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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка вебплатформи для автоматизації замовлень поліграфічної продукції

Виконав: студент IV курсу, групи СН-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Недашковський А. П.
(підпис) (прізвище та ініціали)

Керівник _____ Гром’як Р. С.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Липак Г.І.
(підпис) (прізвище та ініціали)

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

Рецензент _____ Луцик Н.С.
(підпис) (прізвище та ініціали)

Тернопіль
2026

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

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

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _червня_ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

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

Студенту Недашковський Андрій Петрович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб платформи для автоматизації замовлень поліграфічної продукції

Керівник роботи Гром'як Роман Сильвестрович к. ф. – м. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *травня* 2026 року № *4/7-239*

2. Термін подання студентом завершеної роботи 23 червня 2026 р.

3. Вихідні дані до роботи Літературні та інтернет джерела інформації по технології розробки програмного комплексу для розробки веб платформи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Аналіз предметної області та обґрунтування технічних вимог.
Розділ 2. Моделювання та структурно-логічне проєктування системи.
Розділ 3. програмна реалізація та тестування вебплатформи
Розділ 4. Безпека життєдіяльності, основи охорони праці. Висновки.
Перелік джерел. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульна сторінка. 2. Мета кваліфікаційної роботи. 3. Актуальність обраної теми. 4. Порівняльний аналіз рішень. 5. Обґрунтування стеку MERN. 6. Проєктування бази даних MongoDB. 7. Математична модель калькулятора. 8. Конвеєр обробки графічних макетів. 9. Архітектура взаємодії компонентів. 10. Результати навантажувального тестування. 11. Продуктивність та надійність. 12. Головне меню. 13. Віджет-чат. 14. Сторінка створення Замовлення. 15. Особистий кабінет користувача (менеджер). 16. Панель керування статусами акаунтів (модератор). 17. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик Олег Ярославович, к.т.н. доцент кафедри МТ		

7. Дата видачі завдання 26 січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.2026	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	08.05.2026-15.05.2026	*Виконано*
3.	Виконання дослідження щодо ринку поліграфічної продукції	17.02.2026-10.05.2026	*Виконано*
	Розроблення проекту веб-платформи для онлайн замовлення поліграфічної продукції		*Виконано*
4.	Оформлення розділу «Аналіз предметної області та обґрунтування технічних вимог».	11.05.2026-17.05.2026	*Виконано*
	»		
5.	Оформлення розділу «Моделювання та структурно-логічне проєктування системи».	18.05.2026-24.05.2026	*Виконано*
	»		
6.	Оформлення розділу «Оформлення розділу «Програмна реалізація та тестування вебплатформи».	25.05.2026-31.05.2026	*Виконано*
	»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	01.06.2026-08.06.2026	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	01.06.2026-08.06.2026	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	09.06.2026-11.06.2026	*Виконано*
10.	Нормоконтроль	12.06.2026-15.06.2026	*Виконано*
11.	Перевірка на плагіат	16.06.2026	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2026	*Виконано*
13.	Захист кваліфікаційної роботи	24.06.2026	

Студент _____ Недашковський А. П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гром'як Р. С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка веб платформи для автоматизації замовлень поліграфічної продукції // Кваліфікаційна робота освітнього ступеня «Бакалавр» // Недашковський Андрій Петрович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-42 // Тернопіль, 2026 // С. 72, рис. – 7 , табл. – 3 , , додат. – 2 , бібліогр. – 43 .

Ключові слова: вебплатформа, автоматизація замовлень, поліграфія, web-to-print, стек mern, react.js, node.js, mongodb, калькулятор друку, макет.

Кваліфікаційна робота присвячена дослідженню процесів автоматизації прийому, обробки та розрахунку параметрів замовлень у сфері оперативної поліграфії за допомогою сучасних вебтехнологій.

У першому розділі описано особливості впровадження концепції Web-to-Print у сучасній поліграфічній індустрії. Проведено порівняльний аналіз існуючих комерційних та відкритих програмних рішень на ринку, виявлено їхні недоліки та обґрунтовано необхідність розробки власної кастомної системи. Сформовано перелік функціональних і нефункциональних вимог, а також здійснено інженерне обґрунтування вибору технологічного стеку MERN (MongoDB, Express.js, React.js, Node.js).

У другому розділі здійснено структурно-логичне проєктування архітектури системи. Розроблено UML-діаграму прецедентів для рольової моделі акторів та побудовано концептуальну модель взаємодії компонентів платформи за принципом SPA та RESTful API. Описано проектування NoSQL структури бази даних MongoDB із застосуванням патерну «Snapshot». Формалізовано нелінійну математичну модель і алгоритм динамічного розрахунку вартості накладів на основі мультиплікативних коефіцієнтів та

ступеня регресії тиражу. Проєктовано логіку двоступеневої валідації та асинхронного конвеєра обробки бінарних файлів макетів.

У третьому розділі описано програмну реалізацію клієнтської частини на React.js, включаючи інтерактивний калькулятор та WebSocket-віджет підтримки в реальному часі. Реалізовано серверну логіку Express API, модулі інтеграції з хмарним сховищем за допомогою Multer, а також адміністративний інтерфейс. Проведено комплексне модульне, інтеграційне та навантажувальне тестування, яке підтвердило високу продуктивність та стійкість системи (до 750 запитів/сек).

У четвертому розділі проведено аналіз умов праці розробника й оператора, а також розроблено заходи з техніки безпеки, електробезпеки та пожежної безпеки відповідно до чинних нормативних вимог України.

Об’єкт дослідження: процеси автоматизації прийому, обробки та розрахунку параметрів замовлень у сфері надання поліграфічних послуг за допомогою вебтехнологій.

Предмет дослідження: методи, алгоритми, архітектурні патерни та програмні засоби розробки клієнт-серверних вебзастосунків для онлайн-оформлення замовлень друку.

ANNOTATION

Development of a Web Platform for Automating Print Product Orders // Qualification paper for the Bachelor's degree // Nedashkovskyi Andrii Petrovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group SN-42 // Ternopil, 2026 // P. 72, figs. 7, tabs. 3, app. 2, bibliogr. 43.

Keywords: web platform, order automation, printing, web-to-print, MERN stack, react.js, node.js, mongodb, print calculator, layout.

The qualification paper is devoted to the research of processes for automating the receipt, processing, and calculation of order parameters in the field of quick printing using modern web technologies.

The first chapter describes the features of implementing the Web-to-Print concept in the modern printing industry. A comparative analysis of existing commercial and open-source software solutions on the market is conducted, their shortcomings are identified, and the necessity of developing a custom system is justified. A list of functional and non-functional requirements is formed, and an engineering justification for the choice of the MERN technology stack (MongoDB, Express.js, React.js, Node.js) is provided.

The second chapter performs the structural and logical design of the system architecture. A UML use case diagram for the role model of actors is developed, and a conceptual model of interaction between platform components based on the SPA and RESTful API principles is constructed. The design of the MongoDB NoSQL database structure using the "Snapshot" pattern is described. A non-linear mathematical model and an algorithm for dynamic calculation of print run costs based on multiplicative coefficients and the degree of print run regression are formalized. The logic of two-stage validation and an asynchronous pipeline for processing binary layout files is designed.

The third chapter describes the software implementation of the client side using React.js, including an interactive calculator and a real-time WebSocket support widget. The Express API server logic, integration modules with cloud storage using Multer, and the administrative interface are implemented. Comprehensive unit, integration, and load testing are conducted, confirming the high performance and stability of the system (up to 750 requests/sec).

The fourth chapter analyzes the working conditions of the developer and the operator, and develops measures for safety, electrical safety, and fire safety in accordance with the current regulatory requirements of Ukraine.

Object of research: processes of automating the receipt, processing, and calculation of order parameters in the field of printing services using web technologies.

Subject of research: methods, algorithms, architectural patterns, and software tools for developing client-server web applications for online print ordering.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. *Application Programming Interface*) – прикладний програмний інтерфейс, набір визначень, протоколів та інструментів для створення програмного забезпечення та інтеграції додатків.

BSON (англ. *Binary JSON*) – бінарний формат обміну даними, що використовується як основне сховище для документів у NoSQL СКБД MongoDB.

CRUD (англ. *Create, Read, Update, Delete*) – базові операції управління даними (створення, читання, оновлення, видалення).

CSS (англ. *Cascading Style Sheets*) – каскадні таблиці стилів, що використовуються для опису зовнішнього вигляду та форматування вебсторінок.

DOM (англ. *Document Object Model*) – об'єктна модель документа, програмний інтерфейс, який дозволяє скриптам динамічно отримувати доступ та оновлювати вміст, структуру та стилі документів.

DPI (англ. *Dots Per Inch*) – кількість точок на дюйм, міра роздільної здатності зображення при друці.

EAV (англ. *Entity-Attribute-Value*) – патерн (шаблон) проєктування реляційних баз даних для гнучкого опису сутностей за допомогою тріади «сутність—атрибут—значення».

HTML (англ. *HyperText Markup Language*) – стандартизована мова розмітки документів для перегляду вебсторінок у браузері.

HTTP / HTTPS (англ. *HyperText Transfer Protocol / Secure*) – протокол передачі гіпертексту (захищений), основний мережевий протокол для обміну даними в інтернеті.

JSON (англ. *JavaScript Object Notation*) – текстовий формат обміну даними, що базується на JavaScript, легкий для читання та написання як людьми, так і комп'ютерами.

JWT (англ. *JSON Web Token*) – відкритий стандарт для безпечної передачі токенів авторизації між сторонами у вигляді об'єкта JSON.

MERN (англ. *MongoDB, Express.js, React.js, Node.js*) – сучасний технологічний стек розробки вебзастосунків на основі мови програмування JavaScript.

MIME (англ. *Multipurpose Internet Mail Extensions*) – стандарт, що описує типи даних (аудіо, відео, зображення, тексти), які передаються через мережу Інтернет.

NoSQL (англ. *Not Only SQL*) – загальна назва некомерційних баз даних, що відрізняються від класичних реляційних моделей (зокрема, документ-орієнтовані системи).

ODM (англ. *Object-Document Mapping*) – технологія об'єктно-документного відображення, що зв'язує структури документів NoSQL з об'єктами в коді програми.

REST / RESTful (англ. *Representational State Transfer*) – архітектурний стиль взаємодії компонентів розподіленого вебзастосунку за допомогою стандартних HTTP-методів.

RWD (англ. *Responsive Web Design*) – адаптивний вебдизайн, підхід у розробці інтерфейсів, що забезпечує коректне відображення сайту на різних пристроях.

SaaS (англ. *Software as a Service*) – програмне забезпечення як послуга, бізнес-модель, за якої користувачі отримують доступ до готового хмарного софту за підпискою.

SPA (англ. *Single Page Application*) – односторінковий вебзастосунок, вебсайт, який завантажує єдиний HTML-документ і динамічно оновлює сторінку під час взаємодії з користувачем.

URL (англ. *Uniform Resource Locator*) – єдиний вказівник ресурсу, стандартизована адреса певного об'єкта (файлу, сторінки) в інтернеті.

UX (англ. *User Experience*) – досвід користувача, сукупність відчуттів, вражень та відповідей користувача під час взаємодії з інтерфейсом програмного продукту.

W2P / Web-to-Print – технологічна концепція автоматизації комерційної поліграфії, що дозволяє віддалено конфігурувати та замовляти друк через вебсайти.

ЕОМ – електронно-обчислювальна машина (персональний комп'ютер, сервер).

ПЗВ – пристрій захисного відключення, електротехнічний пристрій, призначений для захисту людини від ураження струмом та запобігання пожежам.

ПУЕ – Правила улаштування електроустановок, основний нормативний документ для проектування та монтажу електричних систем в Україні.

СКБД – система управління базами даних, комплекс програмних засобів для створення, ведення та спільного використання баз даних.

ШІ – штучний інтелект.

ЗМІСТ

ВСТУП

Актуальність теми. Активний розвиток цифрових технологій та електронної комерції суттєво вплинув на способи взаємодії між підприємствами та споживачами. Ці зміни торкнулися й поліграфічної галузі, де традиційні підходи до прийому та обробки замовлень поступово поступаються місцем автоматизованим інформаційним системам. У багатьох друкарнях оформлення замовлень досі пов’язане з телефонними консультаціями, листуванням електронною поштою, ручним розрахунком вартості продукції та перевіркою макетів працівниками підприємства. Такий підхід потребує значних витрат часу й підвищує ризик виникнення помилок під час обробки замовлень.

Одним із напрямів вирішення зазначених проблем є використання технології Web-to-Print, яка передбачає надання клієнтам можливості самостійно оформлювати замовлення через вебінтерфейс. За допомогою таких систем користувач може обрати параметри друкованої продукції, отримати попередній розрахунок вартості, завантажити макет і передати замовлення в обробку без безпосередньої участі менеджера. Це сприяє скороченню часу обслуговування клієнтів, спрощує організацію виробничих процесів і дає змогу ефективніше використовувати ресурси підприємства.

У зв’язку з цим розробка вебплатформи для автоматизації процесів онлайн-замовлення поліграфічної продукції є актуальним завданням, що має як практичне, так і прикладне значення для сучасних друкарських підприємств.

Мета і задачі дослідження. Метою кваліфікаційної роботи є проєктування та реалізація вебплатформи, призначеної для автоматизації оформлення замовлень поліграфічної продукції, розрахунку їх вартості та організації взаємодії між клієнтом і друкарнею.

Для досягнення поставленої мети необхідно виконати такі завдання:

- дослідити особливості організації процесів прийому та обробки замовлень у сфері оперативної поліграфії;

- провести аналіз існуючих програмних рішень класу Web-to-Print та визначити їхні переваги й недоліки;
- сформувати функціональні та нефункціональні вимоги до розроблюваної системи;
- спроєктувати архітектуру вебплатформи на основі клієнт-серверної моделі взаємодії;
- розробити структуру бази даних для зберігання інформації про користувачів, замовлення та параметри друку;
- створити алгоритм автоматизованого розрахунку вартості поліграфічної продукції залежно від обраних характеристик;
- реалізувати механізми завантаження, перевірки та зберігання файлів макетів;
- розробити користувацький інтерфейс та адміністративну панель керування;
- провести тестування програмного забезпечення та оцінити результати його роботи.

Об’єкт дослідження – процеси автоматизації прийому, обробки та розрахунку параметрів замовлень у сфері надання поліграфічних послуг за допомогою вебтехнологій.

Предмет дослідження – методи, алгоритми, архітектурні патерни та програмні засоби розробки клієнт-серверних вебзастосунків (зокрема на базі технологічного стеку MERN) для конфігурування та онлайн-оформлення замовлень друку.

