Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

Веб застосунок для обліку замовлень та планування розкрою листових матеріалів
(тема)

Виконав:
здобувач четвертого року навчання
групи ПЗПІ-21-1

Михайло СЕРЕДА
(Власне ім'я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма Програмна інженерія
(повна назва освітньої програми)

Керівник доц. кафедри ПІ Андрій БАБІЙ
(посада, Власне ім'я, ПРІЗВИЩЕ)

Допускається до захисту
Зав. кафедри _____ Кирило СМЕЛЯКОВ
(підпис) (Власне ім'я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти перший (бакалаврський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми Освітньо-професійна
Освітня програма Програмна Інженерія
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Середі Михайлу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Веб застосунок для обліку замовлень та планування розкрою листових матеріалів
Затверджена наказом по університету від 19.05.2025р. № 397 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 13.06.2025
3. Вихідні дані до роботи Розробити веб застосунок для обліку замовлень та планування розкрою листових матеріалів використовуючи платформу .NET, API фреймворк ASP.NET Core і фронтенд фреймворк Blazor, мовою C#.
4. Перелік питань, що потрібно опрацювати в роботі
Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	15.04.25	*виконано*
2	Створення специфікації ПЗ	16.04.25	*виконано*
3	Проектування ПЗ	27.04.25	*виконано*
4	Розробка ПЗ	10.05.25	*виконано*
5	Тестування ПЗ	14.05.25	*виконано*
6	Оформлення пояснювальної записки	29.05.25	*виконано*
7	Підготовка презентації та доповіді	05.06.25	*виконано*
8	Попередній захист	10.06.25	*виконано*
9	Нормоконтроль, рецензування	09.06.25	*виконано*
10	Здача роботи у електронний архів	09.06.25	*виконано*
11	Допуск до захисту у зав. кафедри	13.06.25	*виконано*

Дата видачі завдання «19» «квітня» 2025р.

Здобувач  _____
(підпис)

Керівник роботи _____ доц. кафедри ПІ Андрій БАБІЙ
(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра, 59 стор., 4 рис., 16 джерел.

ВЕБ-ЗАСТОСУНОК, ВЕРСТАТ, ЗАМОВЛЕННЯ, ЛИСТОВИЙ МАТЕРІАЛ, РОЗКРІЙ, СИСТЕМА, ТОВЩИНА, ФАЙЛ DXF, ФАЙЛ SVG.

Об’єкт розробки – веб-орієнтована інформаційна система для обліку замовлень та планування розкрою листових матеріалів.

Мета розробки – створення програмної системи, яка дозволяє ефективно адмініструвати замовлення клієнтів, керувати матеріалами та верстатами, а також виконувати планування розкрою.

Метод рішення – побудова клієнт-серверної архітектури з ASP.NET Core та Blazor WebAssembly із використанням бази даних PostgreSQL, реалізація зберігання креслень у форматі DXF, конвертація у SVG для перегляду, створення інтерфейсу ручного розміщення моделей на аркуші з подальшим формуванням файлів розкрою.

У результаті розроблено MVP-версію інформаційної системи, яка дозволяє приймати замовлення, працювати з кресленнями DXF, виконувати планування розкрою, керувати верстатами та матеріалами, а також формувати виробничі файли DXF і SVG для подальшої обробки.

ABSTRACT

WEB APPLICATION, MACHINE, ORDER, SHEET MATERIAL, CUTTING, SYSTEM, THICKNESS, DXF FILE, SVG FILE.

The object of development is a web-based information system for managing orders and planning the nesting of sheet materials.

The purpose of the work is to create a software system that enables efficient administration of customer orders, management of materials and machines, and planning of sheet material cutting.

Solution Method – implementation of a client-server architecture using ASP.NET Core and Blazor WebAssembly with a PostgreSQL database, support for storing drawings in DXF format, converting them to SVG for viewing, and creating an interface for manual placement of models on a sheet with subsequent generation of cutting files.

As a result of the development, an MVP version of the information system was developed, enabling order processing, DXF drawing management, cutting planning, machine and material management, and generation of DXF and SVG production files for further processing.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

МДФ – англ. Medium-Density Fibreboard

ПВХ – Поливинилхлорид

DXF – Drawing Exchange Format

SVG – Scalable Vector Graphics

ЧПУ – Числове Програмне Управління

CAD – Computer-Aided Design

CAM – Computer-Aided Manufacturing

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

CQRS – Command Query Responsibility Segregation

DI – Dependency Injection

REST – Representational State Transfer

JSON – JavaScript Object Notation

ВСТУП

У сучасних умовах зростаючої конкуренції та попиту на індивідуальні виробничі рішення, особливої актуальності набуває розробка інструментів для цифровізації процесів обробки листових матеріалів. Виробничі сервіси, що займаються різанням металу, фанери, пластику та інших листових матеріалів, стикаються з потребою у швидкому, зрозумілому та контрольованому процесі прийому замовлень, планування розкрою та відстеження виконання.

