) до серверної частини

застосунку. Код реалізує логіку збереження створеного користувачем дизайну на сервері, розділену на два послідовні етапи.

Лістинг 3.7. Відправка AJAX для збереження дизайну

```
var json_data = JSON.stringify(canvasobject.toDatalessJSON());
$.ajax({
    url: 'savedesignvalues',
    type: 'POST',
    data: { _token: CSRF_TOKEN, message: JSON.stringify(designvalues), message1: svg,
    success: function (response) {
        if (response['error'] != 'error') {
            var datatest = canvasobject.toJSON();
            $.ajax({
                url: 'savedesign',
                type: 'POST',
                data: { _token: CSRF_TOKEN, message: JSON.stringify(datatest) },
                success: function (response) {
                    document.getElementById('savedata').innerHTML = "Design Saved Succ
                }
            });
        } else {
            document.getElementById('savedata').innerHTML = "Design Already Exists. Pl
        }
    }
});
```

Представимо детальний опис дій коду.

1. Підготовка даних Canvas:

var json_data = JSON.stringify(canvasobject.toDatalessJSON());

Отримує поточний стан полотна Fabric.js (canvasobject). Метод toDatalessJSON() серіалізує всі об'єкти та їхні властивості на полотні у формат JSON, але без вбудовування даних зображень. Це корисно для збереження структури дизайну без необхідності передавати великі обсяги даних зображень, якщо вони вже доступні на сервері або їх потрібно обробити інакше. Результат серіалізації перетворюється на рядок JSON за допомогою JSON.stringify().

2 Перший AJAX-запит (savedesignvalues):

$.ajax({ ... }); : Ініціює перший AJAX POST-запит до сервера.

url: 'savedesignvalues' : Вказує відносний URL кінцевої точки на сервері. З попередніх маршрутів ми знаємо, що це, ймовірно, маршрут /user/savedesignvalues, який обробляється відповідним методом контролера (UserController@saveDesignvalues).

type: 'POST' : Визначає HTTP-метод запиту як POST.

data: { ... } : Об'єкт, що містить дані, які будуть надіслані на сервер:

- _token: CSRF_TOKEN : Токен для захисту від міжсайтової підробки запитів (CSRF). CSRF_TOKEN – це глобальна змінна, що має бути визначена на стороні сервера та вбудована у HTML сторінки.

- message: JSON.stringify(designvalues) : Серіалізований рядок JSON зі значеннями змінної designvalues. Ця змінна, ймовірно, містить метадані дизайну, введені користувачем (наприклад, назва дизайну, опис, обраний тип продукту тощо).

- message1: svg : Значення змінної svg, яка, імовірно, містить SVG-представлення поточного дизайну (векторний формат).

- message2: json_data : Рядок JSON, що представляє стан Canvas без даних зображень (отриманий на кроці 1).

Обробник успіху першого запиту:

- success: function (response) { ... } : Ця функція виконується, якщо перший AJAX-запит до 'savedesignvalues' успішно завершується.

- if (response['error'] != 'error') { ... } : Перевіряє відповідь, отриману від сервера. Логіка передбачає, що сервер у разі помилки (наприклад, якщо назва дизайну вже зайнята) поверне JSON-відповідь, яка містить ключ 'error' зі значенням 'error'. Якщо відповідь не містить такого ключа/значення, це вважається успіхом першого етапу збереження.

3. Другий (вкладений) AJAX-запит (savedesign) – виконується лише у разі успіху першого етапу:

Якщо умова response['error'] != 'error' істинна, виконується наступний блок коду:

var datatest = canvasobject.toJSON(); : Знову отримує стан полотна Fabric.js, але цього разу методом toJSON(). На відміну від toDatalessJSON(), цей метод серіалізує Canvas разом з усіма даними об'єктів, включаючи дані зображень (або як Data URL, або посилання на джерело).

$.ajax({ ... }); : Ініціює другий AJAX POST-запит.

- url: 'savedesign' : Вказує URL кінцевої точки для другого запиту. З маршрутів відомо, що це, ймовірно, маршрут /user/savedesign, який обробляється методом UserController@saveDesign.

- data: { ... } : Дані, що надсилаються:

- _token: CSRF_TOKEN : Знову токен CSRF.

- message: JSON.stringify(datatest) : Серіалізований рядок JSON, що містить повний стан Canvas (включаючи дані зображень).

- Обробник успіху другого запиту:

- success: function (response) { ... } : Ця функція виконується, якщо другий AJAX-запит до 'savedesign' успішно завершується.

- document.getElementById('savedata').innerHTML = "Design Saved Successfully"; : Оновлює HTML-вміст елемента на сторінці з ідентифікатором savedata, відображаючи повідомлення про успішне збереження дизайну.

4. Обробка помилки першого етапу:

- else { ... } : Якщо умова response['error'] != 'error' хибна (тобто сервер повернув помилку, наприклад, "Дизайн вже існує"):

- document.getElementById('savedata').innerHTML = "Design Already Exists. Please choose another name"; : Оновлює HTML-вміст елемента з ID savedata, відображаючи повідомлення про помилку, отриману на першому етапі збереження.

Цей код реалізує двохетапний процес збереження дизайну.

Перший етап: Надсилаються метадані дизайну (назва, опис тощо), SVG та структура Canvas без зображень. Цей етап, імовірно, призначений для перевірки унікальності назви дизайну або збереження базової інформації.