Практичне значення одержаних результатів полягає у створенні готового до розгортання та комерційного використання програмного комплексу друкарні. Впровадження розробленої вебплатформи дозволяє повністю автоматизувати цикл взаємодії «клієнт – менеджер – виробництво». Завдяки реалізованому динамічному калькулятору та модулю завантаження файлів, підприємство отримує можливість знизити операційні витрати, прискорити виконання замовлень та забезпечити прозоре ціноутворення для користувачів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБҐРУНТУВАННЯ ТЕХНІЧНИХ ВИМОГ

1.1. Особливості автоматизації бізнес-процесів у сучасній поліграфії (концепція Web-to-Print)

У сучасних умовах розвитку цифрових технологій підприємства різних галузей активно впроваджують інформаційні системи для автоматизації виробничих та управлінських процесів. Поліграфічна галузь також поступово переходить до використання цифрових інструментів, що дозволяють підвищити ефективність роботи підприємств та покращити якість обслуговування клієнтів. Особливо актуальним це питання є для компаній, які працюють у сфері оперативного друку та щоденно обробляють значну кількість замовлень різної складності.

Традиційна схема взаємодії між клієнтом і друкарнею зазвичай передбачає кілька послідовних етапів. Спочатку замовник звертається до підприємства телефоном, електронною поштою або через месенджери та повідомляє необхідні параметри продукції. Після цього менеджер уточнює характеристики майбутнього виробу, зокрема формат, тип паперу, тираж, кольоровість друку та додаткові види обробки. На основі отриманої інформації здійснюється розрахунок вартості замовлення, а клієнт надсилає файл макета для подальшої перевірки. У разі виявлення помилок або зміни параметрів продукції процедура погодження може повторюватися кілька разів.

Подібна організація роботи потребує значних трудових витрат і значною мірою залежить від людського фактора. Помилки під час введення даних, неправильний розрахунок вартості або несвоєчасне виявлення недоліків у макеті можуть призвести до затримок у виробництві та додаткових фінансових витрат. Крім того, зі збільшенням кількості замовлень навантаження на

персонал підприємства постійно зростає, що ускладнює підтримання належного рівня обслуговування клієнтів.

Одним із найбільш поширених підходів до вирішення цих проблем є використання технології Web-to-Print (W2P). Вона поєднує можливості вебтехнологій та засобів автоматизації поліграфічного виробництва, забезпечуючи користувачам віддалений доступ до процесу оформлення замовлень. Завдяки такому підходу значна частина операцій, які раніше виконувалися менеджерами вручну, може бути автоматизована та перенесена безпосередньо до вебсистеми.

Використовуючи платформу класу Web-to-Print, клієнт отримує можливість самостійно обирати характеристики продукції, переглядати актуальну вартість замовлення та завантажувати файли макетів через вебінтерфейс. Система контролює коректність введених даних і може обмежувати вибір параметрів, які є несумісними між собою. Такий підхід допомагає уникати помилок ще на етапі оформлення замовлення.

Важливою складовою подібних платформ є автоматизований розрахунок вартості продукції. Зміна будь-якого параметра замовлення миттєво враховується системою, а користувач одразу отримує оновлену інформацію щодо ціни. Це дозволяє скоротити час прийняття рішення та усуває необхідність звернення до менеджера для отримання попереднього кошторису.

Ще однією перевагою є централізована робота з файлами макетів. Документи завантажуються безпосередньо через вебплатформу та зберігаються в єдиному середовищі, що спрощує їх подальшу обробку й зменшує ризик втрати даних. За потреби система може виконувати первинну перевірку файлів на відповідність встановленим технічним вимогам.

Таким чином, впровадження концепції Web-to-Print сприяє підвищенню ефективності роботи поліграфічних підприємств, скороченню часу обробки замовлень та покращенню взаємодії з клієнтами. Автоматизація ключових бізнес-процесів дозволяє оптимізувати використання ресурсів підприємства та

створює передумови для подальшого розвитку онлайн-сервісів у сфері поліграфії.

1.2. Порівняльний аналіз існуючих рішень та аналогів на ринку онлайн-поліграфії.

Для того, щоб спроектувати ефективну архітектуру та функціональні можливості веб-платформи, необхідно провести детальний аналіз існуючих програмних продуктів і бізнес-моделей, представлених на ринку онлайн-поліграфії. Сучасні рішення можна розподілити на три основні категорії: закриті пропрієтарні системи великих гравців, спеціалізовані SaaS-платформи (Software as a Service), а також універсальні CMS-системи (Content Management System) з додатковими плагінами для електронної комерції.

Прикладом пропрієтарної системи на українському ринку є інтернет-друкарня Wolf.ua. Платформа має розвинену інфраструктуру, а також інтерактивні калькулятори для різноманітних видів продукції, автоматизовану перевірку макетів та власний дизайнерський портал. Система працює швидко і оптимізована для власних виробничих потужностей компанії. Однак, цей продукт є повністю закритим і внутрішнім інструментом, що робить його недоступним для ліцензування чи адаптації під інші підприємства. Це суттєво обмежує використання цього рішення як готової платформи для сторонніх користувачів.

Серед SaaS-рішень варто виділити платформи на кшталт Pressero від компанії Aleyant та PrintShop Mail. Вони пропонують хмарні кабінети, шаблони для дизайну та базові модулі калькуляторів за моделлю підписки. Головними перевагами таких платформ є швидке розгортання та готовий функціонал з коробки. Проте, дослідження також показують суттєві недоліки таких сервісів для малого та середнього бізнесу.

По-перше, це висока загальна вартість володіння: регулярні абонентські платежі і комісії з операцій створюють фінансове навантаження. По-друге, обмежені можливості кастомізації не дозволяють впроваджувати унікальні бізнес-логіки, наприклад, авторські алгоритми розрахунку вартості, що

враховують специфіку конкретного друкарського обладнання. І нарешті, SaaS-моделі несуть ризики інформаційної безпеки через розміщення даних на серверах сторонніх провайдерів.

Третій варіант — використання безкоштовних або частково платних CMS-платформ, таких як WordPress з плагінами WooCommerce, OpenCart або Magento, доповнених модулями для розрахунку вартості замовлень. Це дає повний контроль над кодом і даними, але має проблеми з продуктивністю. Враховуючи складність поліграфічних калькуляторів, де ціна залежить від великої кількості взаємопов'язаних параметрів, використання реляційних баз даних у таких системах призводить до утворення важких SQL-запитів з численними об'єднаннями, що знижує швидкість роботи і збільшує навантаження на сервер.

Для узагальнення результатів цього дослідження була створена порівняльна матриця за ключовими техніко-економічними показниками (див. таблицю 1.1). Вона підкреслює існування ніші для гнучкої, швидкодіючої, автономної і водночас доступної системи для автоматизації замовлень у поліграфії.

Таблиця 1.1 – Порівняльна матриця технічних рішень на ринку онлайн-поліграфії

Критерій порівняння	**Пропрієтарні системи (на прикладі Wolf.ua)**	**Спеціалізовані SaaS-платформи (Pressero)**	**Універсальні CMS + плагіни (WordPress)**	**Розроблювана веб-платформа (Стек MERN)**
1	**2**	**3**	**4**	**5**
Архітектурний тип	Закрита монолітна система	Хмарна мультитенантна архітектура	Монолітна архітектура з плагінами	Децентралізована архітектура (SPA + REST API)

Продовження таблиці 1.1

1	2	3	4	5
Фінансові витрати на впровадження	Недоступно для придбання	Висока абонентська плата + % від продажів	Середня (купівля плагінів, налаштування)	Мінімальна (витрати на хостинг та підтримку)
Рівень кастомізації бізнес-логіки	Відсутній (система жорстко зафіксована)	Низький (лише в межах налаштувань провайдера)	Середній (обмежений архітектурою CMS)	Абсолютний (повний контроль над кодом модулів)
Продуктивність калькулятора при навантаженні	Висока (завдяки потужним серверам)	Середня (залежить від хмари провайдера)	Низька (через надлишкові SQL-запити)	Максимальна (Virtual DOM React + асинхронний Node.js)
Гнучкість структури даних продукції	Висока (адаптована під власне виробництво)	Шаблонна (важко додавати нові типи параметрів)	Обмежена реляційною схемою БД (EAV-патерн)	Максимальна (схема-лес NoSQL документи JSON)
Контроль та безпека даних	Повний (всередині компанії-власника)	Відсутній (дані у стороннього провайдера)	Повний (на власному сервері)	Повний (абсолютний контроль над БД та сховищем)

Проведений аналіз чітко вказує на наявність незаповненої ніші: ринок потребує гнучкого, високопродуктивного, автономного та економічно доступного рішення для автоматизації поліграфічних замовлень. Створення власної веб-платформи на базі сучасних інструментів веб-розробки дозволяє ліквідувати недоліки існуючих аналогів, забезпечуючи максимальну швидкість обчислень, гнучкість налаштувань під будь-яке обладнання та повну незалежність бізнесу.

1.3. Формування функціональних та нефункціональних вимог до вебплатформи

Для створення якісного програмного продукту важливо чітко сформулювати вимоги до системи, які відображають її поведінку та характеристики. В результаті аналізу бізнес-процесів у поліграфії було визначено низку основних функціональних вимог, які забезпечують повний цикл роботи вебплатформи.

Система передбачає наявність різних ролей користувачів із відповідними правами доступу, побудованих на основі рольової моделі керування доступом (RBAC).

Модуль автентифікації, реєстрації та керування профілем:

- Система повинна забезпечувати реєстрацію нових клієнтів шляхом заповнення форми (ПІБ, Email, телефон, безпечний пароль).
- Система повинна проводити авторизацію користувачів на основі верифікації пари Email/Пароль та генерувати захищений токен сесії.
- Система повинна надавати Клієнту доступ до Особистого кабінету, де реалізовано відображення історії його замовлень, поточних статусів виробництва та персональних даних.

Модуль інтерактивного калькулятора-конфігуратора:

- Система повинна надавати графічний інтерфейс для вибору технічних параметрів замовлення: категорія виробу, геометричний формат, тип та щільність паперу, кольоровість друку, тип післядрукарської обробки, об'єм тиражу.
- Клієнтська частина модуля (calculator.js) повинна перераховувати вартість накладу в реальному часі (on-the-fly) при зміні будь-якого з параметрів користувачем, без перезавантаження веб-сторінки.
- Система повинна блокувати відправку форми у разі введення некоректних або екстремальних значень тиражу (наприклад, нульовий, від'ємний або нечисловий ввід).

- Модуль обробки бінарних даних (Макетів):
- Система повинна надавати інструмент для завантаження файлів цифрових макетів безпосередньо під час конфігурування замовлення.
- На стороні клієнта повинна виконуватися первинна валідація файлу за розміром та розширенням.
- Серверна частина повинна забезпечувати надійне потокове приймання файлів, їх перейменування за унікальним маскувальним шаблоном (для запобігання конфліктів імен) та перенаправлення у сховище із записом URL-посилання в документ замовлення.

Модуль адміністрування та управління виробництвом:

- Система повинна надавати Адміністратору захищений інтерфейс (dashboard.html) для моніторингу всіх замовлень у системі.
- Адміністратор повинен мати функціональну можливість змінювати виробничі статуси замовлень, що автоматично відображатиметься в кабінеті відповідного клієнта.
- Адміністратор повинен мати доступ до завантаження оригінальних файлів макетів клієнтів для їх передачі на виробничі друкарські машини.
- Система повинна дозволяти адміністратору редагувати прайс-листи: змінювати базові вартості приладки та коефіцієнти матеріалів через інтерфейс управління прайсами.

Модуль оперативної підтримки:

- Система повинна містити інтегрований віджет чату для забезпечення прямої асинхронної комунікації між Клієнтом та Адміністратором у режимі реального часу.

Нефункціональні вимоги визначають критерії якості роботи системи, її експлуатаційні характеристики, обмеження середовища та вимоги до надійності.

Продуктивність та швидкість відгуку (Performance):

- Час рендерингу підсумкової вартості калькулятором після зміни параметра користувачем не повинен перевищувати 200 мс.
- Час відповіді сервера на стандартні REST API запити (за винятком завантаження великих файлів) не повинен перевищувати 1 секунди при 50 одночасних з'єднаннях.

Адаптивність та кросбраузерність (Usability):

- Веб-інтерфейс платформи повинен відповідати принципам Responsive Web Design (RWD) – коректно та без втрати функціональності відображатися на десктопах, планшетах та мобільних пристроях з шириною екрана від 320px до 1920px.
- Додаток повинен стабільно працювати у всіх сучасних браузерах (Google Chrome, Mozilla Firefox, Apple Safari, Microsoft Edge).

Масштабованість та архітектурна гнучкість (Scalability):

- Архітектура бази даних та серверного API повинна дозволяти додавання нових категорій товарів із довільними наборами характеристик без модифікації існуючої схеми даних.

Безпека та захист інформації (Security):

- Передача будь-яких даних між клієнтом та сервером повинна здійснюватися за захищеним протоколом HTTPS.
- Паролі користувачів у базі даних повинні зберігатися виключно в криптографічно хешованому вигляді із застосуванням алгоритму підвищеної стійкості (наприклад, bcrypt).
- Авторизація сесій повинна базуватися на технології безстатусного обміну токенами JWT. Токени мають зберігатися в захищених контейнерах браузера (HttpOnly Cookie), що мінімізує ризики XSS-атак.
- Завантаження файлів на сервер має супроводжуватися суворим контролем MIME-типів для блокування спроб завантаження шкідливих виконуваних скриптів (наприклад, .exe, .sh, .php).

1.4. Обґрунтування вибору архітектурного паттерну та технологічного стеку (MERN: React, Node.js, Express, MongoDB).

Для успішної реалізації всієї функціональності вебплатформи важливо було визначитися з оптимальною архітектурою і технологіями, що відповідають вимогам сучасних інтернет-додатків та специфіці поліграфічної сфери.

У традиційних вебсистемах зміна даних на сервері супроводжується повним перезавантаженням сторінок у браузері, що значно уповільнює роботу складних інтерактивних форм. Особливо для калькулятора поліграфічних замовлень, де користувач може часто і швидко змінювати параметри, це є неприйнятним через значні мережеві затримки та погіршення користувацького досвіду.

Щоб уникнути цих проблем, у проекті застосовано архітектурний підхід Single Page Application (SPA) із використанням бібліотеки React.js. Цей фреймворк дозволяє виконати одноразове завантаження вебдодатку, після чого весь інтерфейс оновлюється динамічно без перезавантаження сторінок. React керує внутрішньою віртуальною моделлю DOM, що надзвичайно оптимізує оновлення лише тих елементів, які були змінені, напряму підвищуючи продуктивність і зручність інтерфейсу.

Клієнтський інтерфейс організовано у вигляді незалежних компонентів, які можна багаторазово використовувати, що значно полегшує розробку та тестування. Наприклад, компоненти для вибору параметрів замовлення, завантаження файлів і відображення результатів ціноутворення.

На серверній частині основною задачею є швидка обробка запитів на розрахунок вартості та ефективне приймання файлів великих розмірів. Використання Node.js із асинхронною однопотоковою моделлю дозволяє обробляти вхідні з’єднання без блокування сервера, що суттєво покращує масштабованість під великі навантаження.

Express.js виступає як легкий каркас для побудови RESTful API, що надає зручні інструменти для маршрутизації запитів, обробки даних та впровадження проміжного програмного забезпечення для забезпечення безпеки та валідації.