На світовому ринку вже існує ряд програмних продуктів, орієнтованих на оптимізацію розміщення моделей на аркуші з метою мінімізації відходів. Втім, ці системи переважно зосереджені на алгоритмах автоматичного розміщення і часто не враховують потреби в організаційному адмініструванні сервісного процесу розкрою, зокрема – в управлінні ролями користувачів, чергами станків, та ручному плануванні.

Актуальність розробки зумовлена необхідністю створення інтегрованої веб-системи, яка дозволить сервісам з розкрою матеріалів ефективно управляти замовленнями, налаштовувати обладнання, здійснювати ручне планування моделей на листі з урахуванням типу матеріалу та товщини, і в подальшому – розширюватися до автоматизованої оптимізації.

Метою цієї роботи є створення веб-орієнтованого застосунку, який дозволяє:

- замовникам – подавати креслення на розкрій із вказаною кількістю копій;
- адміністраторам сервісів – налаштовувати перелік підтримуваних матеріалів і параметри станків;
- працівникам сервісів – планувати оптимізований розкрій вручну через редактор, призначення на станки та керування чергою обробки.

На відміну від існуючих систем, які фокусуються на алгоритмічному розміщенні фігур, запропонована система надаватиме гнучкий інтерфейс керування процесом розкрою як адміністративним, так і виконавчим ролям. Система покликана стати базовим цифровим інструментом для малого та середнього бізнесу, що надає послуги розкрою широкого спектра матеріалів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Аналіз предметної галузі

Розкрій листових матеріалів є однією з ключових технологічних операцій у багатьох галузях виробництва – зокрема, у металообробці, меблевому виробництві, виготовленні рекламної продукції, будівництві, суднобудуванні, приладобудуванні тощо. Суть цього процесу полягає у перетворенні стандартних аркушів сировини (металу, фанери, МДФ, ПВХ, композитів, скла) на деталі заданої форми, відповідно до креслень замовника [1], з мінімальними втратами матеріалу.

Класичний цикл розкрою передбачає декілька послідовних етапів:

- отримання креслень від клієнта (найчастіше у форматі DXF);
- перевірка відповідності технічних параметрів;
- планування розміщення моделей на листі з урахуванням матеріалу і товщини;
- виконання розкрою на верстаті з ЧПУ;
- передача готових деталей клієнту.

Сучасні виробничі процеси дедалі більше орієнтовані на автоматизацію та цифрову трансформацію. Багато підприємств поступово впроваджують CAD/CAM-системи [2], які дозволяють автоматизувати підготовку креслень та управління обробкою на верстатах. Проте, незважаючи на ці тенденції, значна кількість малих і середніх підприємств продовжують використовувати фрагментовані або ручні методи планування розкрою. У таких випадках оператори змушені вручну координувати розміщення моделей, прийом замовлень, зберігання файлів та ведення черг на верстатах. Це призводить до затримок, людських помилок, неефективного використання ресурсів та зростання витрат.

На ринку присутні програмні продукти, які частково закривають потреби у плануванні розкрою, але здебільшого зосереджені лише на геометричній оптимізації. До таких рішень належать, зокрема:

- CutLogic 2D [3] – потужне програмне забезпечення для автоматизованого розміщення деталей, але потребує глибокого навчання та є комерційним продуктом;
- NestFab [4] – система автоматичного вкладення моделей, яка орієнтована на CAM-вихід, але не має сервісного компонента;
- Cutting Optimization Pro [5] – рішення, обмежене роботою з прямокутниками і не пристосоване до криволінійних форм;
- Smart2DCutting [6] – має базовий функціонал, інтеграцію з Excel, але не підтримує розмежування користувачів і веб-інтерфейс.

Спільним недоліком більшості таких рішень є відсутність інтеграції з реальними сервісними процесами. Більшість програм фокусуються виключно на оптимальному геометричному розміщенні моделей на аркуші, нехтуючи при цьому логікою обробки замовлень, управління виробничими ресурсами та потребами кінцевих користувачів. Вони не підтримують менеджмент клієнтів, не дозволяють адмініструвати перелік матеріалів та їх товщин, не враховують зв’язок моделей із конкретними верстатами та їх технічними обмеженнями. Також їм бракує механізмів роботи зі статусами замовлень, чергами на верстатах, багатокористувацькими ролями та правами доступу.

Водночас на практиці сервіси розкрою стикаються з широким спектром задач, які виходять далеко за межі простої геометричної оптимізації. Замовник, який звертається до сервісу, очікує не лише точний розкрій, а й повну цифрову прозорість процесу: можливість відстежити статус свого замовлення, побачити, які деталі були розміщені, коли замовлення потрапить на верстат тощо. З іншого боку, працівники сервісу потребують інструментів для планування, контролю, керування виробничим навантаженням.