Другий етап: Якщо перший етап успішний, надсилається повний стан Canvas з усіма даними об'єктів, включаючи зображення. Цей етап, імовірно, призначений для збереження детального стану дизайну, який можна буде потім відновити.

Результат операції (успіх чи помилка першого етапу) відображається в елементі з ID savedata.

Використання AJAX дозволяє виконувати ці операції асинхронно, без перезавантаження сторінки, що покращує користувацький досвід. Забезпечується захист від CSRF-атак за допомогою токена.

3.7. Реалізація графічного інтерфейсу системи генерації дизайн-зображень на одязі

На рисунках 3.2 і 3.3 показано сторінки реєстрації та авторизації в системі.

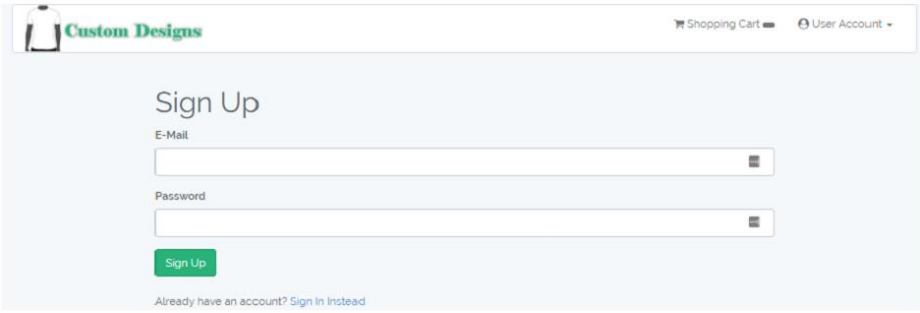


Рисунок 3.2 – Сторінка реєстрації

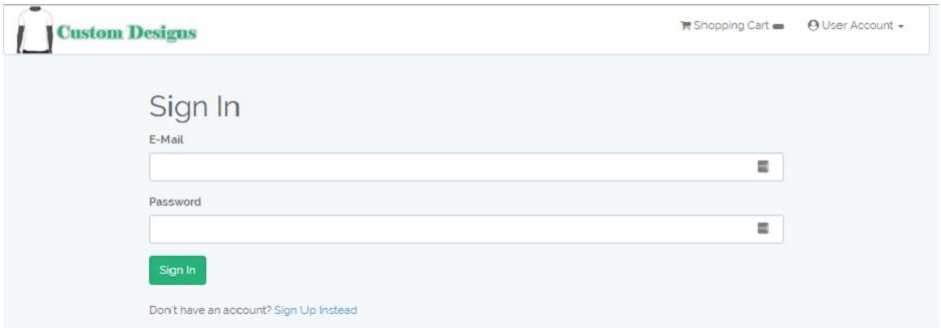


Рисунок 3.3 – Сторінка авторизації