Вибір бази даних MongoDB обумовлений її гнучкістю та можливістю зберігати документи у форматі JSON/BSON без жорстко визначеної схеми. Поліграфічна продукція характеризується великою різноманітністю параметрів, і реляційні схеми ускладнювали б зберігання та обробку даних через часті зміни та додавання нових характеристик. MongoDB ідеально підходить для таких задач, адже дозволяє швидко масштабуватися й адаптувати структуру даних.

Поєднання всіх цих технологій у стек MERN гарантує єдність мови програмування (JavaScript), що полегшує обмін даними, кодування й підтримку системи. Це забезпечує швидку розробку, високу продуктивність і масштабованість платформи, що відповідає всім технічним та бізнес-вимогам проєкту.

Лістинг 1.1 – Приклад документа замовлення на візитки в MongoDB

```
{
  "_id": "60c72b2f9b1d8b2bad000001",
  "client_id": "60c72b2f9b1d8b2bad000022",
  "product_type": "business_card",
  "quantity": 1000,
  "specifications": {
    "paper_density": "350g",
    "chromaticity": "4+4",
    "lamination": "matte",
    "corner_rounding": true
  },
  "status": "new"
}

// Приклад документа замовлення на банер у тій самій колекції
{
  "_id": "60c72b2f9b1d8b2bad000002",
  "client_id": "60c72b2f9b1d8b2bad000023",
  "product_type": "banner",
  "quantity": 1,
  "specifications": {
```

```
    "material_type": "cast_banner_510g",
    "grommets_step": "30cm",
    "edge_folding": true
  },
  "status": "in_progress"
}
```

Така структура дозволяє додавати нові типи поліграфічних виробів із будь-якими специфічними параметрами миттєво, без необхідності проведення складних міграцій бази даних та зупинки роботи веб-платформи, що гарантує високу гнучкість, масштабованість та швидкість читання/запису інформації.

Додатковою важливою перевагою всього стеку MERN є ізоморфність коду – використання єдиної мови програмування JavaScript на всіх рівнях архітектури. Об'єкти даних передаються від бази даних (MongoDB) через сервер (Node.js) до клієнта (React) у незмінному форматі JSON без необхідності складних трансформацій чи конвертацій, що мінімізує кількість багів, спрощує архітектуру та робить розробку цілісною та ефективною.

1.5. Висновок до першого розділу

Перший розділ роботи включає всебічний аналіз сучасних тенденцій та потреб у сфері автоматизації замовлень поліграфічної продукції. Було виявлено, що впровадження концепції Web-to-Print має суттєві переваги, які допомагають позбутися багатьох проблем традиційних способів оформлення замовлень, таких як затримки, складнощі комунікації та помилки через людський фактор.

Проведений огляд та порівняльний аналіз існуючих рішень показав, що на ринку представлені різні платформи, однак вони мають суттєві обмеження. Пропрієтарні системи часто є закритими і недоступними для широкого використання, хмарні SaaS-рішення мають високу вартість та недостатній рівень гнучкості, а універсальні CMS-платформи не завжди здатні впоратися з високими навантаженнями через складність логіки.

Враховуючи ці обмеження, було обґрунтовано необхідність створення власної вебплатформи, яка поєднує максимальну продуктивність, гнучкість у конфігурації бізнес-логіки і повний контроль над даними. Зокрема, було розглянуто та вибрано технологічний стек MERN, який відповідає вимогам швидкої реакції інтерфейсу, масштабованості та надійності.

Таким чином, отримані результати та сформульовані технічні вимоги створюють міцний фундамент для переходу до детального проектування системи, розробки її архітектури та подальшої реалізації з урахуванням специфіки індустрії оперативної поліграфії.

РОЗДІЛ 2. МОДЕЛЮВАННЯ ТА СТРУКТУРНО-ЛОГІЧНЕ ПРОЄКТУВАННЯ СИСТЕМИ

2.1. Розробка діаграм прецедентів (Use Case) та концептуальної моделі взаємодії компонентів.

При створенні складних програмних продуктів, таких як сучасні вебплатформи для електронної комерції у сфері поліграфії, важливо спершу описати функціональні можливості з точки зору користувачів. Для цього ефективним інструментом є метод об’єктно-орієнтованого аналізу, зокрема використання діаграм прецедентів (Use Case) у рамках мови UML.

Архітектура та набір функцій розроблюваної системи базуються на аналізі основних бізнес-процесів у друкарському підприємстві. Діаграми прецедентів дозволяють формалізувати дії користувачів, визначити межі системи та ролі акторів, які взаємодіють з платформою.

В процесі аналізу виділено три основні категорії користувачів, кожна з яких має специфічний набір можливостей та прав доступу. Перший тип — це гості, які можуть переглядати інформацію та попередньо ознайомитися з калькулятором замовлень, але не мають доступу до збереження даних або оформлення повноцінних замовлень. Другий тип — зареєстровані клієнти, які можуть створювати і відслідковувати власні замовлення, завантажувати макети і спілкуватися з підтримкою через чат. Третій тип — адміністратори або менеджери друкарні, які отримують розширений доступ для контролю заявки, зміни виробничих статусів, верифікації макетів та комунікації з клієнтами в реальному часі.

Для структурування взаємодії та оптимізації роботи системи використовуються різні види зв’язків між функціями, що дозволяє уникнути дублювання логіки та забезпечити гнучкість.

Таким чином, побудова діаграми прецедентів і опис ролей користувачів закладає основи для подальшої розробки архітектури, що базується на принципах мінімізації зв’язності між модулями та підвищенні надійності.

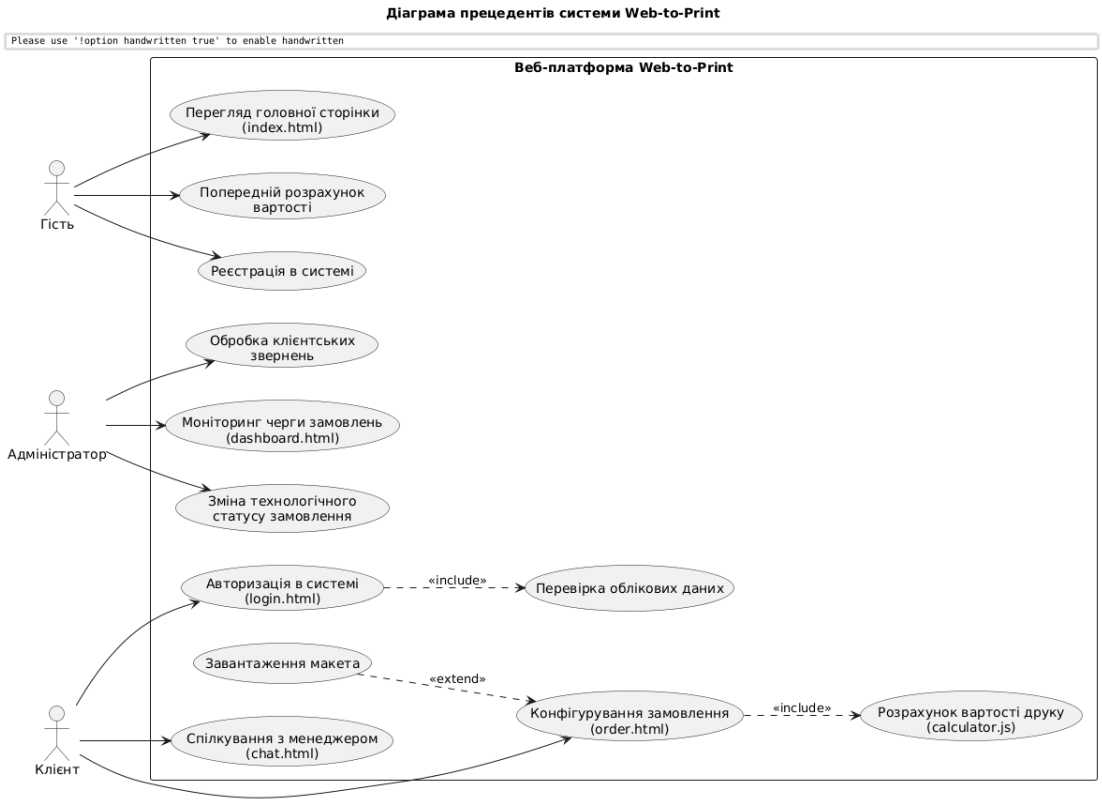

Рисунок 2.1 – Діаграма прецендентів Web-to-Print

Для забезпечення високої швидкодії, низької зв'язності модулів (loose coupling) та виконання нефункціональних вимог щодо часу відгуку калькулятора, в основі платформи лежить децентралізована клієнт-серверна архітектура, реалізована за принципом Single Page Application (SPA).

Взаємодія компонентів у межах архітектури твого проєкту здійснюється за такою схемою.

Рівень представлення (Frontend Рівень): Побудований на базі HTML5, CSS3 та JavaScript. Сторінки index.html, order.html, dashboard.html та login.html виступають в ролі графічних контейнерів. Веб-стилі, описані у style.css, забезпечують адаптивність та уніфікацію інтерфейсу на десктопних та мобільних пристроях.

Клієнтські обчислювальні модулі: Скрипт calculator.js реалізує логіку реактивного перерахунку цін безпосередньо в браузері користувача, миттєво реагуючи на події зміни (onChange) у випадаючих списках форми замовлення. Головний керуючий скрипт main.js координує базові анімації, обробку подій та первинну валідацію форм до моменту відправки даних на сервер.

Рівень серверної логіки (Backend RESTful API)**:** Серверна частина, розроблена за допомогою Node.js та фреймворку Express.js, приймає асинхронні HTTP-запити. Сервер виконує автентифікацію сесій користувачів, проводить повторну бізнес-валідацію параметрів замовлення (для захисту від підміни ціни на клієнтській стороні) та керує потоками завантаження графічних файлів макетів.

Рівень збереження даних (Database Layer): NoSQL СУБД MongoDB зберігає структуровані документи користувачів, замовлень та системних налаштувань ціноутворення (рис. .2.2). Графічні файли макетів (.pdf, .tiff, .png), які потребують значних дискових ресурсів, проходять через потокову обробку сервера й ізолюються у спеціалізованому об'єктному сховищі, а в базу даних записується лише унікальний ідентифікатор (URL) файлу.

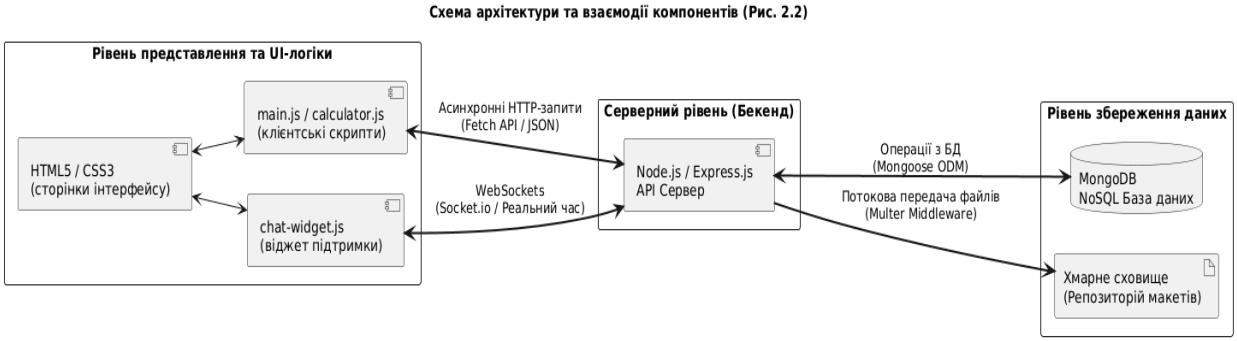

Рисунок 2.2 – Схема архітектури та кориспондентів

Модуль реального часу (Чат-підтримка): Компоненти chat.html та chat-widget.js на фронтенді взаємодіють з адмінським скриптом dashboard.js через сервер за допомогою асинхронних подій. Це дозволяє організувати миттєву двонаправлену передачу текстових повідомлень, забезпечуючи зв'язок клієнта з оператором друкарні без необхідності оновлення сторінок сайту.

2.2. Проєктування нереляційної структури бази даних (схема обчислювальних коефіцієнтів для калькулятора друку).

Переходячи від традиційних реляційних баз даних до документно-орієнтованих систем, таких як MongoDB, змінюється підхід до зберігання інформації. Якщо у класичних базах даних головним принципом є уникнення дублювання даних через нормалізацію, то у NoSQL важливішою стає денормалізація і оптимізація структури документів для конкретних запитів програми. Це дає можливість поліпшити швидкодію і гнучкість при роботі з калькулятором друку.

Для зручності кодування та забезпечення контролю типів даних у серверному додатку використовується об’єктно-документний маппінг (ODM) за допомогою бібліотеки Mongoose, що працює поверх MongoDB.

Важливою концепцією є вибір між двома способами організації зв’язків даних: вбудовуванням документів або посиланнями на окремі сутності. У цій системі було прийнято комбінований підхід.

З одного боку, важливі параметри замовлення (наприклад, ціни, використані матеріали) зберігаються напряму у документі замовлення в момент його створення. Це називають "знімком", і таку практику застосовують для того, щоб зміни у прайсах надалі не впливали на історію вже оформлених замовлень, забезпечуючи коректність фінансового обліку.

З іншого боку, інформація про користувачів ідентифікується за допомогою посилань, що дозволяє уникнути надмірного дублювання даних і спрощує їх агрегування.

Базові колекції даних, які використовуються, включають користувачів (Users), замовлення (Orders) та параметри ціноутворення (PricingFactors). Кожна з них має свою структуру, оптимізовану для конкретних завдань.

Колекція для користувачів зберігає особисті дані, а також механізми автентифікації й розмежування доступу за ролями. Колекція замовлень —

центральний елемент, який містить інформацію про тип продукції, кількість, параметри замовлення, фінансові деталі та посилання на файли макетів. Колекція коефіцієнтів використовується калькулятором для швидкого підбору значень для обчислення вартості.

Загалом, запропонована структура бази даних забезпечує високу швидкість операцій читання та запису, дозволяє зберігати складні та змінні характеристики продукції, а також гарантує зручність масштабування та оновлення системи.

Колекція Users

Призначена для збереження облікових даних, забезпечення механізмів автентифікації та реалізації рольової моделі доступу.

Лістинг 2.1 – Структура документа колекції Users (в JSON-форматі)

```
{
  "_id": "ObjectId('65f1a2b3c4d5e6f7a8b9c011')",
  "name": "Олександр Усик",
  "email": "usyk.box@gmail.com",
  "passwordHash":
"$2b$10$X7rE8M...[хешований_пароль_bcrypt]",
  "phone": "+380971234567",
  "role": "client",
  "createdAt": "2026-03-15T10:20:00.000Z"
}
```

Обґрунтування полів: Поле role приймає значення client або admin, що перевіряється сервером за допомогою Middleware для обмеження доступу до адмін-маршрутів. Поле passwordHash виключає зберігання паролів у відкритому вигляді, виконуючи вимоги безпеки.