У зв’язку з цим користувачам бракує функціоналу для зручної інтеграції із робочими процесам:

- зберігати замовлення у структурованому вигляді: кожне замовлення може містити параметри – такі як матеріал, товщина, кількість, прикріплений

файл DXF, статус і дату створення – що дозволятиме уникати плутанини, забезпечувати простий пошук і повторне використання;

- забезпечувати інтерактивне планування розміщення моделей: у багатьох випадках оператор може хотіти розташовувати моделі, враховуючи особливості конкретного верстата, залишки матеріалу або подальшу обробку за допомогою веб-інтерфейсу, який надавав би зручний інструмент для планування з візуалізацією всіх розміщених моделей на аркуші;
- автоматично оновлювати статуси замовлень відповідно до їх етапу обробки: це дало б можливість оператору або адміністратору не витрачати час на ручне перемикання статусів, а клієнтам – бачити актуальний стан виконання замовлення в режимі реального часу;
- контролювати сумісність моделей із параметрами верстата із автоматичною перевіркою, чи підтримує обраний верстат зазначений матеріал, товщину, розміри листа та попереджати у разі невідповідності. Це зменшувало б кількість помилок на етапі планування;
- давати змогу вказувати кількість копій для кожної моделі: коли замовлення містить повторювані деталі, оператор хотів би мати змоги додавати кілька екземплярів однієї моделі на аркуш і керувати їх розміщенням;
- надавати прозорий канал взаємодії з клієнтом: клієнт хотів би мати доступ до власного кабінету, де відображатимуться всі замовлення, їхній статус, прев’ю креслення та інформація про обробку, це підвищувало б довіру до сервісу й зменшуватиме кількість уточнюючих звернень.

Таким чином, реальні користувачі – як клієнти, так і працівники сервісу – потребують системи, що поєднує оптимізацію виробництва з керованістю, інтерактивністю та зручністю. Простого "розміщення на аркуші" недостатньо: потрібне повноцінне середовище обліку, контролю та планування процесів розкрою.

У цьому контексті особливе значення має вибір формату файлів для завантаження креслень. Найбільш поширеним і технічно доцільним у виробництві є формат DXF (Drawing Exchange Format) – відкритий векторний формат, розроблений компанією Autodesk спеціально для обміну даними між CAD-системами [7]. DXF підтримується більшістю верстатів з ЧПУ, дозволяє зберігати точні координати геометрії, працює у фізичних одиницях та легко обробляється серверною логікою. Його відкритість, стабільність і орієнтація на виробництво дозволяють обрати цей формат як єдиний у рамках початкових версій системи. Це дає змогу стандартизувати сценарії завантаження, забезпечити сумісність із CAM-обладнанням та мінімізувати технічні складнощі у впровадженні.

Таким чином, сформувалася необхідність у створенні спеціалізованої інформаційної системи, яка буде не просто програмним оптимізатором розкрою, а повноцінною платформою керування сервісом розкрою матеріалів, що поєднує замовлення, планування, візуалізацію та контроль процесів.

1.2 Виявлення та вирішення проблем

У результаті аналізу предметної галузі та дослідження існуючих програмних рішень у сфері розкрою листових матеріалів було виявлено низку типових недоліків, з якими стикаються підприємства при автоматизації цього процесу. Особливо гостро ці проблеми проявляються в умовах недостатньої інтеграції між технічними засобами, замовленням клієнта та виробничим циклом. Підсумуємо і деталізуємо основні проблеми, що стали основою для формування цілей і функціональних вимог до майбутньої інформаційної системи.

Проблема 1 – відсутність централізованого обліку замовлень. У більшості сервісів розкрою інформація про замовлення передається неформалізовано (через пошту, месенджери) та зберігається у вигляді розрізнених файлів. Це ускладнює керування процесом, ведення історії замовлень, повторне використання креслень, а також контроль за поточним станом виконання. Рішенням є впровадження

централізованої системи, де кожне замовлення фіксується зі своїми параметрами, прикріпленим кресленням (DXF), кількістю копій і статусом.

Проблема 2 – розрив між замовленням і ручним плануванням розкрою. На практиці між створенням замовлення та його обробкою на аркуші розкрою немає узгодженого процесу: розміщення моделей здійснюється поза системою, без перевірки сумісності з верстатом, матеріалами, товщиною чи автоматичної генерації виробничих файлів. Через це можливі помилки, втрати, неузгодженість дій працівників. Рішенням є впровадження єдиного цифрового процесу, в якому оператор може розміщувати моделі замовлень безпосередньо у вбудованому веб-редакторі, із збереженням геометрії, кількості копій і відповідно вибраним матеріалам. Планування інтегрується із замовленням і змінює статус замовлення.

Проблема 3 – відсутність підтримки масштабованого управління ресурсами та виробничими чергами. Типові сервіси не мають повноцінного механізму організації роботи кількох верстатів, які працюють з різними матеріалами і товщинами. Відсутній облік сумісності ресурсів, централізоване керування чергами або механізм динамічного планування з урахуванням технічних обмежень обладнання. Рішенням є впровадження системи з незалежними чергами для кожного верстата, перевіркою допустимості матеріалів, можливістю прив’язки ресурсів до обладнання, а також підтримкою масштабування для розширення виробничої бази.