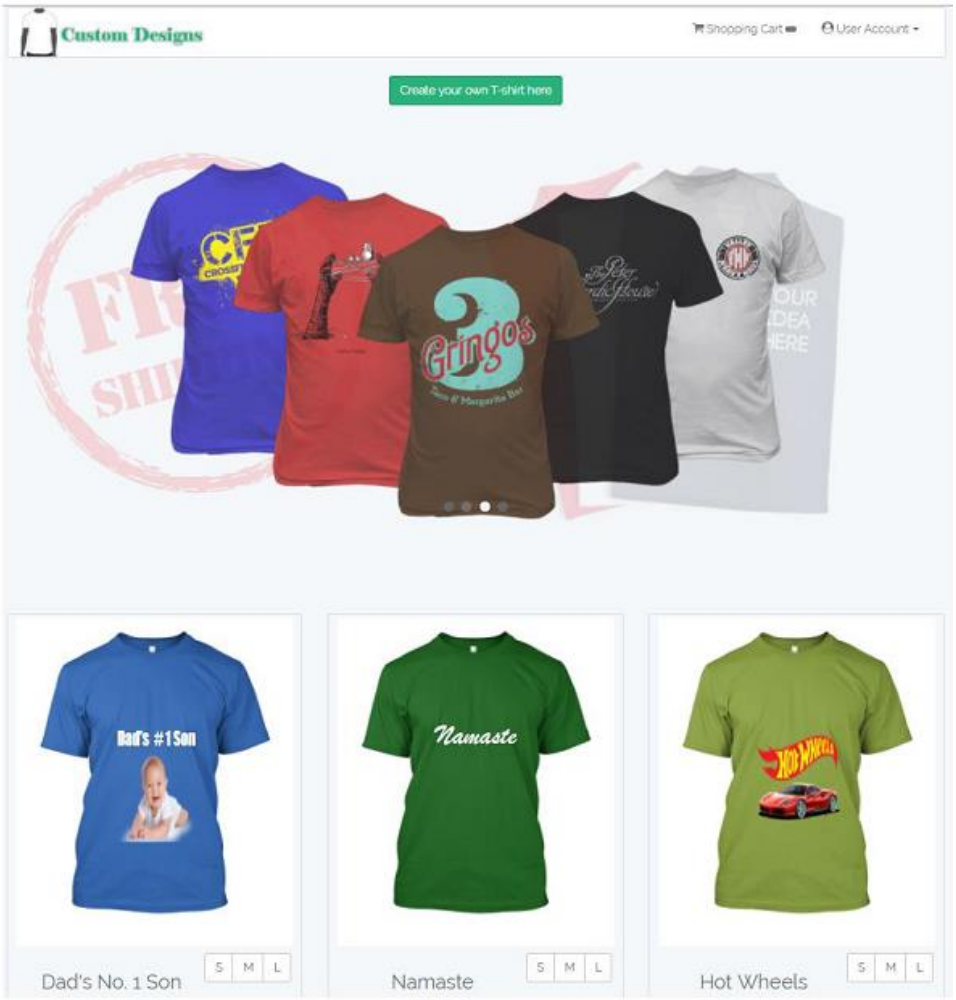


Рисунок 3.4 – Вигляд сторінки із продукцією

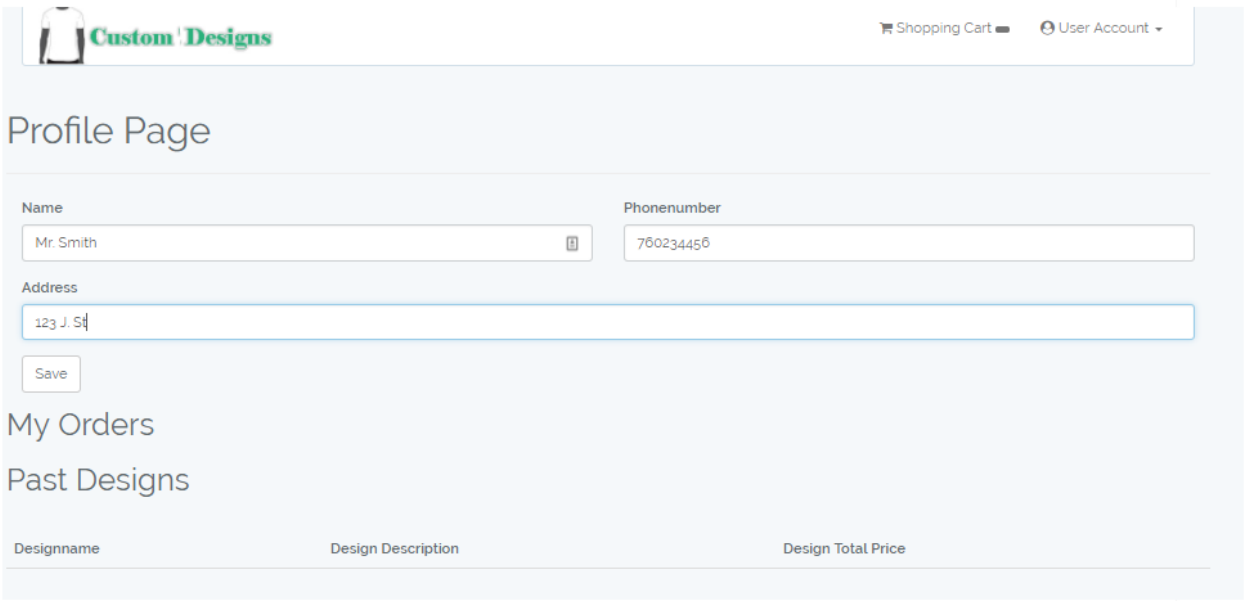


Рисунок 3.5 - Сторінка профілю

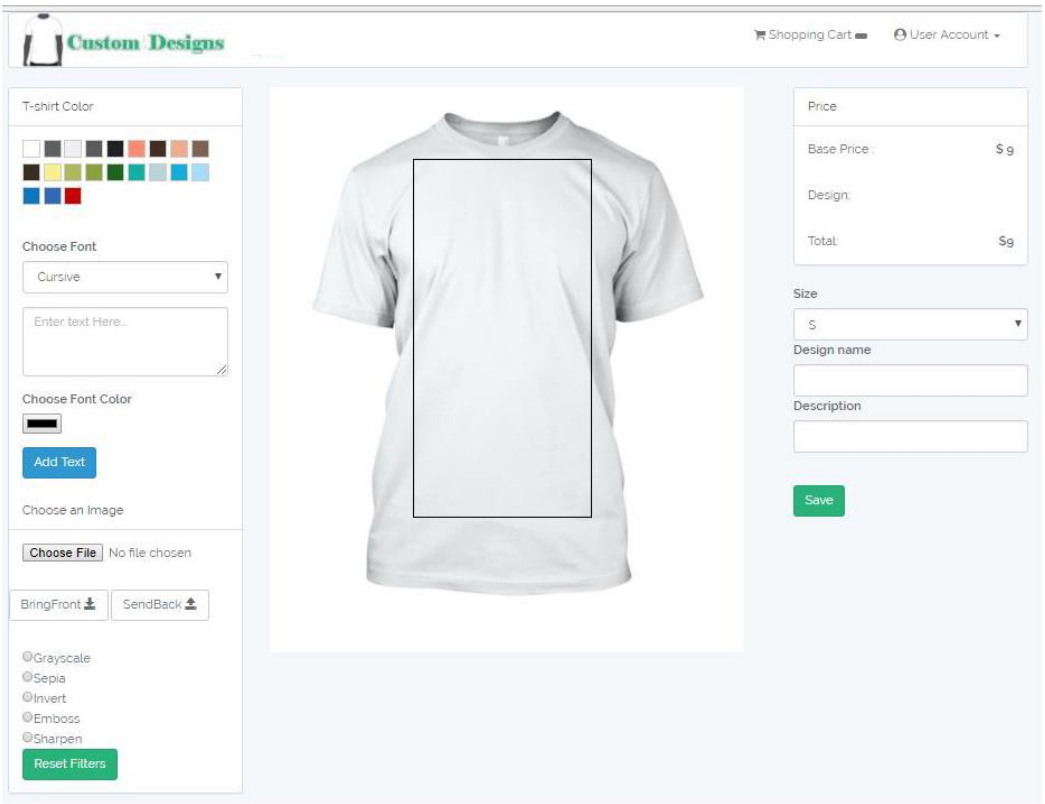


Рисунок 3.6 - Сторінка дизайну за замовчуванням