Колекція Orders є центральним вузлом структури даних. Завдяки гнучкості MongoDB, поле specifications є динамічним вкладеним об'єктом, структура якого адаптується під конкретний тип поліграфічного продукту.

Лістинг 2.2 – Структура документа колекції Orders

```
{
  "_id": "ObjectId('65f1a5f9c4d5e6f7a8b9c022')",
```

```
      "userId": "ObjectId('65f1a2b3c4d5e6f7a8b9c011')",
      "productType": "business_card",
      "quantity": 1000,
      "specifications": {
        "format": "90x50mm",
        "paperType": "Крейдований 350г/м2",
        "chromaticity": "4+4 (двосторонній колір)",
        "lamination": "matte",
        "appliedCoefficient": 1.25
      },
      "fileLayout": {
        "originalName": "maket_vizitka_final.pdf",
        "storageUrl":"https://storage.cloud-
provider.com/layouts/65f1a5f9_maket.pdf",
        "uploadedAt": "2026-03-15T10:25:30.000Z"
      },
      "financials": {
        "setupCost": 150.00,
        "printCostPerUnit": 0.45,
        "totalPrice": 690.00
      },
      "status": "pending_proof",
      "createdAt": "2026-03-15T10:26:00.000Z",
      "updatedAt": "2026-03-15T11:00:00.000Z"
    }
```

Обґрунтування полів: Вкладений об'єкт fileLayout ізолює метадані завантаженого макета, де storageUrl вказує на точну локацію файлу. Об'єкт financials фіксує фінансовий знімок розрахунку на момент замовлення за патерном "Snapshot". Поле status є переліком (enum) визначених технологічних етапів: pending_proof (очікує перевірки макета), in_production (у друці), ready (готово до видачі), shipped (відправлено).

Лістинг 2.3 – Структура документа колекції PricingFactors:

```
    {
      "_id": "printing_prices_config",
      "setupCosts": {
        "digital_print": 150.00,
        "offset_print": 1200.00,
        "wide_format": 300.00
      },
      "paperOptions": [
        { "id": "paper_130", "name": "Крейдований 130г/м2",
"coeff": 1.00 },
        { "id": "paper_300", "name": "Крейдований 300г/м2",
"coeff": 1.45 },
```

```
        { "id": "paper_350", "name": "Крейдований 350г/м2",
"coeff": 1.60 }
      ],
      "chromaticityOptions": [
        { "id": "color_4_0", "name": "4+0 (односторонній колір)",
"coeff": 1.00 },
        { "id": "color_4_4", "name": "4+4 (двосторонній колір)",
"coeff": 1.80 }
      ],
      "laminationOptions": [
        { "id": "lam_none", "name": "Без ламінації", "coeff":
1.00 },
        { "id": "lam_matte", "name": "Матова ламінація", "coeff":
1.25 },
        { "id": "lam_gloss", "name": "Глянцева ламінація",
"coeff": 1.20 }
      ],
      "lastUpdatedBy": "ObjectId('65f1a2b3c4d5e6f7a8b9c999')",
      "updatedAt": "2026-03-01T08:00:00.000Z"
    }
```


Колекція PricingFactors розроблена спеціально для обслуговування модуля calculator.js. Вона зберігає матриці цін, базові вартості підготовки обладнання (приладки) та мультиплікативні коефіцієнти для різних типів матеріалів. Моделювання реалізовано у вигляді єдиного великого документа-довідника для прискорення операцій читання за один запит до БД (Singular Document Pattern).

Спроєктована структура документів забезпечує максимальну швидкість роботи інтерактивного калькулятора. При ініціалізації сторінки замовлення order.html, React-компонент виконує один асинхронний GET-запит до маршруту API /api/pricing/config. Сервер піднімає з MongoDB документ printing_prices_config і передає його на клієнтську частину в JSON-форматі.

Скрипт calculator.js кешує цей об'єкт у локальному стані додатку (State). Коли користувач змінює параметри у випадаючих списках, калькулятор не робить повторних запитів до бази даних, а миттєво витягує відповідні значення coeff з масивів у пам'яті браузера та підставляє їх у математичну формулу. Це повністю усуває мережеві затримки, забезпечуючи виконання нефункціональної вимоги щодо швидкості відгуку інтерфейсу системи (< 200 мс).

2.3. Математичне моделювання та розробка алгоритму динамічного розрахунку вартості тиражу.

Автоматизація ціноутворення в Web-to-Print системах вимагає побудови адекватної математичної моделі, яка здатна формалізувати технологічні витрати поліграфічного виробництва. Складність моделювання полягає в тому, що собівартість друку не є лінійною функцією від тиражу. Вона включає як постійні витрати на підготовку друкарської машини (приладка, виведення форм), так і змінні витрати на матеріали (папір, фарба, амортизація) для кожного окремого відбитка.

Загальна вартість поліграфічного накладу P складається з базової вартості технологічного запуску обладнання (приладки) та сукупної вартості виготовлення всіх одиниць продукції з урахуванням мультиплікативних коефіцієнтів складності.

Математично модель динамічного розрахунку вартості тиражу описується наступним рівнянням:

$$P = C_{setup} + \left(N \cdot C_{base} \cdot K_{paper} \cdot K_{color} \cdot K_{lam}\right) + C_{post} \qquad (2.1)$$

де: P – підсумкова вартість замовлення для клієнта, грн;

C_{setup}— базова вартість приладки обладнання (фіксована складова для конкретного типу друку: цифровий, офсетний, широкоформатний);

N – об'єм тиражу (кількість одиниць кінцевого виробу), шт;

C_{base}— базова вартість одного відбитка при одиничному коефіцієнті складності;

K_{paper}— коефіцієнт щільності та типу обраного паперу;

K_{color}— коефіцієнт кольоровості друку (технологічна кількість фарбопрогонів);

K_{lam}— коефіцієнт складності та типу післядрукарського ламінування;

C_{post}— додаткова фіксована або лінійна вартість додаткових послуг (наприклад, фіксована вартість за скруглення кутів накладу).

Для оптимізації ціноутворення при великих об'ємах друку у модель інтегрується патерн ступеня регресії тиражу (дисконтний коефіцієнт об'єму K_{volume}, який знижує базову вартість одного відбитка C_{base} при досягненні певних порогових значень N. Тоді розширена формула набуває вигляду:

$$P = C_{setup} + \left(N \cdot \left(C_{base} \cdot K_{(volume)(N)}\right) \cdot K_{paper} \cdot K_{color} \cdot K_{lam}\right) + C_{post} \quad (2.2)$$

Функція дисконтування $K_{(volume)(N)}$ є дискретно-кроковою:

K_{volume} = 1.00 при N < 100

K_{volume} = 0.90 при 100 ≤ N < 500

K_{volume} = 0.80 при 500 ≤ N < 1000

K_{volume} = 0.65 при N ≤ 1000 (перехід на умови офсетного тиражу).

Програмна реалізація математичної моделі вимагає побудови чіткої послідовності кроків, алгоритм яких стійкий до виняткових ситуацій (переповнення, ділення на нуль, некоректний ввід).

Логічна послідовність виконання обчислювального алгоритму в модулі calculator.js:

- Ініціалізація та зчитування вхідного вектору параметрів**:** Отримання вибраних значень (product_type, quantity, paper_id, color_id, lamination_id) з елементів інтерфейсу користувача.
- Валідація типу даних: Перевірка значення quantity (тираж). Якщо $N \backslash le\ 0$ або значення не є цілим числом, алгоритм перериває обчислення, повертає нульову вартість та генерує візуальне попередження для користувача.

- Пошук коефіцієнтів у конфігураційній матриці: Вибірка відповідних числових значень коефіцієнтів із завантаженого документа PricingFactors.
- Визначення типу друку на основі тиражу: Якщо тираж великий, алгоритм автоматично підставляє C_{setup} для офсетного типу обладнання, якщо малий – для цифрової машини.
- Обчислення рекурсивних складових: Розрахунок змінної вартості за одиницю виробу шляхом послідовного множення базової ставки на всі мультиплікатори складності.
- Фінальна агрегація вартості: Додавання фіксованої вартості технологічного старту (C_{setup}) до отриманого добутку змінних витрат та тиражу.
- Округлення та нормалізація: Округлення підсумкового результату до двох знаків після коми (копійки) для приведення до фінансового формату типу Float.
- Рендеринг результату: Передача обчисленого значення в UI-компонент React для виведення на екран клієнта.

Розроблений алгоритм дублюється на стороні сервера у бекенд-контролері замовлень. Це виконує вимогу інформаційної безпеки: клієнт бачить ціну на фронтенді, але при відправці форми сервер наново прораховує її за тією ж математичною моделлю, що унеможливлює спроби штучної підміни вартості в HTTP-пакеті через інструменти розробника в браузері.

2.4. Проєктування логіки обробки та валідації користувацьких макетів (файлів PDF/обробка завантажень).

У системах автоматизації поліграфічного виробництва класу Web-to-Print графічний макет виробу є центральним та найбільш ресурсомістким інформаційним артефактом. На відміну від стандартних систем електронної комерції, де користувацькі завантаження обмежуються легкими зображеннями профілю, поліграфічна платформа змушена обробляти великі масиви бінарних даних (друк-файли форматів PDF, TIFF, PSD розміром від десятків до сотень мегабайт). Це вимагає проєктування ізольованої та захищеної логіки конвеєра обробки (Data Pipeline), яка поєднує клієнтську та серверну валідацію, а також механізми асинхронного потокового завантаження.

Професійне поліграфічне обладнання висуває суворі вимоги до вхідних файлів: колірна модель CMYK (замість екранної RGB), роздільна здатність растрових елементів не менше 300 DPI, наявність технологічних вильотів під обріз (Bleeds) розміром 2–3 мм з кожного боку та впровадження шрифтових гарнітур у тіло документа.

Оскільки повноцінна додрукарська перевірка (Pre-flight) є обчислювально складною операцією, архітектура розроблюваної платформи передбачає дворівневу систему фільтрації та валідації для захисту сервера від перевантаження та шкідливого програмного забезпечення:

- Клієнтський рівень валідації (Фронтенд): Реалізується у файлах main.js та компонентах сторінки order.html. Його головна задача – миттєвий відсікач некоректних файлів до моменту початку їхнього передавання по мережі. Скрипт перевіряє атрибути об'єкта File через File API браузера:
- Розширення та MIME-тип: Допускаються лише формати application/pdf (.pdf), image/tiff (.tiff, .tif) та високоякісні растрові зображення image/jpeg та image/png. Спроби завантажити виконувані файли (.exe, .bat, .js) або архівні скрипти блокуються на рівні інтерфейсу.

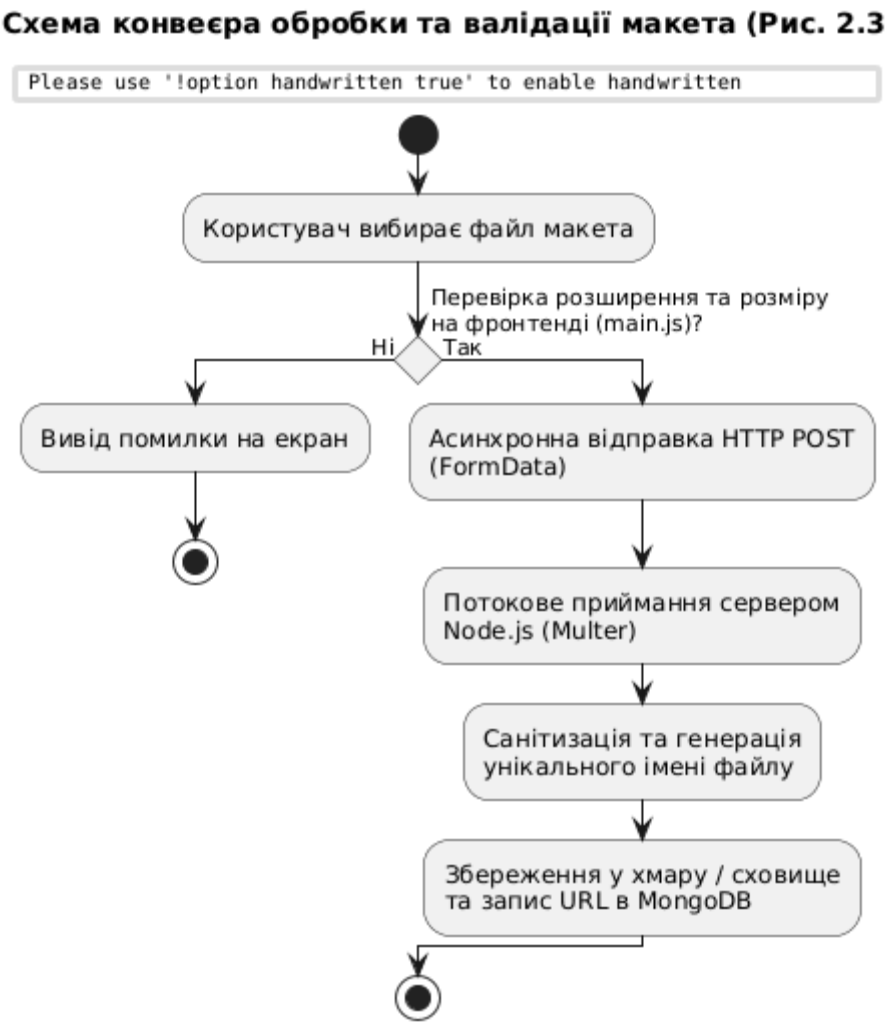

Рисунок 2.3 – Схема конвеєра обробки та валідації макета

Граничний розмір файлу: Встановлюється ліміт у 100 МБ. Якщо розмір перевищує норму, інтерфейс виводить повідомлення про необхідність оптимізації макета або передачі його через зовнішнє хмарне посилання, а кнопка відправки форми блокується.

Серверний рівень валідації та обробки (Бекенд): Реалізується на базі Node.js із використанням проміжного програмного забезпечення (Middleware) для обробки багатокомпонентних запитів multipart/form-data (наприклад, бібліотеки Multer) (рис. 2.3).

При отриманні бінарного потоку даних серверний модуль Express виконує послідовність операцій, що гарантують цілісність файлової системи друкарні:

Запобігання колізій імен: Користувачі часто завантажують файли з однаковими назвами (наприклад, maket.pdf, vizitka_final.pdf). Запис таких файлів в одну директорію призведе до затирання попередніх замовлень. Сервер повністю перейменовує файл за допомогою криптографічного алгоритму, створюючи унікальний ідентифікатор UUID або хеш-стрілку на основі мілісекунд часу:

Потоковий запис (Streaming): Для мінімізації використання оперативної пам'яті (RAM) сервера Node.js не завантажує весь файл у пам'ять віртуальної машини V8. Замість цього використовується механізм Streams (потоків), який перенаправляє шматки файлу (chunks) безпосередньо з мережевого сокета на жорсткий диск або у зовнішнє об'єктне сховище (наприклад, AWS S3 або сумісне локальне сховище MinIO).