1.3 Постановка задачі

На основі виявлених особливостей предметної галузі та аналізу типових проблем, що притаманні сучасним сервісам розкрою листових матеріалів, сформульовано технічне завдання на розробку програмної системи.

Метою є створення веб-орієнтованої інформаційної системи, яка дозволить автоматизувати процеси прийому замовлень, управління ресурсами (матеріалами, товщинами, верстатами), планування розкрою аркушів, формування виробничих файлів, а також моніторингу виконання.

Для досягнення цієї мети система повинна реалізовувати наступні функції:

- прийом замовлень від клієнтів із можливістю завантаження креслення у форматі DXF, вказання матеріалу, товщини та кількості копій;
- збереження замовлень у структурованому вигляді з можливістю перегляду статусів;
- адміністрування переліку доступних матеріалів і товщин;
- створення та налаштування верстатів з вказанням допустимих матеріалів та розмірів листа;
- створення аркуша розкрою з обранням верстата, матеріалу та габаритів;
- розміщення моделей замовлень на аркуші з фіксацією координат і кількості копій;
- формування оновленого DXF-файлу листа;
- керування чергою аркушів на кожному верстаті;
- автоматичне оновлення статусів замовлень залежно від етапу обробки;
- ізоляція доступу відповідно до ролей користувачів: клієнт, працівник сервісу, адміністратор сервісу.

Система повинна бути реалізована у вигляді клієнт-серверного веб-застосунку з використанням сучасних веб-технологій, бути масштабованою та такою, що допускає подальше розширення (зокрема, шляхом додавання алгоритмів автоматичної оптимізації розкрою).

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Окреслення концепції

Концепція розроблюваної системи полягає у створенні веб-орієнтованого програмного забезпечення для сервісів розкрою листових матеріалів, яке охоплює не лише процес геометричного розміщення моделей на листі, а й повний цикл керування замовленнями, ресурсами, чергами обробки та статусами виконання.

На відміну від традиційних CAD/CAM систем або автономних програм для оптимізації розкрою, дана система розробляється як сервісна платформа з розмежуванням ролей, інтерактивним інтерфейсом для працівників, можливістю масштабування на декілька верстатів, та інтеграцією з веб-технологіями, що не вимагають інсталяції на стороні клієнта.

Основні концептуальні засади рішення:

- клієнт-серверна архітектура: система складатиметься з ASP.NET Core Web API [8] (серверна частина) і Blazor WebAssembly [9] (клієнтська частина), що забезпечує масштабованість, безпечний доступ та швидке оновлення інтерфейсу;
- модульність і рольовий доступ: чітко розмежовані функції для замовників, операторів і адміністраторів сервісів, кожна роль має свій набір доступних дій і ресурсів;
- орієнтація на виробничі нюанси: замовлення класифікуються за матеріалом та товщиною, які повинні відповідати технічним можливостям конкретного верстата, розміщення моделей можливе тільки на листах сумісного типу;
- ручне планування з візуальним редактором: користувач отримує можливість розміщувати моделі на віртуальному аркуші з подальшим призначенням листа на обраний верстат;
- менеджмент черги на верстати: кожен верстат має окрему чергу обробки, а зміна статусу аркуша автоматично оновлює пов’язані замовлення;

– формат креслень DXF: використовується як основний обмінний формат, що підтримує високоточну векторну геометрію та сумісний із CAM-процесами більшості верстатів з ЧПУ;
– можливість розширення: архітектура дозволяє у майбутньому інтегрувати автоматичні алгоритми оптимізації, підтримку інших форматів, API для зовнішніх сервісів та мультисервісну модель.

2.2 Головна функціональність

Програмна система реалізує набір функціональних можливостей, які забезпечують повний цикл керування замовленнями, ручного планування розкрою та контролю черг на верстатах. Усі функції згруповані відповідно до ролей користувачів і логічних модулів системи.

Функції клієнта (замовника) наступні:

– MF-1: створення нового замовлення через веб-інтерфейс із вказанням матеріалу, товщини та кількості копій;
– MF-2: завантаження креслення у форматі DXF для кожного замовлення;
– MF-3: перегляд списку власних замовлень із фільтрацією за статусом;
– MF-4: ознайомлення з поточним статусом виконання кожного замовлення.

Функції працівника сервісу наступні:

– MF-5: перегляд усіх замовлень, що очікують на розміщення для обробки;
– MF-6: створення листа (аркуша розкрою) із вибором матеріалу, товщини та верстата;
– MF-7: розміщення моделей на аркуші;
– MF-8: вибір кількості копій моделі, що має бути розміщена;
– MF-9: видалення або переміщення моделей на аркуші в редакторі;
– MF-10: збереження розміщення на аркуші, що призводить до генерації плану та оновлення статусів замовлень;
– MF-11: призначення аркуша на конкретний верстат із подальшим додаванням у чергу;

– MF-12: завершення обробки листа, що змінює статус усіх пов’язаних замовлень на Done;
– MF-13: видалення листа з поверненням замовлень у статус Created.