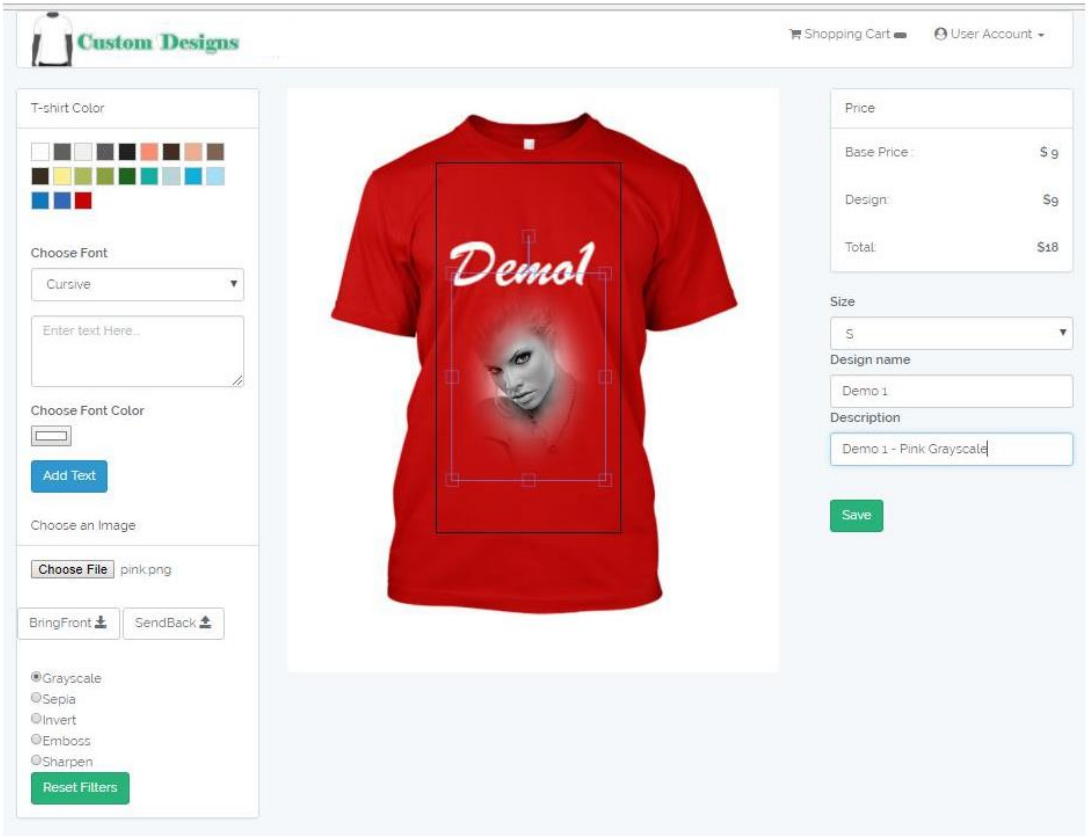


Рисунок 3.7 - Сторінка дизайну після оформлення

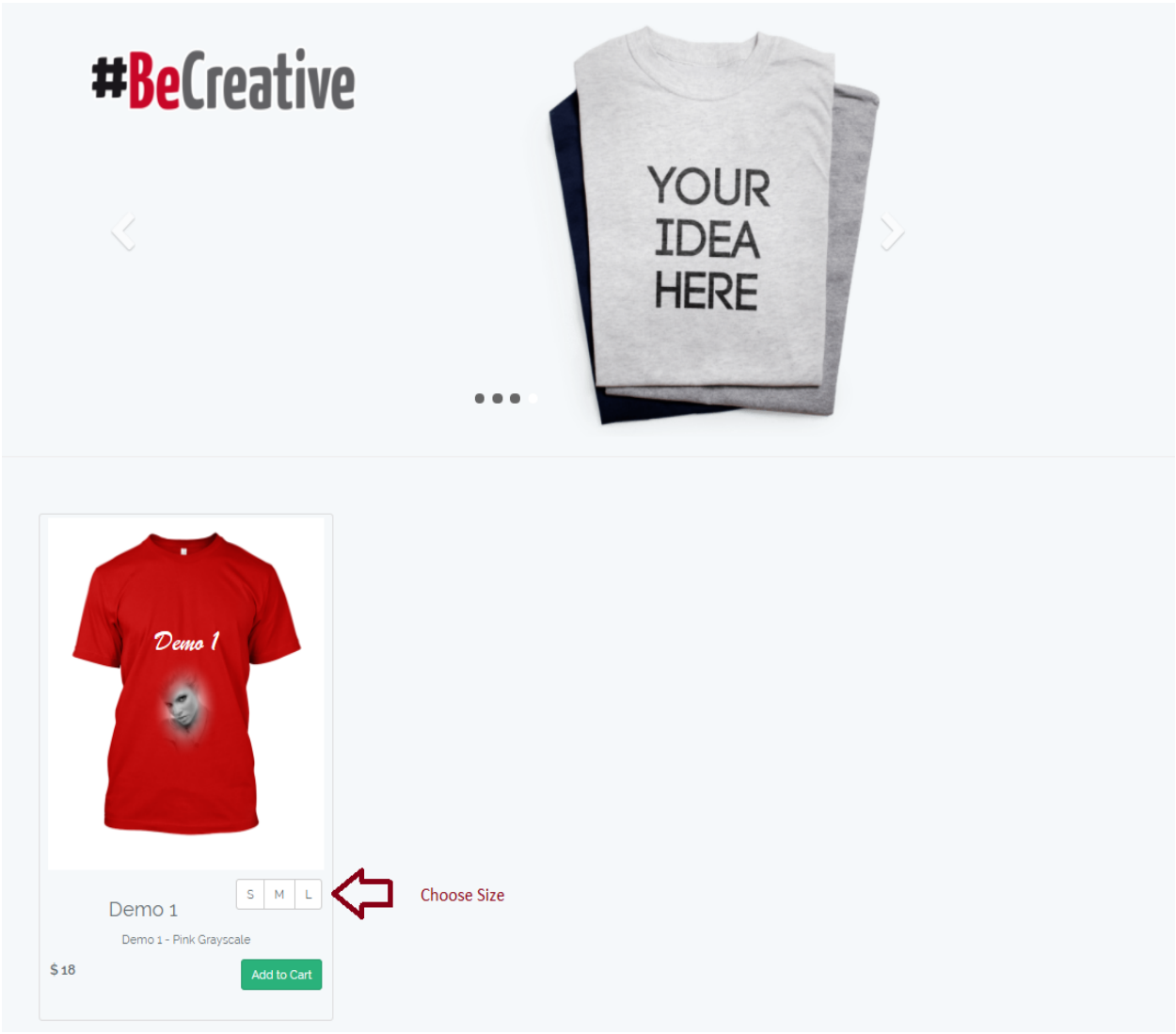


Рисунок 3.8 - Вибір розміру та додавання до кошика

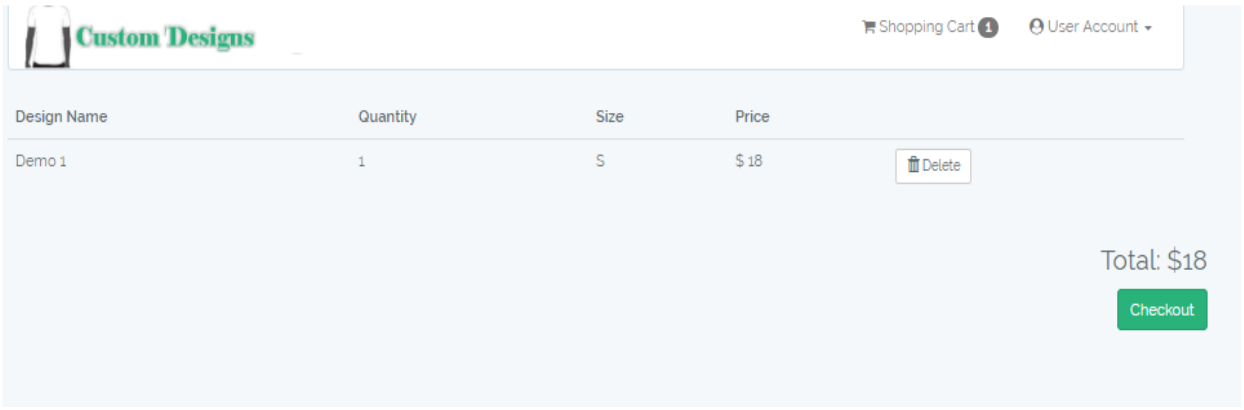
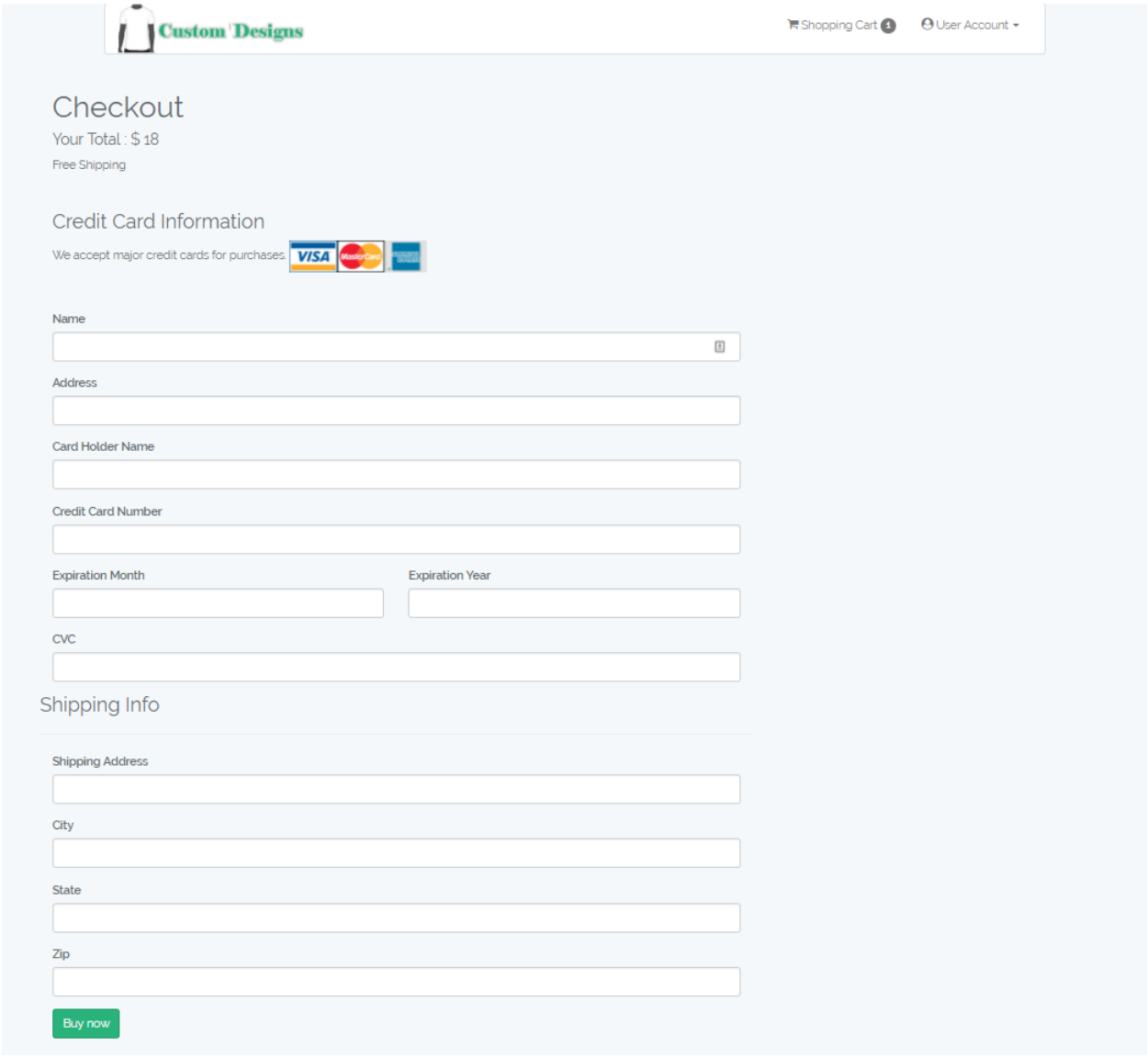