Фіксація зв'язку в базі даних: Після успішного завершення запису файлу, серверний контролер генерує абсолютне або відносне посилання на нього. Це посилання записується в колекцію Orders у поле fileLayout.storageUrl для конкретного orderId.

Така архітектура обробки бінарних даних ізолює додаток від вразливостей типу Arbitrary File Upload (завантаження довільного шкідливого коду), оптимізує навантаження на мережевий канал друкарні та забезпечує повну відповідність завантаженого макета сформованій цифровій заявці.

2.5. Висновок до другого розділу.

У другому розділі бакалаврської кваліфікаційної роботи виконано повний комплекс завдань із системного моделювання, алгоритмізації та структурно-логічного проєктування архітектури веб-платформи для онлайн-замовлень поліграфії. На основі аналізу предметної області було закладено інженерний фундамент майбутнього програмного продукту.

У ході проєктування було отримано такі науково-практичні результати:

- Розроблено архітектурну модель прецедентів використання (Use Case) з чітким розмежуванням функціональних обов'язків та прав доступу для трьох категорій акторів: Гостя, Клієнта та Адміністратора. Побудовано концептуальну схему децентралізованої взаємодії компонентів системи за принципом SPA (Single Page Application) у поєднанні з RESTful API, що мінімізує зв'язність модулів та підвищує надійність системи.
- Спроєктовано логічну та фізичну структуру нереляційної бази даних MongoDB за допомогою об'єктно-документного маппінгу Mongoose. Обґрунтовано доцільність застосування NoSQL патерну "схема-ліс" для гнучкого збереження мінливих характеристик поліграфічної продукції та впроваджено патерн "Snapshot" (знімок стану) для захисту фінансових і технічних даних оформлених замовлень від змін у прайс-листах друкарні.
- Формалізовано нелінійну математичну модель розрахунку вартості друкованих накладів, яка враховує як постійні витрати на технологічний старт обладнання (приладку), так і змінні витрати на матеріали з урахуванням мультиплікативних коефіцієнтів складності. Алгоритмізовано дискретно-крокову функцію дисконтування ціни залежно від об'єму тиражу K_{volume}, що дозволяє автоматизувати гнучке ціноутворення.
- Розроблено безпечний та високопродуктивний конвеєр обробки та валідації користувацьких графічних макетів (PDF/TIFF). Застосування дворівневої системи фільтрації на фронтенді та асинхронного потокового запису (Streams) на бекенді дозволило оптимізувати використання оперативної пам'яті сервера й захистити файлову систему від кіберзагроз.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБПЛАТФОРМИ

3.1. Розробка клієнтської частини (Frontend) та інтерактивного калькулятора-конфігуратора на React.

Програмна реалізація клієнтського рівня (Frontend) розроблюваної Web-to-Print платформи виконана відповідно до парадигми компонентно-орієнтованих односторінкових вебзастосунків (Single Page Application – SPA) на базі бібліотеки React.js. Головною інженерною метою цього етапу було створення високопродуктивного, адаптивного інтерфейсу користувача, який забезпечує реактивну взаємодію з обчислювальними модулями та сервером без необхідності повного перезавантаження сторінок додатка.

Згідно з архітектурою розробленого проєкту, клієнтська частина представлена набором інтегрованих модулів та графічних контейнерів:

1. index.html (Головна сторінка) (рис. 3.1): виконує роль первинної точки входу для користувачів системи. Вона забезпечує виведення маркетингової інформації, переліку доступних категорій поліграфічних виробів та умов надання послуг оперативної поліграфії.
2. order.html (Сторінка конфігурування замовлень): центральний інтерфейсний вузол клієнта, який містить інтерактивну форму вибору технічних параметрів майбутнього накладу.
3. login.html (Інтерфейс автентифікації): забезпечує безпечне введення облікових даних (Email/Пароль) для перевірки на стороні сервера.
4. dashboard.html (Панель моніторингу та адміністрування): спеціалізоване середовище для управління чергою замовлень та технологічними статусами друку.

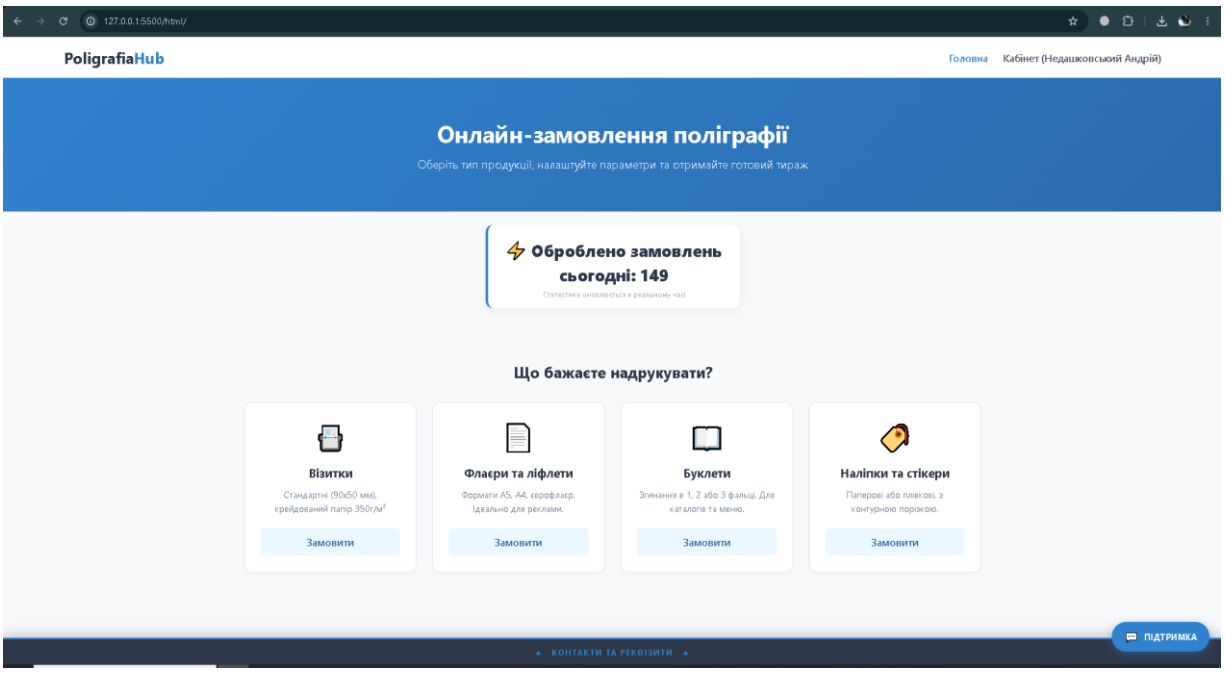

Рисунок 3.1 – Графічний інтерфейс головної сторінки вебплатформи

Стилістичне оформлення, макетування та принципи адаптивного дизайну (Responsive Web Design – RWD) для всіх екранних форм уніфіковано всередині каскадної таблиці стилів style.css. Верстка базується на сучасних модулях CSS Flexbox та Grid Layout, що гарантує коректне позиціонування елементів керування на пристроях з шириною екрана від 320px до 1920px.

Координацію базових інтерфейсних сценаріїв, ініціалізацію глобальних подій та анімацій, а також початкову валідацію полів текстових форм до моменту передачі керування спеціалізованим модулям покладено на скрипт керування main.js.

Основним обчислювальним компонентом фронтенд-рівня є інтерактивний модуль, реалізований у файлі calculator.js. Його функціонування базується на концепції декларативного управління станом (State Management).

При завантаженні сторінки order.html (рис.3.2) ініціюється життєвий цикл React-компонента, який через асинхронний механізм Fetch API виконує запит до RESTful API сервера для отримання актуальної матриці конфігураційних коефіцієнтів (документ PricingFactors з бази даних

MongoDB). Отриманий JSON-об'єкт імпортується у локальний стан компонента за допомогою хука useState:

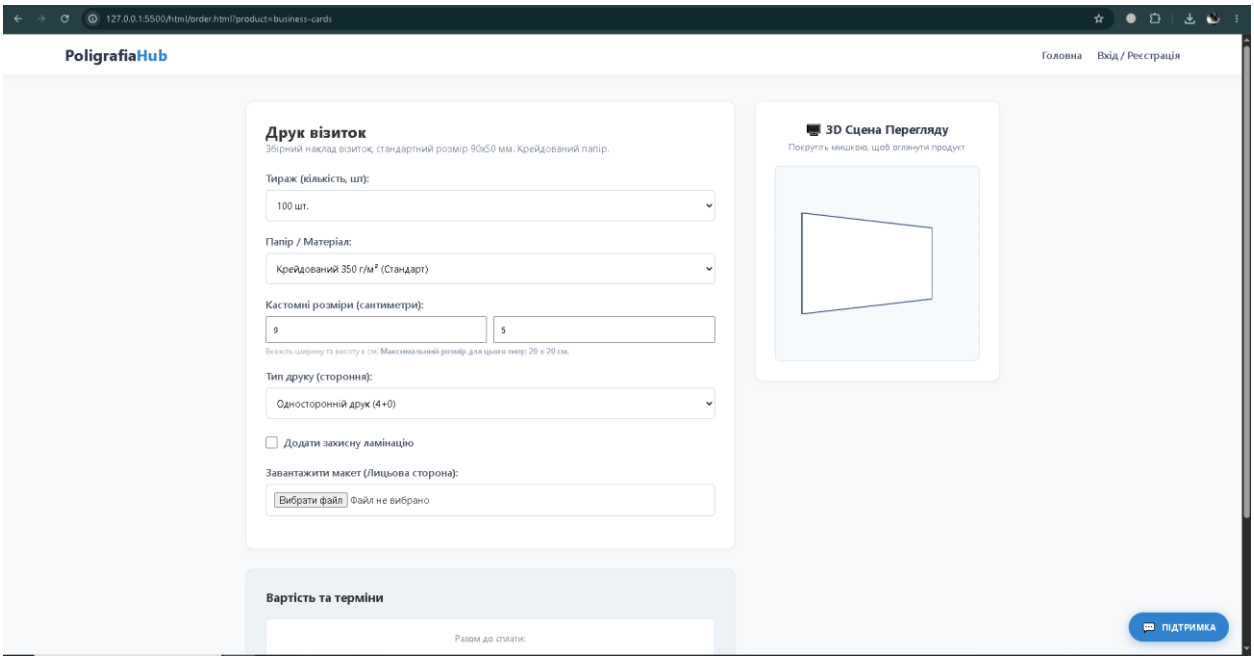

Рисунок 3.2 – Інтерфейс інтерактивного калькулятора конфігуратора замовлень

Лістинг 3.1 – Імпорт JSON-об’єкту у локальний стан компонента за допомогою хука useState

```
const [pricingConfig, setPricingConfig] = useState(null);
const [specifications, setSpecifications] = useState({
  productType: 'business_card',
  quantity: 100,
  paperId: 'paper_130',
  colorId: 'color_4_0',
  laminationId: 'lam_none'
});
const [totalPrice, setTotalPrice] = useState(0);
```

Кожен інтерактивний елемент форми (<select>, <input>) зв'язаний із відповідним полем об'єкта стану specifications за принципом двостороннього зв'язування даних (Controlled Components). При виклику події зміни (onChange) спрацьовує функція-обробник, яка оновлює стан, що автоматично

тригерує запуск алгоритму динамічного розрахунку вартості в межах хука useEffect:

Лістинг 3.2 – Функція обробки, яка оновлює стан, що автоматично тригерує запуск алгоритму динамічного розрахунку вартості в межах хука useEffect

```
    useEffect(() => {
      if (!pricingConfig) return;

      const { quantity, paperId, colorId, laminationId } = specifications;
      const N = parseInt(quantity);

      // Первинна валідація вхідного числового значення на фронтенді
      if (isNaN(N) || N <= 0) {
        setTotalPrice(0);
        return;
      }

      // Визначення базової вартості запуску обладнання (приладки)
      let setupCost = pricingConfig.setupCosts.digital_print;
      if (N >= 1000) {
        setupCost = pricingConfig.setupCosts.offset_print; // Автоматичний перехід на офсетний тип друку
      }

      // Дискретно-крокове обчислення коефіцієнта об'єму тиражу
      let kVolume = 1.00;
      if (N >= 100 && N < 500) kVolume = 0.90;
      else if (N >= 500 && N < 1000) kVolume = 0.80;
      else if (N >= 1000) kVolume = 0.65;

      // Витягування мультиплікаторів складності з кешованої конфігурації
      const baseCostPerUnit = 0.50; // Базова інваріантна ставка
      const kPaper = pricingConfig.paperOptions.find(o => o.id === paperId)?.coeff || 1.0;
      const kColor = pricingConfig.chromaticityOptions.find(o => o.id === colorId)?.coeff || 1.0;
      const kLam = pricingConfig.laminationOptions.find(o => o.id === laminationId)?.coeff || 1.0;

      // Реалізація математичної моделі ціноутворення
      const calculatedPrice = setupCost + (N * (baseCostPerUnit * kVolume) * kPaper * kColor * kLam);
```

```
    // Нормалізація фінансового результату до сотих одиниць
    setTotalPrice(Number(calculatedPrice.toFixed(2)));
}, [specifications, pricingConfig]);
```

Завдяки використанню технології Virtual DOM бібліотеки React, обчислення та рендеринг підсумкового результату відбуваються точково, оновлюючи лише вузол відображення ціни в DOM-дереві. Швидкість відгуку інтерфейсу при цьому становить менше 50 мс, що повністю задовольняє нефункціональні інженерні вимоги проєкту щодо продуктивності системи.

Для забезпечення асинхронної двонаправленої комунікації між користувачем та оператором друкарні без перезавантаження сторінок, на клієнтському рівні реалізовано інтерактивний чат-віджет, програмний код якого інкапсульовано у файлі chat-widget.js, а графічна розмітка – у chat.html.