Функції адміністратора сервісу наступні:

– MF-14: створення та редагування довідника матеріалів із вказанням типу та доступних товщин;
– MF-15: створення та редагування верстатів із вказанням максимальних розмірів листа та списку допустимих матеріалів;
– MF-16: призначення матеріалів верстатам;
– MF-17: управління обліковими записами працівників сервісу.

Системні функції (спільні):

– MF-18: аутентифікація та авторизація користувачів;
– MF-19: автоматичне оновлення статусу замовлення залежно від дій оператора;
– MF-20: захищене зберігання файлів креслень на сервері (без публічних URL);
– MF-21: генерація SVG-прев’ю на основі DXF-файлу;
– MF-22: завантаження DXF-файлів працівником або адміністратором для подальшого використання.

2.3 Припущення та залежності

Припущення системи наступні:

– A-1: усі користувачі системи мають доступ до інтернету через сучасний веб браузер (Chrome, Firefox, Edge);
– A-2: клієнти завантажують креслення у форматі DXF, що відповідає мінімальному стандарту підтримки (наприклад, 2D полілінії, без вкладених блоків);
– A-3: для кожного замовлення припускається використання одного типу матеріалу й однієї товщини;

– A-4: усі верстати попередньо сконфігуровані адміністратором сервісу з актуальними параметрами (допустимі матеріали, розміри листів тощо);
– A-5: розміщення моделей на листі у редакторі здійснюється вручну – автоматичне оптимізоване розміщення у версії MVP не передбачено;
– A-6: статуси замовлень змінюються лише через визначені дії у системі (збереження листа, завершення обробки замовлення, видалення тощо).

Залежності системи наступні:

– D-1: облікові записи користувачів керуються внутрішньою системою авторизації на основі ASP.NET Identity та JWT токенів;
– D-2: візуальний редактор базується на можливостях Blazor WebAssembly та обробки SVG;
– D-3: дані про матеріали, товщини та верстати зберігаються в централізованій базі даних і повинні бути актуальними для коректної роботи логіки створення листів;
– D-4: кожен аркуш розкрою може бути оброблений лише на одному верстаті, обраному вручну оператором;
– D-5: завантажені DXF-файли зберігаються у хмарному файловому сховищі (наприклад, Azure Blob Storage або AWS S3), доступ до файлів здійснюється виключно через авторизовані API з перевіркою ролі користувача та приналежності до замовлення. Пряма адреса до файлу не публікується.

2.4 Рамки та обмеження проєкту

2.4.1 Рамки первинного випуску

Первинний випуск системи реалізує базову, але функціонально завершену версію програмного забезпечення, що дозволяє виконати повний цикл обробки замовлення на розкрій – від моменту створення замовником до завершення обробки на верстаті. Основна мета MVP – впровадити ключову бізнес-логіку, що відображає реальні виробничі процеси, забезпечивши при цьому гнучку основу для подальшого розвитку.

У рамках MVP буде реалізовано:

а) рольову модель доступу:

1) ролі: замовник, працівник сервісу, адміністратор сервісу;
2) аутентифікація через JWT;
3) веб-інтерфейс для кожної ролі.

б) менеджмент замовлень:

1) створення замовлення із завантаженням DXF-файлу;
2) вказання матеріалу, товщини, кількості копій;
3) перегляд статусу та візуалізація креслення (SVG-прев'ю).

в) довідники матеріалів і верстатів:

1) CRUD для матеріалів із переліком допустимих товщин;
2) CRUD для верстатів з параметрами листа та прив'язкою матеріалів.

г) інтерактивне планування розкрою:

1) ручне розміщення моделей у редакторі;
2) перевірка типу матеріалу і товщини для листа;
3) вибір кількості копій моделі;
4) збереження плану розміщення.

д) менеджмент листів і черги:

1) призначення листа на верстат;
2) черга для кожного верстата;
3) завершення або скасування обробки листа;
4) масове оновлення статусів замовлень.

е) обробка файлів креслень:

1) завантаження DXF-файлів;
2) зберігання у файловій системі або хмарному сховищі;
3) генерація SVG-прев'ю.

ж) захист доступу:

1) перевірка ролей при доступі до ресурсів;
2) відсутність публічного доступу до креслень;
3) обмеження дій для незареєстрованих користувачів.

MVP орієнтований на ручне планування, що відображає реальні потреби малого та середнього бізнесу у розкрої, забезпечує повний цикл роботи і не містить складних алгоритмів автоматизації, які заплановані на наступні фази.

2.4.2 Рамки наступних випусків

Після успішної реалізації MVP-платформи передбачається поступове розширення функціональності системи відповідно до потреб користувачів, зростання навантаження та інтеграційних вимог. Основні напрямки розвитку системи у наступних релізах:

а) автоматизація планування розкрою:

1) інтеграція алгоритмів геометричної оптимізації (наприклад, Guillotine, Nesting);
2) можливість порівняння ручного й автоматичного розміщення;
3) автовибір верстата за заданими параметрами.