Рисунок 3.9 – Перегляд кошика



+

Рисунок 3.10 – Платіжна інформація

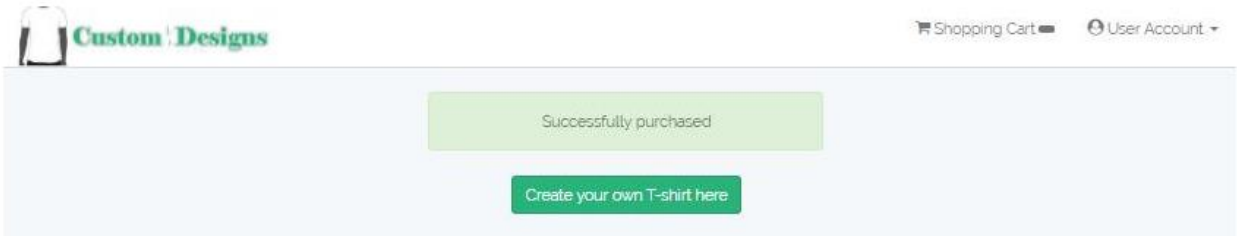


Рисунок 3.11 - Сповіщення про успішну покупку

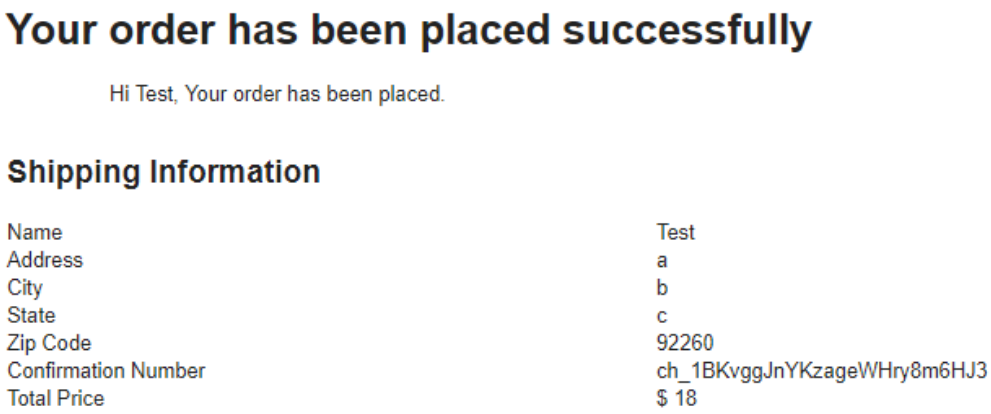


Рисунок 3.12 - Електронне сповіщення про покупку для клієнта

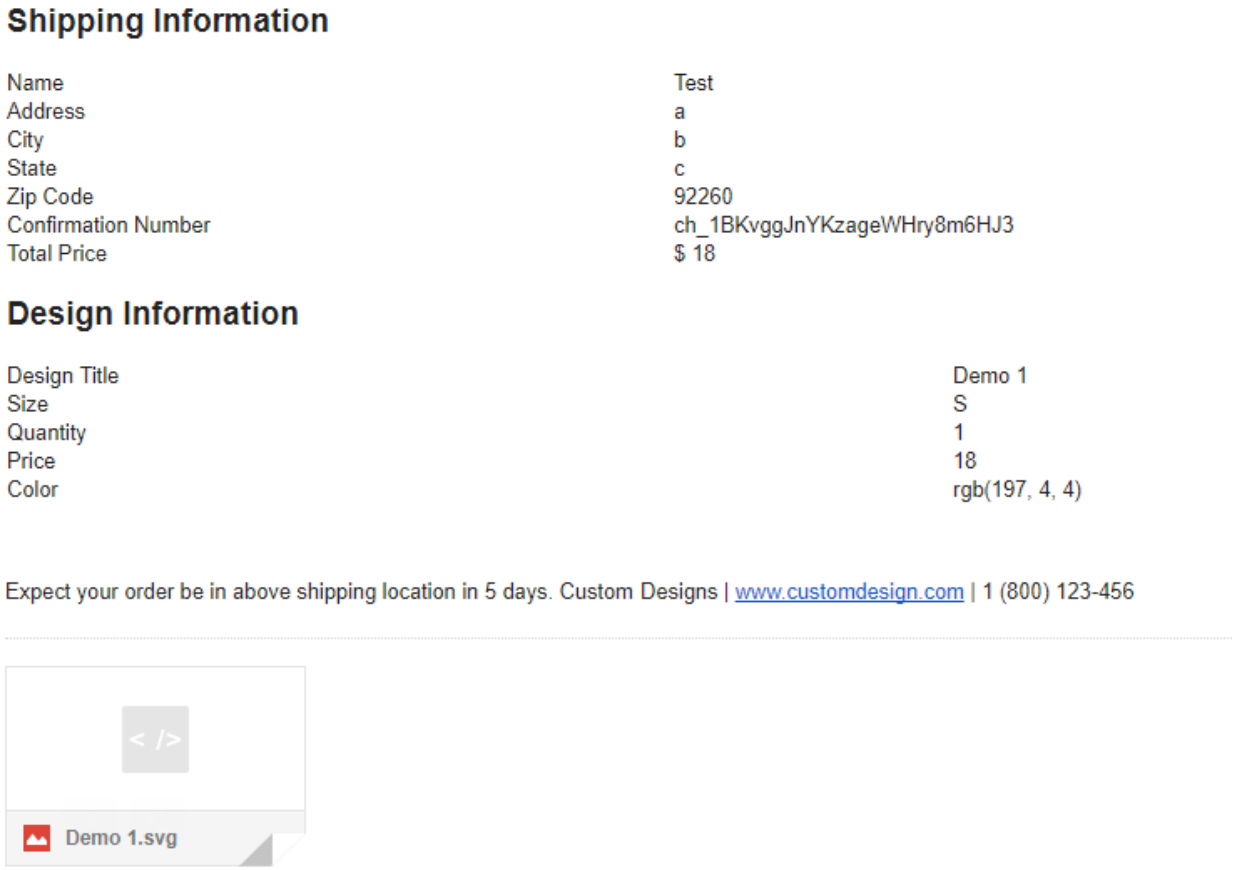


Рисунок 3.13 - Електронне сповіщення з вкладенням дизайну для адміністратора

Отже, реалізація даного проєкту базується на інтеграції шаблону програмної архітектури MVC (Model-View-Controller) з метою побудови високопродуктивного та стабільного веб-застосунку. Використання

технології JavaScript спільно з бібліотекою fabric.js дозволяє створити ефективний інструмент для розробки дизайнів власних футболок. Функціональні можливості застосунку передбачають надання користувачам інструментів для вибору готових шаблонів або створення індивідуальних дизайнів шляхом додавання та редагування текстових і графічних елементів, застосування фільтрів, а також збереження розроблених дизайнів для подальшого використання.

На майбутнє передбачається розширення функціоналу автентифікації користувачів шляхом інтеграції з сервісами соціальних мереж. Планується розширення бібліотеки готових графічних елементів, організованих за категоріями, як альтернатива можливості завантаження користувацьких зображень.

ВИСНОВКИ

В дипломній роботі було здійснено всебічний аналіз сучасних інформаційних технологій, що використовуються для індивідуалізації одягу шляхом генерації дизайн-зображень. Розглянуто функціональні можливості існуючих веб-застосунків та програмних рішень у сфері web-to-print, таких як Zakeke, Lumise Product Designer та Fancy Product Designer, що дало змогу окреслити актуальні підходи до інтерактивної кастомізації текстильних виробів.