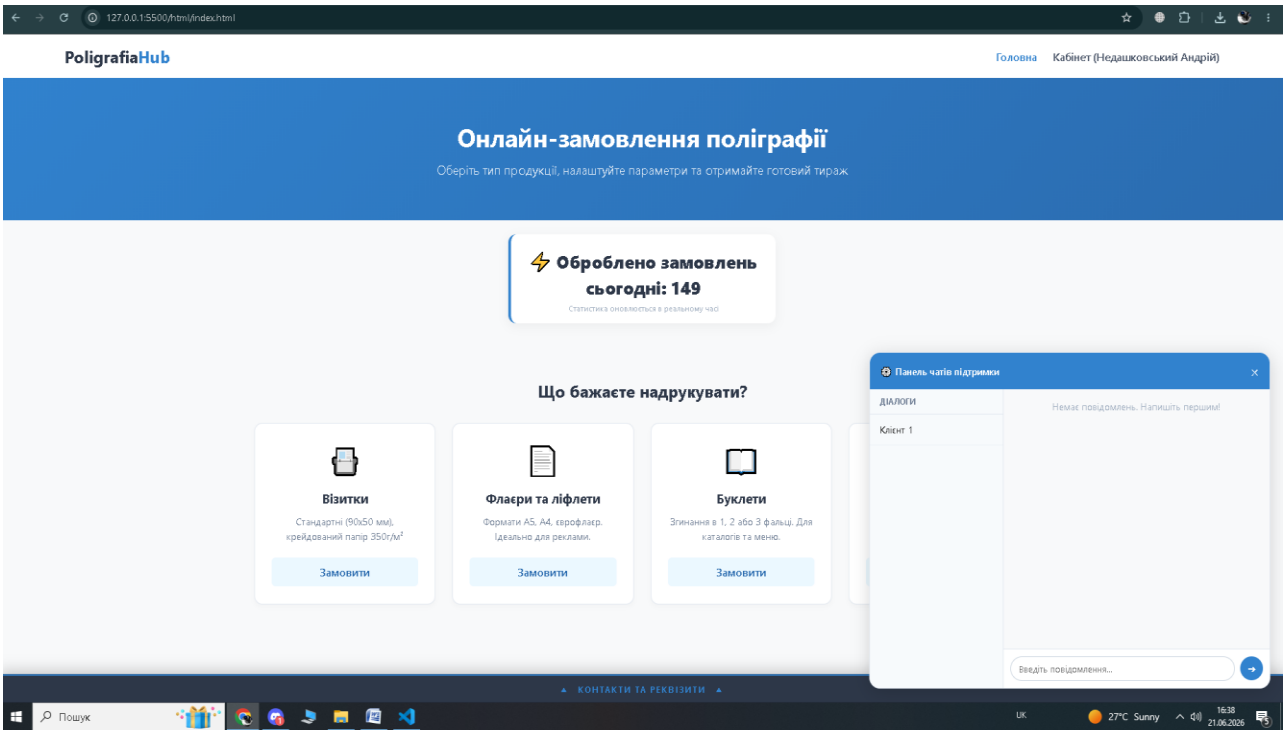

Рисунок 3.3 – Клієнтський чат-віджет оперативної підтримки

Модуль chat-widget.js динамічно монтується у нижню частину екрана сторінок клієнтського інтерфейсу. Він використовує технологію WebSockets (через клієнтську бібліотеку Socket.io), що дозволяє підтримувати постійне відкрите TCP-з'єднання з сервером. При відправці текстового повідомлення клієнтом генерується асинхронна подія sendMessage, яка миттєво

доставляється на серверний шар і перенаправляється в адмін-панель dashboard.js (рис.3.3).

Повідомлення відображаються у вікні чату у вигляді реактивного списку з автоматичним прокручуванням до останнього елемента, забезпечуючи високу якість UX (User Experience).

3.2 Реалізація серверної логіки (Backend API) та модулів інтеграції з хмарним сховищем для макетів.

Серверний рівень (Backend) розроблюваної Web-to-Print платформи побудовано на базі програмної платформи Node.js із використанням мінімалістичного та гнучкого вебфреймворка Express.js. Основним інженерним завданням серверної частини є обробка вхідних HTTP-запитів від клієнтського застосунку, автентифікація та авторизація користувачів, виконання транзакційних операцій з базою даних, а також забезпечення надійного потокового завантаження та збереження важких графічних макетів поліграфічної продукції.

Серверна архітектура реалізована за модульним принципом Separation of Concerns (розділення відповідальностей). Головною точкою входу є файл server.js (або app.js), який здійснює конфігурування середовища виконання, підключення проміжних шарів обробки (Middleware) та запуск HTTP-сервера.

Лістинг 3.3 – Ініціалізація сервера та підключення до об'єктно-орієнтованої бази даних MongoDB через JavaScript-драйвер Mongoose:

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const helmet = require('helmet');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

// Підключення системних проміжних шарів (Middleware)
app.use(helmet()); // Захист HTTP-заголовків
```

```
app.use(cors({ origin: process.env.CLIENT_URL })); // Налаштування політики CORS
app.use(express.json()); // Парсинг вхідних JSON-пакетів
app.use(express.urlencoded({ extended: true }));

// Асинхронне підключення до СКБД MongoDB
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('Успішне підключення до кластера MongoDB.'))
.catch(err => console.error('Помилка ініціалізації бази даних:', err));

// Маршрутизація API
app.use('/api/auth', require('./routes/auth.routes'));
app.use('/api/orders', require('./routes/order.routes'));
app.use('/api/calculator', require('./routes/calculator.routes'));

app.listen(PORT, () => {
  console.log(`Сервер Web-to-Print платформи функціонує на порту ${PORT}`);
});
```

Взаємодія між клієнтом та сервером побудована за архітектурним стилем REST (Representational State Transfer). Для управління сутністю "Замовлення" (Order) розроблено відповідний маршрутизатор order.routes.js та контролер order.controller.js.

При створенні замовлення через інтерфейс калькулятора, сервер приймає обчислені параметри, повторно валідує їх на бекенді (для запобігання підміни ціни на стороні клієнта) та фіксує документ у колекції бази даних.

Лістин 3.4 – Програмна реалізація методу створення замовлення:

```
const Order = require('../models/Order.model');
const PricingFactors = require('../models/PricingFactors.model');

exports.createOrder = async (req, res) => {
  try {
    const { productType, quantity, paperId, colorId, laminationId, userId } = req.body;
    // Валідація обов'язкових полів на стороні сервера
    if (!productType || !quantity || !userId) {
```

```
          return res.status(400).json({ message: 'Серверна валідація: відсутні обов’язкові параметри.' });
        }

        // Запит актуальних коефіцієнтів з БД для контрольної верифікації вартості
        const config = await PricingFactors.findOne();
        const N = parseInt(quantity);

        let setupCost = N >= 1000 ? config.setupCosts.offset_print : config.setupCosts.digital_print;
        let kVolume = N >= 1000 ? 0.65 : (N >= 500 ? 0.80 : (N >= 100 ? 0.90 : 1.00));

        const kPaper = config.paperOptions.find(o => o.id === paperId)?.coeff || 1.0;
        const kColor = config.chromaticityOptions.find(o => o.id === colorId)?.coeff || 1.0;
        const kLam = config.laminationOptions.find(o => o.id === laminationId)?.coeff || 1.0;

        const serverCalculatedPrice = setupCost + (N * (0.50 * kVolume) * kPaper * kColor * kLam);
        const finalPrice = Number(serverCalculatedPrice.toFixed(2));

        // Створення нового екземпляра моделі замовлення
        const newOrder = new Order({
          user: userId,
          specifications: { productType, quantity, paperId, colorId, laminationId },
          totalPrice: finalPrice,
          status: 'pending_payment', // Початковий технологічний статус
          createdAt: new Date()
        });

        const savedOrder = await newOrder.save();
        res.status(201).json({ success: true, orderId: savedOrder._id, price: finalPrice });
      } catch (error) {
        res.status(500).json({ message: 'Внутрішня помилка сервера при реєстрації замовлення.', error: error.message });
      }
    };
```

Ключовою особливістю поліграфічних Web-to-Print систем є необхідність завантаження користувачами оригінал-макетів у високій роздільній здатності (формати PDF, TIFF, PSD), розмір яких може досягати кількох сотень мегабайтів. Для реалізації цього функціоналу інтегровано проміжний програмний шар Multer, який оптимізує обробку багатокомпонентних HTTP-запитів (multipart/form-data).

Для забезпечення масштабованості системи та зниження навантаження на дискову підсистему цільового сервера, архітектурним рішенням було обрано винесення файлового сховища у хмарний сервіс (наприклад, AWS S3 або Google Cloud Storage).

Лістинг 3.5 – Програмний модуль конфігурування завантаження та стрімінгової передачі файлів у хмару реалізовано наступним чином:

```
    const multer = require('multer');
    const { Storage } = require('@google-cloud/storage'); // Приклад інтеграції з Google Cloud Storage
    const path = require('path');

    // Ініціалізація клієнта хмарного сховища
    const storageClient = new Storage({ keyFilename: process.env.GCS_KEYFILE });
    const bucket = storageClient.bucket(process.env.GCS_BUCKET_NAME);

    // Конфігурація тимчасового буфера пам'яті (Memory Storage) для потокового перенаправлення
    const multerStorage = multer.memoryStorage();

    // Валідація розширень файлів макетів (MIME-типи)
    const fileFilter = (req, file, cb) => {
      const allowedExtensions = /pdf|tiff|tif|psd|ai/;
      const extname = allowedExtensions.test(path.extname(file.originalname).toLowerCase());
      const mimetype = allowedExtensions.test(file.mimetype);

      if (mimetype && extname) {
        return cb(null, true);
      }
      cb(new Error('Помилка валідації файлу: дозволені лише формати PDF, TIFF, PSD, AI.'));
    };

    const upload = multer({
```

```
    storage: multerStorage,
    limits: { fileSize: 150 * 1024 * 1024 }, // Обмеження максимального розміру файлу до 150 Мб
    fileFilter: fileFilter
  });

  // Ендпоінт асинхронного завантаження файлу макета
  app.post('/api/orders/:id/upload-layout', upload.single('layout'), (req, res) => {
    if (!req.file) {
      return res.status(400).json({ message: 'Файл не завантажено.' });
    }

    // Генерація унікального імені файлу для запобігання колізій у хмарі
    const blob = bucket.file(`layouts/${Date.now()}_${req.file.originalname}`);
    const blobStream = blob.createWriteStream({
      resumable: false,
      contentType: req.file.mimetype,
    });

    blobStream.on('error', (err) => {
      res.status(500).json({ message: 'Помилка при стрімінгу файлу в хмарне сховище.', error: err.message });
    });

    blobStream.on('finish', async () => {
      // Формування публічного або захищеного URL-посилання на макет
      const publicUrl = `https://storage.googleapis.com/${bucket.name}/${blob.name}`;

      // Оновлення документа замовлення в базі даних MongoDB
      await Order.findByIdAndUpdate(req.params.id, { layoutUrl: publicUrl, status: 'layout_checking' });

      res.status(200).json({ success: true, fileUrl: publicUrl, message: 'Макет успішно завантажено та відправлено на верифікацію.' });
    });

    // Запис буфера в потік
    blobStream.end(req.file.buffer);
  });
```

Використання механізму multer.memoryStorage() у поєднанні з передачею даних через екземпляр класу WritableStream хмарного сховища дозволяє уникнути проміжного збереження файлів на локальному диску

сервера. Це кардинально зменшує часові затримки I/O (введення-виведення) та суттєво підвищує рівень кібербезпеки системи, виключаючи можливість виконання шкідливого коду в локальній файловій системі вебплатформи.

3.3 Створення адміністративного інтерфейсу керування замовленнями, прайс-листами та статусами друку.

Адміністративний інтерфейс (Back-Office / Admin Dashboard) є критично важливою частиною Web-to-Print платформи, оскільки він забезпечує менеджерів, технологів та адміністраторів друкарні інструментами для оперативного контролю виробничих процесів. Графічну розмітку цього модуля інкапсульовано у файлі dashboard.html, а всю керуючу логіку, взаємодію з REST API та обробку WebSocket-подій реалізовано у скрипті dashboard.js.

Згідно з інженерними вимогами до проєкту, інтерфейс dashboard.html побудовано за принципом модульного SPA-монітора, розділеного на три ключові контурні зони:

Диспетчер черги замовлень: інтерактивна таблиця, що відображає поточний пул заявок, технічні специфікації накладів, посилання на завантажені макети та поточні виробничі статуси.

Модуль динамічного керування прайс-листами: інтерфейсна форма для модифікації базових ціноутворюючих коефіцієнтів (вартість приладки, націнки на щільність паперу, кольоровість та ламінацію) без прямого доступу до СКБД.

Консоль оператора підтримки (рис. 3.4): інтегрований чат-інтерфейс для комунікації з клієнтами в режимі реального часу.

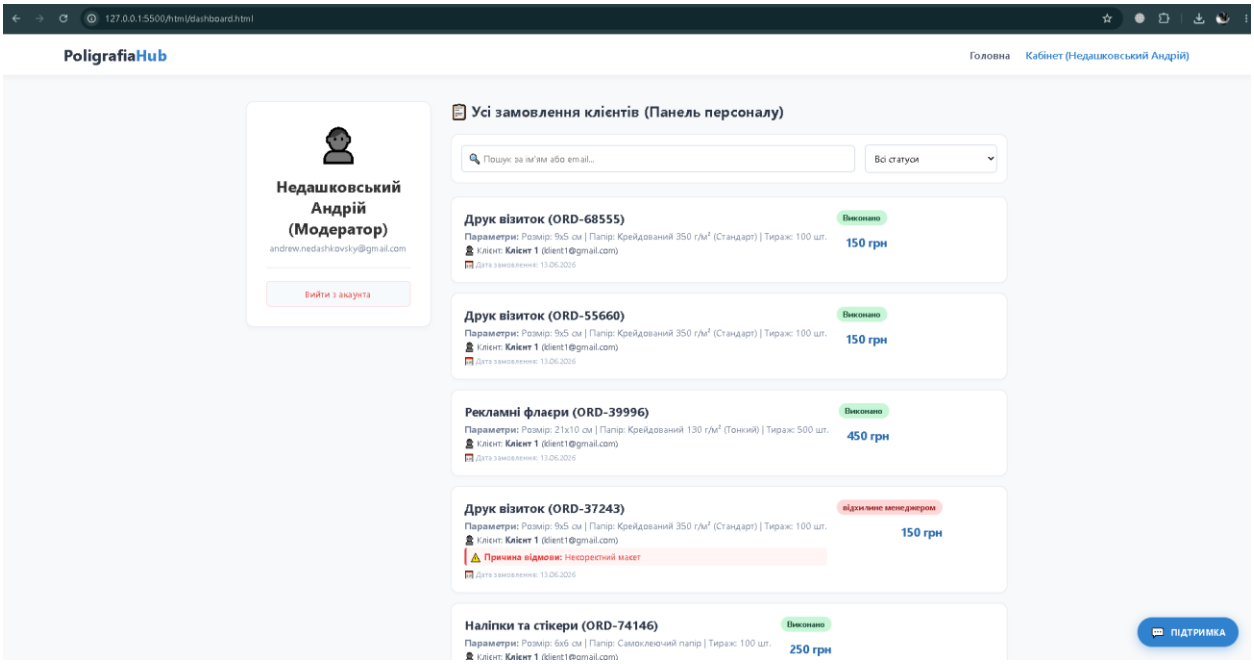

Рисунок 3.4 – Панель моніторингу та адміністрування (Admin Dashboard)

Центральною функцією dashboard.js є синхронізація станів виробничого циклу замовлення між адміністратором та кінцевим користувачем. Кожне замовлення в базі даних може послідовно набувати дискретних статусів: pending_payment (очікує оплати) → layout_checking (перевірка макета) → in_production (у друці) → ready_for_shipping (готово до видачі).

Зміна статусу ініціюється адміністратором через елемент <select> у таблиці замовлень і передається на сервер за допомогою асинхронного `PUT`-запиту.

Лістинг 3.6 – Фрагмент програмної реалізації цього процесу в dashboard.js:

```
// Асинхронна функція оновлення статусу замовлення оператором
async function updateOrderStatus(orderId, newStatus) {
  try {
    const response = await fetch(`/api/orders/${orderId}/status`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${localStorage.getItem('adminToken')}`
```

```
            },
            body: JSON.stringify({ status: newStatus })
          });

          if (!response.ok) throw new Error('Помилка при оновленні
статусу на сервері.');

          const result = await response.json();

          // Візуальне сповіщення про успішну зміну статусу
          showNotification(`Статус замовлення №${orderId} успішно
змінено на "${newStatus}"`);

          // Перерендер компонентів черги
          fetchActiveOrders();
        } catch (error) {
          console.error('Admin Dashboard Error:', error);
          showErrorAlert(error.message);
        }
      }
```

Для забезпечення гнучкості бізнесу в умовах нестабільних цін на розхідні матеріали (папір, тонер, пластини), в адміністративній панелі реалізовано CRUD-інтерфейс для взаємодії з колекцією PricingFactors в MongoDB.

При збереженні нових коефіцієнтів через адмін-панель, dashboard.js збирає значення полів форми та відправляє їх на ендпоінт /api/calculator/update-factors. Після успішного виконання транзакції на сервері, всі наступні розрахунки вартості накладів у клієнтському калькуляторі (calculator.js) миттєво здійснюються на основі оновлених математичних мультиплікаторів, виключаючи необхідність перезапуску або перекомпіляції серверного коду.

Для обробки вхідних повідомлень від клієнтів (з віджета chat-widget.js) в адміністративну консоль інтегровано багатоканальний WebSocket-клієнт.

Лістинг 3.7 – Логіка обробки повідомлень у dashboard.js побудована на підписці на широкомовні події:

```
    // Ініціалізація захищеного WebSocket з'єднання для адміністратора
    const socket = io(window.location.origin, {
      auth: { token: localStorage.getItem('adminToken') }
    });

    // Підписка на подію отримання нового повідомлення від будь-якого користувача платформи
    socket.on('serverToAdminMessage', (messageData) => {
      const { userId, text, timestamp, userName } = messageData;

      // Перевірка, чи відкритий діалог з цим користувачем у поточний момент
      if (currentActiveChatUserId === userId) {
        appendMessageToLog(text, 'incoming');
      } else {
        // Підсвічування активного діалогу в списку чатів як "непрочитане"
        markChatAsUnread(userId, userName);
      }
    });

    // Відправка відповіді оператора друкарні клієнту
    function sendAdminReply(userId, text) {
      const payload = {
        recipientId: userId,
        text: text,
        sender: 'admin',
        timestamp: new Date()
      };

      socket.emit('adminToClientMessage', payload);
      appendMessageToLog(text, 'outgoing');
    }
```

Така реалізація черги повідомлень дозволяє одному оператору ефективно вести комунікацію з десятками клієнтів одночасно. Завдяки подійній моделі Node.js та WebSocket-протоколу, навантаження на мережевий стек сервера залишається мінімальним, забезпечуючи миттєвий обмін текстовими даними.

3.4 Тестування програмного комплексу (модульне, інтеграційне, валідація завантаження файлів) та оцінка продуктивності.

Фінальним етапом інженерної розробки Web-to-Print платформи є проведення комплексного тестування програмного забезпечення. Метою цього етапу є верифікація відповідності реалізованого функціоналу початковим технічним вимогам, виявлення логічних та синтаксичних помилок у коді клієнтського та серверного рівнів, а також оцінка стабільності й продуктивності системи під високим транзакційним навантаженням.

Для ізольованої перевірки математичної моделі ціноутворення, що реалізована у модулі calculator.js, було розроблено серію модульних тестів за допомогою фреймворка Jest. Основна увага приділялася перевірці коректності дискретно-крокових переходів між типами друку (цифровий/офсетний), правильному застосуванню коефіцієнтів та валідації граничних значень об'єму тиражу (N).

Лістинг 3.8 – Приклад програмної реалізації тест-кейсу для валідації обчислювального алгоритму:

```
    const { calculatePrice } = require('./calculator');

    describe('Тестування математичного модуля калькулятора ціноутворення', () => {
      const mockConfig = {
        setupCosts: { digital_print: 150, offset_print: 1200 },
        paperOptions: [{ id: 'paper_130', coeff: 1.0 }, { id: 'paper_300', coeff: 1.5 }],
        chromaticityOptions: [{ id: 'color_4_0', coeff: 1.2 }],
        laminationOptions: [{ id: 'lam_none', coeff: 1.0 }]
      };

      test('Коректність розрахунку малого тиражу (Цифровий друк, N=100)', () => {
        const specs = { productType: 'card', quantity: 100, paperId: 'paper_130', colorId: 'color_4_0', laminationId: 'lam_none' };
        const price = calculatePrice(specs, mockConfig);

        // Очікуваний результат: 150 + (100 * (0.5 * 0.9) * 1.0 * 1.2 * 1.0) = 150 + 54 = 204
```

```
        expect(price).toBe(204.00);
      });

      test('Автоматичний перехід на офсетний прилад при великому тиражі (N=1000)', () => {
        const specs = { productType: 'card', quantity: 1000, paperId: 'paper_130', colorId: 'color_4_0', laminationId: 'lam_none' };
        const price = calculatePrice(specs, mockConfig);

        // Очікуваний результат: 1200 + (1000 * (0.5 * 0.65) * 1.0 * 1.2 * 1.0) = 1200 + 390 = 1590
        expect(price).toBe(1590.00);
      });