б) розширена підтримка форматів:

1) підтримка альтернативних форматів креслень;
2) автоматичне конвертування популярних форматів у DXF або внутрішній формат.

в) модуль повторного замовлення:

1) збереження типових конфігурацій;
2) можливість "повторити замовлення" без повторного завантаження креслення.

г) моніторинг і аналітика:

1) візуалізація завантаження верстатів, часів простою, кількості виконаних деталей;
2) звіти для адміністратора сервісу.

д) облік матеріалів:

1) облік використаних аркушів і розрахунок залишків;
2) механізм зарезервованих матеріалів під заплановані замовлення.

2.4.3 Обмеження та винятки

Розроблювана система у версії MVP має низку усвідомлених обмежень, які зумовлені технічними, часовими та стратегічними рамками поточного етапу реалізації. Вони не є недоліками, а визначають фокус і межі відповідальності рішення на першому етапі впровадження.

Обмеження MVP:

- система не виконує автоматичну оптимізацію розміщення моделей на листі, усі дії здійснюються вручну оператором;
- підтримується лише один формат креслень – DXF, інші формати не конвертуються і не приймаються;
- система не здійснює складський облік залишків матеріалу, облік ведеться на рівні вибору матеріалу при плануванні, без кількісного контролю;
- відсутня багатомовність інтерфейсу – реалізовано лише одну мову;
- платіжна система або облік оплати за замовлення – відсутні й не входять до обсягу функціоналу;
- ідентифікація клієнтів за юридичними особами, контрактами або ІПН – не реалізується, авторизація здійснюється тільки за e-mail/паролем.

3 АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 UML проєктування ПЗ

Для формалізації основних функціональних можливостей програмної системи було побудовано діаграму випадків використання. Вона відображає взаємодію основних користувачів із системою та визначає межі її відповідальності.

У проєкті визначено три типи акторів:

– клієнт (Замовник) - користувач, який створює замовлення на розкрій матеріалу, завантажує креслення та відстежує статус виконання (див. рис. 3.1);
– працівник сервісу - відповідальний за планування розміщення моделей на листі, призначення на верстат і виконання обробки (див. рис. 3.2);
– адміністратор сервісу - користувач, який здійснює налаштування матеріалів, верстатів та керує акаунтами працівників (див. рис. 3.3).

Кожен актор взаємодіє лише з тими процесами, які відповідають його функціональній зоні відповідальності.

Діаграма охоплює такі ключові сценарії:

– створення замовлення клієнтом;
– завантаження креслення та вказання параметрів матеріалу;
– перегляд статусів замовлень;
– планування розкрою працівником сервісу;
– керування чергою листів;
– адміністрування довідників матеріалів і верстатів.

Діаграма визначає межі системи як інтерфейс для прийому замовлень, планування розкрою, керування ресурсами та забезпечення контрольованого виконання.

Нижче наведено графічне зображення сценаріїв взаємодії кожного з користувачів із системою. Кожна діаграма ілюструє доступні функції відповідно до ролі користувача.