На основі аналізу сформульовано цілі та завдання проєкту розробки власної системи генерації шаблонів дизайн-зображень на одязі, визначено її функціональні межі та цільову аудиторію. Особливу увагу приділено вибору оптимального технологічного стеку для реалізації системи: фреймворку Laravel для серверної логіки, СУБД MySQL для зберігання структурованих даних, а також бібліотек Fabric.js і Bootstrap — для реалізації динамічного клієнтського інтерфейсу та адаптивного дизайну відповідно.

У проєктній частині роботи спроєктовано архітектуру системи, реалізовано ключові компоненти серверної логіки (моделі, контролери, маршрутизація), а також створено інтерактивний графічний інтерфейс, який дозволяє завантажувати зображення, додавати текстові елементи та зберігати готові шаблони. Завдяки використанню Fabric.js реалізовано можливості drag-and-drop редагування, масштабування та позиціонування об’єктів на полотні, що значно підвищує юзабіліті розробленої системи.

Отримані результати підтверджують ефективність застосованих інструментів для побудови адаптивного web-додатку з інтерактивною генерацією дизайн-зображень. Запропонована система може бути використана як окремий сервіс, так і вбудовуватись у більші e-commerce рішення з кастомізацією продукції.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Lumise Designer Tool – #1 Product Designer Tool - https://lumise.com/lumise-designer-tool/
2. “Fabric.js is a powerful and simpleJavascript HTML5 canvas library,” Fabric.js Javascript Canvas Library. http://fabricjs.com/.
3. Fancy Product Designer For WooCommerce Plugin - https://woodev.net/product/fancy-product-designer-for-woocommerce-plugin/
4. “JavaScript,” Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Web/JavaScript.
5. “What is JavaScript?,” Mozilla Developer Network. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_step/What_is_JavaScript
6. What is MVC in PHP and how does it work? – Ozzu - https://www.ozzu.com/questions/610473/what-is-mvc-in-php-and-how-does-it-work
7. "Bootstrap (front-end framework)", En.wikipedia.org, https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework).
8. MySQL client – DbVisualizer - https://www.dbvis.com/database/mysql/
9. Bootstrap · The world's most popular mobile-first and responsive front-end framework. - https://getbootstrap.com/docs/3.3/
10. Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1995). Computer Graphics: Principles and Practice. Addison-Wesley.
11. Gonzalez, R. C., & Woods, R. E. (2018). Digital Image Processing (4th ed.). Pearson.
12. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

13. Shneiderman, B., & Plaisant, C. (2010). Designing the User Interface: Strategies for Effective Human-Computer Interaction. Addison-Wesley.
14. Kalakota, R., & Whinston, A. B. (1997). Electronic Commerce: A Manager's Guide. Addison-Wesley.
15. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. IEEE CVPR.
16. Leondes, C. T. (Ed.). (2002). Intelligent Systems: Technology and Applications. CRC Press.
17. McCormack, J., & d'Inverno, M. (Eds.). (2012). Computers and Creativity. Springer.
18. Radford, A. et al. (2021). Learning Transferable Visual Models From Natural Language Supervision (CLIP). arXiv:2103.00020.
19. Ramesh, A. et al. (2021). Zero-Shot Text-to-Image Generation. arXiv:2102.12092.
20. Stauffer, M., & Matt, R. (2019). Laravel: Up and Running (2nd ed.). O'Reilly Media.
21. Popovici, V. (2020). Mastering Laravel: PHP Web Development Made Easy. Packt Publishing.
22. Otwell, T. (2024). Laravel Documentation. https://laravel.com/docs
23. Hujanen, J. (2022). A Scalable Architecture for Laravel Applications. International Journal of Software Engineering and Applications.
24. Awwad, A. A. (2021). Design Patterns Implementation in Laravel Framework. Journal of Web Engineering.
25. Park, T., Liu, M. Y., Wang, T. C., & Zhu, J. Y. (2019). Semantic Image Synthesis with Spatially-Adaptive Normalization. IEEE CVPR.
26. Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. IEEE ICCV.

27. Xu, T. et al. (2018). AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. IEEE CVPR.
28. Karras, T., Laine, S., & Aila, T. (2019). A Style-Based Generator Architecture for Generative Adversarial Networks (StyleGAN). IEEE CVPR.
29. Reese, G. (2021). SQL and Relational Theory: How to Write Accurate SQL Code (3rd ed.). O'Reilly Media.
30. MySQL AB (2024). MySQL 8.0 Reference Manual. Oracle Corporation. https://dev.mysql.com/doc
31. Garcia-Molina, H., Ullman, J. D., & Widom, J. (2008). Database Systems: The Complete Book. Pearson.
32. Dziuba, B., & Borowski, M. (2022). Optimization Techniques for MySQL Databases. Lecture Notes in Computer Science
33. Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-Image Translation with Conditional Adversarial Networks. IEEE CVPR.
34. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI.
35. Frich, J., MacTavish, T., & Dalsgaard, P. (2019). Creativity Support Tools in the Wild: Understanding How Designers Use Generative Design Tools. CHI.
36. Lee, J. A. et al. (2020). Fashion++: Minimal Editing of Fashion Images to Improve Compatibility. IEEE ICCV.

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Розробка шаблонів генерації дизайн-зображень на одязі”

Обсяг пояснювальної записки: 79 аркушів.

Дата закінчення роботи: 10 червня 2025 р.

Підпис студента _____