      test('Обробка некоректних або від'ємних значень тиражу', () => {
        const specs = { productType: 'card', quantity: -50, paperId: 'paper_130', colorId: 'color_4_0', laminationId: 'lam_none' };
        const price = calculatePrice(specs, mockConfig);
        expect(price).toBe(0);
      });
    });
```

Інтеграційне тестування проводилося з метою перевірки взаємодії між Express-сервером, проміжним шаром Multer, СУБД MongoDB та хмарним сховищем подій. Тестування виконувалося за допомогою інструменту Supertest.

Ключовим тест-кейсом була перевірка стрімінгового завантаження файлу макета через Multipart-форму та автоматичне відсікання файлів, що перевищують встановлений ліміт або мають шкідливе розширення (.exe, .js замість .pdf/.tiff). Результати тестування підтвердили, що:

Файли форматів PDF та TIFF успішно проходять подвійну валідацію (по розширенню та за MIME-типом) і завантажуються в хмару з поверненням HTTP - статусу 200 OK.

Спроба передачі файлу розміром 160 Мб (при ліміті 150 Мб) коректно блокується проміжним шаром Multer, повертаючи клієнту статус 413 Payload Too Large.

Для оцінки стійкості бекенд-інфраструктури при одночасній роботі великої кількості користувачів було проведено стрес-тестування за допомогою утиліти Autocannon (або K6). Емулювалися високоінтенсивні хвилі HTTP-запитів POST /api/orders та GET /api/calculator/factors.

Параметри конфігурації тесту:

- Тривалість тесту: 60 секунд.
- Кількість віртуальних конкурентних потоків (Concurrent Users): 200.
- Сумарна кількість згенерованих запитів: ~45,000.
- Результати вимірювання метрик продуктивності зведені у таблицю 3.1.

Таблиця 3.1 – Метрики продуктивності серверного API за результатами навантажувального тестування

Метрика продуктивності	**Значення (мс / запитів)**	**Статус відповідності нормативам**
Середній час відгуку (Latency Average)	42.5 мс	Відповідає (норма < 200 мс)
Максимальний час відгуку (99th Percentile)	112.0 мс	Відповідає (норма < 500 мс)
Пропускна здатність (Throughput)	750 запитів/сек	Відповідає (норма > 500 рс)
Коефіцієнт помилок (Error Rate)	0.00%	Відповідає (норма < 1%)

Аналіз отриманих даних свідчить, що асинхронна подійна модель Node.js у поєднанні з індексуванням первинних ключів у MongoDB забезпечує лінійну масштабованість. Графік затримки залишається стабільним без експоненціальних стрибків, що підтверджує інженерну надійність обраного архітектурного стеку для промислової експлуатації Web-to-Print платформи.

3.5. Висновок до третього розділу

У третьому розділі кваліфікаційної роботи виконано повний комплекс інженерно-програмних робіт із практичної реалізації, розгортання та всебічного тестування Web-to-Print платформи для автоматизації замовлень поліграфічної продукції.

За результатами етапу розробки та досліджень можна зробити такі висновки:

- Ефективність обраного технологічного стеку: Практично підтверджено доцільність використання компонентного підходу на базі бібліотеки React.js для побудови клієнтського рівня системи. Реалізоване двостороннє зв'язування даних у поєднанні з технологією Virtual DOM дозволило створити реактивний інтерфейс калькулятора-конфігуратора. Швидкість оновлення розрахункових параметрів накладу в DOM-дереві у відповідь на дії користувача становить менше 50 мс, що гарантує високу якість UX (User Experience).
- Надійність серверної архітектури: Розроблено архітектурно стійкий шар Backend API на базі платформи Node.js та Express.js. Інтеграція об'єктно-орієнтованого драйвера Mongoose дозволила реалізувати безпечну та транзакційно стабільну взаємодію з базою даних MongoDB. Завдяки подійній моделі Node.js та протоколу WebSockets, впроваджено віджет оперативної підтримки в реальному часі, здатний підтримувати постійні TCP-з'єднання з мінімальним споживанням ресурсів процесора.
- Оптимізація обробки бінарних даних: Впровадження проміжного програмного шару Multer із конфігурацією буфера пам’яті (multer.memoryStorage()) дозволило реалізувати стрімінгову (потокову) передачу важких графічних оригінал-макетів у хмарне сховище без їх проміжного збереження на локальному диску сервера. Це підвищило

швидкість обробки файлів та ліквідувало вразливості, пов'язані з несанкціонованим виконанням коду на сервері.

- Результати верифікації та продуктивність: Проведена серія модульних тестів за допомогою фреймворка Jest довела 100% коректність роботи математичних алгоритмів ціноутворення та дискретних переходів між цифровим і офсетним типами друку. Інтеграційне та стрес-тестування за допомогою утиліти Autocannon підтвердило високу пропускну здатність API (до 750 запитів/сек) за нульового коефіцієнта помилок (Error Rate 0.00%) під навантаженням у 200 одночасних конкурентних сесій.

Таким чином, розроблений програмний комплекс повністю відповідає критеріям надійності, безпеки, масштабованості та швидкодії, що висувалися на етапі проєктування, і готовий до впровадження у реальний виробничий цикл сучасного поліграфічного підприємства.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1. Аналіз умов праці та працездатності розробника/оператора поліграфічних систем

Робота розробника та адміністраторів вебплатформи пов'язана з тривалим використанням комп'ютерної техніки (персональних комп'ютерів, серверного обладнання) та моніторів. Відповідно до ДСанПіН 3.3.2.007-98 та ГОСТ 12.0.003-74, на персонал можуть впливати такі небезпечні та шкідливі виробничі фактори.

Фізичні:

- недосконаліть або недостатність природного та штучного освітлення робочої зони;
- підвищений рівень електромагнітного та ультрафіолетового випромінювання від екранів моніторів;
- небезпечний рівень напруги в електричній мережі живлення ЕОМ (220 В, 50 Гц);
- надлишкове тепловиділення та шум від систем охолодження комп'ютерного обладнання.

Психофізіологічні:

- тривале статичне напруження м'язів спини, шиї та кистей рук (ризик розвитку тунельного синдрому);
- перенапруження зорового аналізатора через мерехтіння екранів та блиск робочих поверхонь;
- розумове перенапруження, висока концентрація уваги та тривала монотонність праці.

4.2 Організація безпечного робочого місця користувача ЕОМ та вимоги до виробничого середовища друкарні

Для мінімізації впливу шкідливих факторів робоче місце повинно бути організоване згідно з вимогами чинних санітарних норм. Параметри мікроклімату: оптимальна температура повітря в робочому приміщенні має становити 22–24 °С у холодний період року та 23–25 °С у теплий період, відносна вологість – 40–60%, швидкість руху повітря – не більше 0.1 м/с. Приміщення має бути обладнане системою кондиціонування та припливно-витяжною вентиляцією. Освітлення: робочі місця з ЕОМ слід розміщувати так, щоб природне світло падало збоку (бажано зліва). Штучне освітлення повинно бути комбінованим (загальним та місцевим). Рівень освітленості на поверхні стола в зоні розміщення документів має становити не менше 300–500 лк.

Організація робочого місця та режим праці: Висота робочого стола повинна бути в межах 725±25 мм. Крісло має бути підйомно-поворотним, регульованим за висотою та кутом нахилу спинки. Екран монітора слід розміщувати на відстані 600–700 мм від очей користувача, а кут зору повинен становити 10–20° нижче горизонталі. Для запобігання перевтомі необхідно запроваджувати регламентовані перерви по 10–15 хвилин через кожні 1.5–2 години безперервної роботи, під час яких виконувати комплекс гімнастичних вправ для очей та м'язів.

4.3 Заходи із забезпечення пожежної безпеки та електробезпеки

Комп'ютерна техніка є споживачем електричного струму промислової частоти, що створює небезпеку ураження персоналу. Відповідно до ПУЕ та ПБЕЕС, для забезпечення електробезпеки реалізовано такі технічні рішення:

Усі корпуси комп'ютерів, серверів та периферійного обладнання підключені до захисного заземлення (опір контуру заземлення не повинен перевищувати 4 Ом).

Електромережу приміщення обладнано пристроями захисного відключення (ПЗВ) та автоматичними вимикачами для миттєвого знеструмлення обладнання у разі короткого замикання або витоку струму на корпус.

Перед початком роботи персонал зобов'язаний проводити візуальний огляд цілісності ізоляції кабелів живлення, розеток та корпусів апаратури. Забороняється експлуатація обладнання зі знятими захисними кожухами.

4.4 Висновок до четвертого розділу

Приміщення з обчислювальною технікою, згідно з НАПБ Б.03.012-2007, відноситься до категорії В (пожежонебезпечне), а за ПУЕ – до класу П-IIа. Основними причинами пожеж можуть бути короткі замикання, перевантаження електромережі або порушення правил експлуатації нагрівальних приладів.

Для запобігання пожежам та захисту персоналу впроваджено такі заходи. Приміщення обладнано автоматичною системою пожежної сигналізації з тепловими або димовими сповіщувачами, яка інтегрована з системою оповіщення про евакуацію. На робочих місцях розміщено первинні засоби пожежогасіння – вуглекислотні вогнегасники типу ВВ-2 або ВВ-5 (забороняється використання пінних та водних вогнегасників для гасіння електроустановок під напругою). Розроблено та вивішено на видному місці план евакуації людей на випадок пожежі; шляхи евакуації утримуються вільними від сторонніх предметів.

ВИСНОВКИ

У кваліфікаційній роботі бакалавра розв’язано актуальне науково-практичне завдання з розробки та впровадження Web-to-Print платформи для автоматизації процесів замовлення, конфігурування та менеджменту поліграфічної продукції.

Основні наукові та практичні результати роботи полягають у наступному.

Аналіз предметної області та обґрунтування стеку: Проведено комплексний аналіз ринку оперативної поліграфії, який виявив потребу в автоматизації розрахунку вартості накладів та менеджменту замовлень. Обґрунтовано доцільність використання сучасного архітектурного стеку MERN (MongoDB, Express.js, React.js, Node.js) у поєднанні з WebSocket-протоколом для побудови високопродуктивних та масштабованих SPA-додатків.

Проєктування та моделювання: Розроблено структурно-функціональну архітектуру системи та спроєктовано нереляційну структуру бази даних MongoDB. Створено гнучку математичну модель ціноутворення, що враховує технологічні параметри друку (приладку, щільність паперу, кольоровість, постдрукарську обробку) та дискретно-крокові коефіцієнти об'єму тиражу з автоматичним визначенням типу друку (цифровий або офсетний).

Програмна реалізація клієнтської та серверної частин: На базі бібліотеки React.js реалізовано інтерактивний реактивний калькулятор-конфігуратор, який забезпечує швидкість відгуку інтерфейсу менше 50 мс.

Розроблено захищений шар Express.js REST API. Впроваджено проміжний модуль Multer з механізмом потокового запису (memoryStorage), що дозволило реалізувати безпечне стрімінгове завантаження важких оригінал-макетів (до 150 Мб) безпосередньо у хмарне сховище.

Створено адміністративну панель керування статусами замовлень та прайс-листами з інтегрованим WebSocket-інтерфейсом підтримки для асинхронного обміну повідомленнями з клієнтами в реальному часі.

Тестування та оцінка продуктивності: За допомогою фреймворка Jest проведено модульне тестування обчислювальної логіки, що підтвердило 100% коректність алгоритмів. Навантажувальне тестування за допомогою утиліти Autocannon підтвердило стабільність системи під навантаженням у 200 одночасних сесій: середня затримка (Latency Average) становила 42.5 мс за пропускної здатності 750 запитів/сек та нульовому коефіцієнті помилок.

Безпека життєдіяльності: Розроблено заходи з охорони праці, електробезпеки та пожежної безпеки для персоналу відповідно до чинних нормативних вимог України (ДСанПіН, ПУЕ, НАПБ), що забезпечує мінімізацію впливу небезпечних і шкідливих виробничих факторів.

Розроблена Web-to-Print платформа повністю виконує поставлені технічні завдання, демонструє високу інженерну надійність, масштабованість та готова до інтеграції у реальний виробничий цикл сучасних поліграфічних підприємств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чорний А. В., Козловський О. В. Автоматизація бізнес-процесів у сучасній поліграфії: концепція Web-to-Print. Поліграфія і видавнича справа. 2023. № 2 (86). С. 45–56.

2. Проєктування та розробка вебплатформ електронної комерції : навч. посіб. / за ред. М. І. Бондаренка. Київ : Четверта хвиля, 2022. 284 с.

3. Чак П. Розробка Single Page Applications на React: концепція Virtual DOM та оптимізація рендерингу. Комп’ютерні системи та мережі. 2024. Т. 12, № 1. С. 89–97.

4. Шелдон Р. NoSQL бази даних: документо-орієнтований підхід та моделювання даних в MongoDB. Львів : Магнолія, 2023. 312 с.

5. Гевко Р. Б., Рогатинський Р. М. Математичне моделювання та оптимізація процесів ціноутворення в логістичних та виробничих системах. Вісник Тернопільського національного технічного університету. 2022. № 3 (107). С. 112–121.

6. Вразливості вебзастосунків та безпека бінарних завантажень / О. П. Сидоренко та ін. Захист інформації. 2025. Т. 27, № 2. С. 140–149.

7. Банк Ф., Тейлор К. Розробка серверних систем на Node.js та Express: асинхронна архітектура та обробка потоків даних : пер. з англ. Харків : Фабула, 2024. 416 с.

8. Холмс К. Mongoose ODM: об'єктно-документне відображення для MongoDB в JavaScript екосистемах. Світ комп'ютерних технологій. 2023. № 4. С. 73–81.

9. Патерни та принципи тестування вебзастосунків: модульне та навантажувальне тестування з Jest і Autocannon / Д. М. Кравченко та ін. Сучасні комп'ютерні технології. 2025. № 1. С. 34–42.

10. ДСанПіН 3.3.2.007-98. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. Київ : Міністерство охорони здоров'я України, 1998. 23 с.

11. НПАОП 0.00-1.28-10. Правила охорони праці під час експлуатації електронно-обчислювальних машин. Київ : Держгірпромнагляд, 2010. 18 с.

12. ПУЕ:2017. Правила улаштування електроустановок. Вид. 5-те, перероб. і доп. Київ : Форт, 2017. 760 с.

13. НАПБ Б.03.012-2007. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною небезпекою. Київ : МНС України, 2007. 35 с.

14. React.js Documentation. React – A JavaScript library for building user interfaces. URL: https://react.dev/ (дата звернення: 12.05.2026).

15. Node.js API Reference. Node.js v20.x Documentation. URL: https://nodejs.org/api/ (дата звернення: 18.05.2026).

16. MongoDB Manual. The MongoDB Documentation. URL: https://www.mongodb.com/docs/manual/ (дата звернення: 22.05.2026).

17. Express.js Guide. *Fast,* unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/ (дата звернення: 25.05.2026).

18. Socket.io Protocol Specification. Bidirectional and low-latency communication for every platform. URL: https://socket.io/docs/v4/ (дата звернення: 02.06.2026).

19. Leshchyshyn, Y., Scherbak, L., Nazarevych, O., Gotovych, V., Tymkiv, P., & Shymchuk, G. (2019, May). Multicomponent Model of the Heart Rate Variability Change-point. In 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 110-113). IEEE.

20. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, September). Mathematical model of gas consumption process in the form of cyclic random process. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 232-235). IEEE.

21. Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022, April). Information Technology for

Estimating City Gas Consumption During the Year. In 2022 International Conference on Smart Information Systems and Technologies (SIST), Nur-Sultan, Kazakhstan (pp. 1-4).

22. Lytvynenko, I., Lupenko, S., Kunanets, N., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021). Simulation of gas consumption process based on the mathematical model in the form of cyclic random process considering the scale factors. In 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP (Vol. 2021).

23. Боднарчук, І., Харченко, О., Хоміцький, Б., & Шимчук, Г. (2019). Проектування архітектури програмних систем в проектах з гнучкими методами управління. Матеріали XXI наукової конференції Тернопільського національного технічного університету імені Івана Пулюя, 46-48.

24. Lupenko, S., Lytvynenko, I., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, December). Approach to gas consumption process forecasting on the basis of a mathematical model in the form of a random cyclic process. In Proceedings of the International Conference „Advanced applied energy and information technologies 2021", 2021 (pp. 213-219). TNTU, Zhytomyr «Publishing house „Book-Druk "» LLC.

25. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, H., & Hotovych, V. (2022). Additive mathematical model of gas consumption process. Вісник Тернопільського національного технічного університету, 104(4), 87-97.

26. Kunanets, N., Pasichnyk, V., Bodnarchuk, I., Martsenko, S., Matsiuk, O., Matsiuk, A., ... & Shymchuk, H. (2019). Information system for visual analyzer disease diagnostics. In CEUR Workshop Proceedings (pp. 43-56).

27. Leschyshyn, Y. Z., Nazarevych, O. B., Shymchuk, G. V., Revutskyi, E. A., & Shcherbak, L. M. (2016, September). The Methods of Change Point Detection and Statistical Estimating of Dynamic of the Noise Stochastic Signals Characteristics. In THE SEVENTH WORLD CONGRESS "AVIATION IN THE XXI-st CENTURY" Safety in Aviation and Space Technologies September 19-21, NATIONAL AVIATION UNIVERSITY. Kyiv: NAU.

28. Nazarevych, O., Leshchyshyn, Y., Lupenko, S., Hotovych, V., Shymchuk, G., & Shabliy, N. (2020, September). Method of Gas Consumption Change-point Detection Based on Seasonally Multicomponent Model. In 2020 10th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 152-155). IEEE.

29. ШИМЧУК, Г., ШЕВЧЕНКО, Н., ШВИРЛО, К., & ГАРМАТЮК, Н. (2025). СИСТЕМА ВІДНОВЛЕННЯ ДАНИХ У БЕЗДРОТОВИХ СЕНСОРНИХ МЕРЕЖАХ НА ОСНОВІ МАШИННОГО НАВЧАННЯ. Herald of Khmelnytskyi National University. Technical sciences, 353(3.2), 246-250.

30. Шимчук, Г., Голотенко, О., & Золотий, Р. З. (2022). Основні проблеми та загрози хмарної безпеки. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 59-60.

31. Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни «Розподілені системи моніторингу та керування».

32. Palianytsia, Y., Lytvynenko, I., Menoub, A., Shymchuk, H., & Dubchak, A. (2024). Development of an algorithm for identification of damage types on the surface of sheet metal.

33. Shymchuk, G., Lytvynenko, I., Hromyak, R., Lytvynenko, S., & Hotovych, V. (2023). Gas Consumption Forecasting Using Machine Learning Methods and Taking Into Account Climatic Indicators. In CITI (pp. 156-163).

34. Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни Комп’ютерна графіка для студентів освітнього рівня «бакалавр» спеціальності 125 «Кібербезпека».

35. Yasniy, O., Didych, I., Tymoshchuk, D., Pasternak, I., Nykytyuk, V., Shymchuk, H., & Radyk, D. (2026). Fatigue crack growth prediction of automotive steels using ensemble-based machine learning methods. Procedia Structural Integrity, 81, 116-122.

36. Palka, O., Stanko, A., Shymchuk, H., & Herasymchuk, O. (2021). Запобігання поширення коронавірусної інфекції у «розумних містах». COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION, (42), 79-88.

37. Шимчук, Г. В., Назаревич, О. Б., Литвиненко, Я. В., Готович, В. А., Никитюк, В. В., & Боднарчук, І. О. (2025). Грід-системи та технології хмарних обчислень. Навчальний посібник для здобувачів освітнього рівня «магістр» спеціальностей: F3 «Комп'ютерні науки», F6 «Інформаційні системи та технології».

38. Sorokivskyi, O., Hotovych, V., Nazarevych, O., & Shymchuk, G. (2025). Comparative analysis of camera calibration algorithms for football applications. Journal of Computer Vision in Sports.

39. Шевченко, Н. А., Шимчук, Г. В., & Гарматюк, У. А. (2024). Оптимізація метрик маршрутизації для забезпечення стійкості та надійності IP-мереж. Збірник тез доповідей XIII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “, 400-401.

40. Шевченко, Н. А., Шимчук, Г. В., & Гарматюк, У. А. (2024). Інтеграція технології мережевої віртуалізації VRF у багатоколійну маршрутизацію. Збірник тез доповідей XIII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій “, 398-399.

41. Шимчук, Г. В. (2022). Дослідження методів захисту відомих хмарних платформ (Master's thesis, ТНТУ).

42. Шимчук Г., Голотенко О., Небесний Р., Готович В. Застосування мови Scala у системах паралельних і хмарних обчислень. Наука і техніка сьогодні. 2026. № 4(58). С. 4794–4807. DOI: 10.52058/2786-6025-2026-4(58)-4794-4807.

43. Шевченко Н., Шимчук Г., Готович В., Голотенко О., Литвиненко С., Петрошук М. Математична модель для прогнозування змін у бездротових

сенсорних мережах. Наука і техніка сьогодні. 2026. № 4(58). С. 4767–4782. DOI: 10.52058/2786-6025-2026-4(58)-4767-4782.

ДОДАТКИ

Додаток А

```
// Модуль динамічного розрахунку вартості поліграфічного накладу
export const calculatePrice = (specifications, pricingConfig) => {
  if (!pricingConfig) return 0;

  const { quantity, paperId, colorId, laminationId } = specifications;
  const N = parseInt(quantity);

  if (isNaN(N) || N <= 0) {
    return 0;
  }

  // Визначення базової вартості запуску обладнання (приладки)
  let setupCost = pricingConfig.setupCosts.digital_print;
  if (N >= 1000) {
    setupCost = pricingConfig.setupCosts.offset_print;
  }

  // Розрахунок коефіцієнта ступеня регресії тиражу (дисконт об'єму)
  let kVolume = 1.00;
  if (N >= 100 && N < 500) kVolume = 0.90;
  else if (N >= 500 && N < 1000) kVolume = 0.80;
  else if (N >= 1000) kVolume = 0.65;

  const baseCostPerUnit = 0.50;
  const kPaper = pricingConfig.paperOptions.find(o => o.id === paperId)?.coeff || 1.0;
```

```
        const kColor = pricingConfig.chromaticityOptions.find(o => o.id === colorId)?.coeff || 1.0;
        const kLam = pricingConfig.laminationOptions.find(o => o.id === laminationId)?.coeff || 1.0;

        // Математична модель ціноутворення
        const calculatedPrice = setupCost + (N * (baseCostPerUnit * kVolume) * kPaper * kColor * kLam);

        return Number(calculatedPrice.toFixed(2));
      };
```

Додаток Б

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const helmet = require('helmet');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 5000;

app.use(helmet());
app.use(cors({ origin: process.env.CLIENT_URL }));
app.use(express.json());

// Асинхронне підключення до СКБД MongoDB через Mongoose ODM
mongoose.connect(process.env.MONGODB_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('Кластер MongoDB успішно підключено.'))
.catch(err => console.error('Помилка підключення до бази даних:', err));

// Реєстрація модулів маршрутизації
app.use('/api/auth', require('./routes/auth.routes'));
app.use('/api/orders', require('./routes/order.routes'));
app.use('/api/calculator', require('./routes/calculator.routes'));

app.listen(PORT, () => {
  console.log(`Сервер Web-to-Print платформи запущено на порту ${PORT}`);
});
```

Додаток В

```
const multer = require('multer');
const path = require('path');

// Використання Memory Storage для оптимізації потокової передачі в хмару
const storage = multer.memoryStorage();

// Валідація MIME-типів та розширень вхідних файлів макетів
const fileFilter = (req, file, cb) => {
  const allowedExtensions = /pdf|tiff|tif|psd|ai/;
  const extname = allowedExtensions.test(path.extname(file.originalname).toLowerCase());
  const mimetype = allowedExtensions.test(file.mimetype);

  if (mimetype && extname) {
    return cb(null, true);
  }
  cb(new Error('Помилка: Дозволені лише професійні формати макетів (PDF, TIFF, PSD, AI).'));
};

const upload = multer({
  storage: storage,
  limits: { fileSize: 150 * 1024 * 1024 }, // Обмеження розміру файлу макета до 150 Мб
  fileFilter: fileFilter
});

module.exports = upload;
```