Рисунок 3.1 – Use-case діаграма клієнта

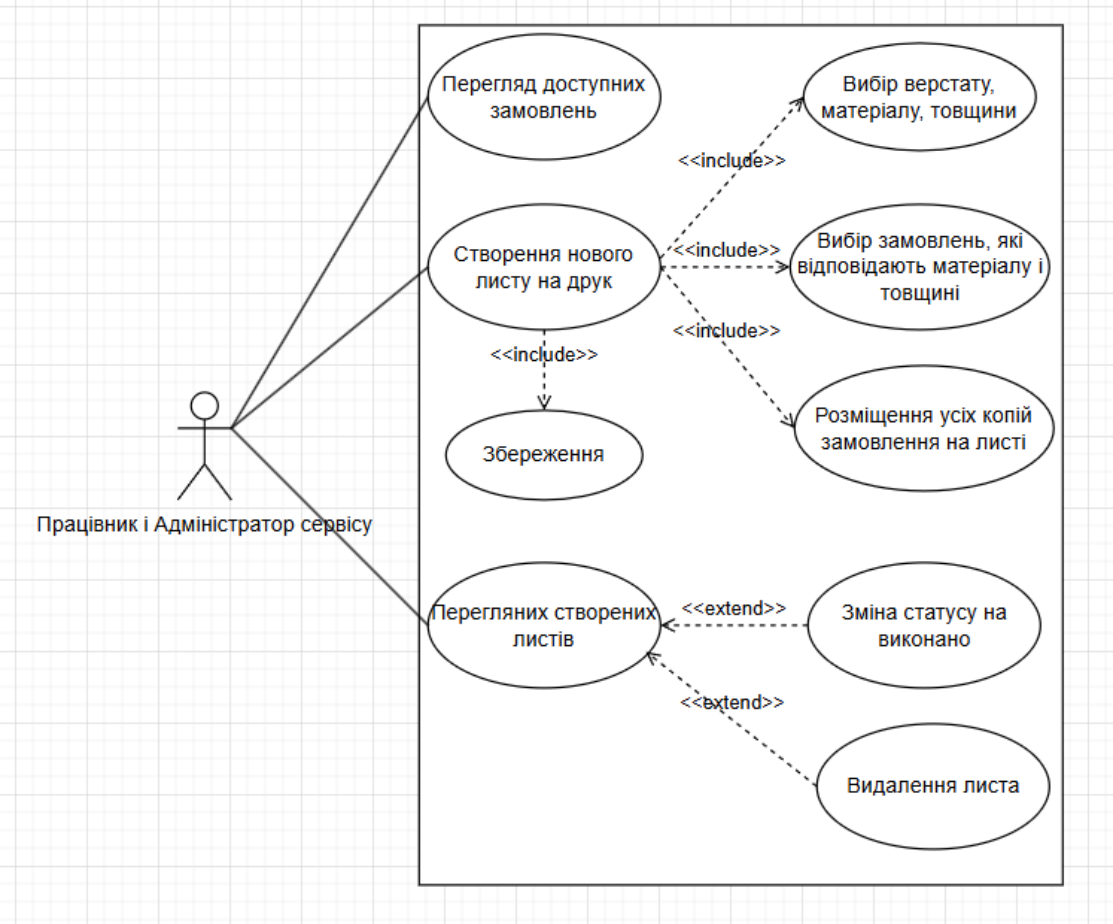

Рисунок 3.2 – Use-case діаграма працівника та адміністратора сервісу

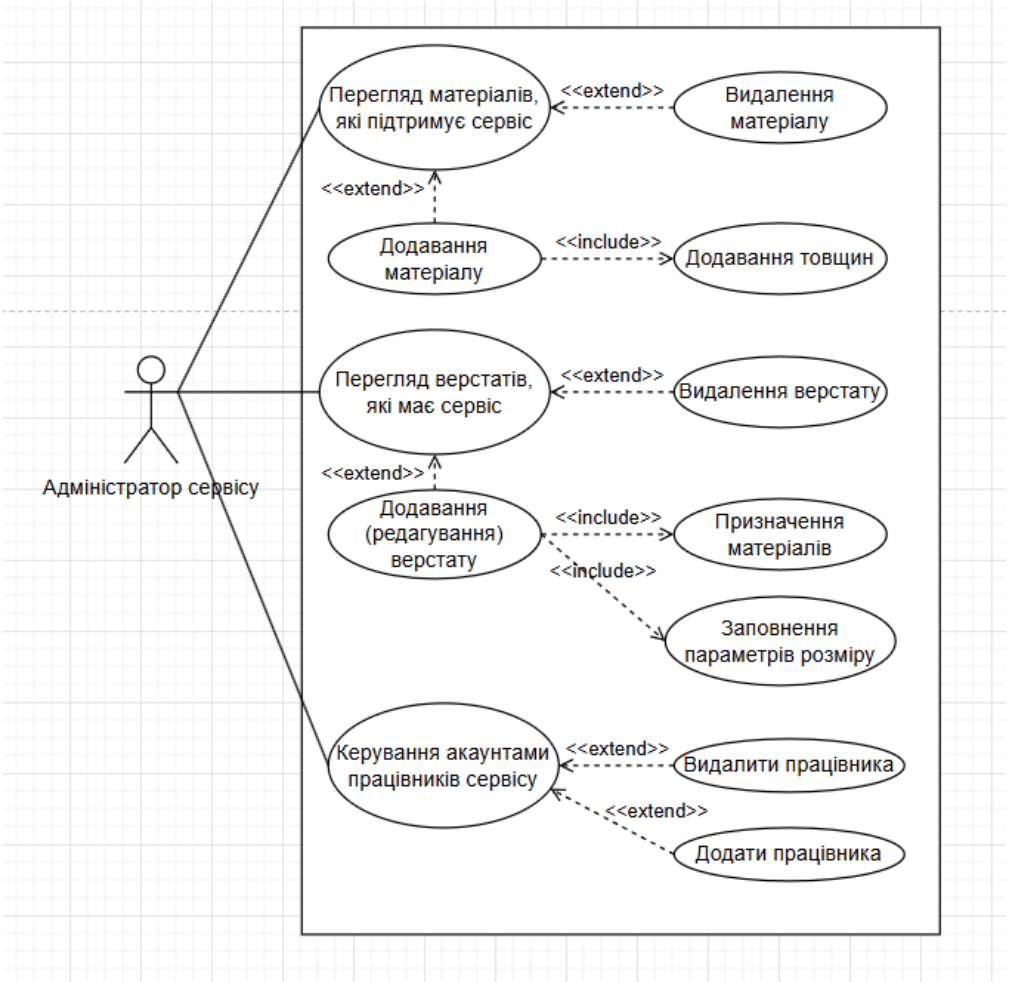

Рисунок 3.3 – Use-case діаграма адміністратора сервісу

Дані діаграми випадків використання формують основу для подальшого проєктування архітектури системи. Вони дозволяють чітко визначити ролі користувачів, межі відповідальності та основні бізнес-процеси, що підтримуються програмним забезпеченням.

3.2 Проєктування архітектури ПЗ

Архітектура системи побудована за моделлю розділення відповідальностей між клієнтською та серверною частинами. Основна задача архітектури – забезпечити стабільне виконання бізнес-логіки, просте адміністрування ресурсів сервісу (матеріалів, верстатів, замовлень), а також можливість ручного планування розкрою з урахуванням виробничих обмежень.

3.2.1 Загальна концепція архітектури

Система реалізує класичну клієнт-серверну архітектуру. Усі дані та бізнес-логіка зосереджені на серверній частині, тоді як клієнтська частина відповідає за інтерфейсну взаємодію та відображення інформації.

Клієнтська частина реалізована за допомогою Blazor WebAssembly – сучасної технології, яка дозволяє виконувати C#-код безпосередньо у браузері користувача через WebAssembly. Це означає, що інтерфейс застосунку завантажується лише один раз і повністю виконується на стороні клієнта без додаткових звернень до сервера при зміні сторінок. Такий підхід забезпечує гнучкість і високу швидкодію, особливо у випадку інтерактивного середовища, такого як редактор аркушів розкрою, де важлива локальна обробка координат, переміщення об’єктів та моментальне оновлення UI.

Серверна частина побудована на платформі ASP.NET Core Web API. Вона реалізує бізнес-логіку, обробляє HTTP-запити, відповідає за взаємодію з базою даних PostgreSQL [10], виконує генерацію виробничих файлів (DXF, SVG), зберігає файли креслень та контролює доступ користувачів залежно від ролей. Уся логіка модульно організована, з використанням шаблонів CQRS [11] та MediatR, що дозволяє легко масштабувати проєкт і підтримувати розділення відповідальності [12].

Вибір технологій обґрунтований такими факторами:

- єдина мова програмування (C#) для клієнта і сервера;
- висока інтеграція Blazor із ASP.NET Core;
- можливість запуску без додаткових JavaScript-фреймворків;
- відмінна підтримка роботи з формами, маршрутизацією, DI;
- зручність обробки креслень і геометрії безпосередньо в клієнті.

У майбутніх версіях системи планується перехід на Blazor Web App (Hybrid Model) – нову модель, яка поєднує переваги обох підходів: серверного (Blazor Server) і клієнтського (Blazor WebAssembly). Такий перехід дозволить:

- зменшити час першого завантаження (завдяки попередній генерації сторінок на сервері);

- покращити продуктивність у слабких браузерах;
- водночас зберегти реактивність і незалежність для редактора аркуша, що залишатиметься на WebAssembly.

Такий архітектурний підхід забезпечує як швидкий розвиток MVP-версії, так і надійне технологічне підґрунтя для подальшого масштабування, оптимізації продуктивності й інтеграції з іншими системами.

3.2.2 Логічна структура серверної частини

Архітектура серверної частини поділена на шари, кожен з яких виконує чітко визначену роль [13]:

- контролери (API Layer): приймають запити від клієнтської частини, проводять базову валідацію запитів, викликають відповідні команди або запити, використовують REST API з маршрутами, структурованими за принципом ресурсів (Orders, Sheets, Materials тощо);
- обробники запитів і команд (CQRS Layer): логіка розділена на запити (Query) для читання даних і команди (Command) для зміни стану, усі обробники реалізовані через бібліотеку MediatR, що дозволяє централізовано обробляти логіку без змішування її з контролерами;
- сервісний рівень (Service Layer): відповідає за виконання специфічної бізнес-логіки, яка не належить до CRUD-операцій, приклади: логіка зміни статусів замовлень, генерація прев'ю моделей, логіка перевірки відповідності матеріалів верстатам;
- рівень доступу до даних (Data Access Layer): реалізований через Entity Framework Core, включає конфігурацію сутностей, контекст даних, виконання запитів до бази даних, забезпечує взаємодію із реляційною базою даних PostgreSQL.

3.2.3 Зберігання файлів креслень

Файли креслень завантажуються користувачами у форматі DXF. На першому етапі вони зберігаються локально на сервері у захищеному каталозі.

У наступних версіях системи передбачається перенесення файлів у хмарне сховище (Azure Blob Storage або аналогічне рішення), що забезпечить масштабованість та стійкість до збоїв.

Доступ до файлів здійснюється через авторизовані API. Прямі посилання на файли не надаються, дані файлів завантажуються напряму, щоб забезпечити безпеку приватних даних.

3.2.4 Взаємодія між логікою всередині сервера і між сервером і клієнтом

У системі реалізовано чітко визначений механізм обміну даними між клієнтом та сервером на основі HTTP-запитів з JSON-представленням. Всі запити надсилаються з клієнтської частини Blazor WebAssembly через створений сервіс IHttpService, який реалізує логіку конструювання і обробки запитів, а відповіді очікуються у вигляді структурованої обгортки ServiceResponse.

Формат обміну - виклик API здійснюється методом POST, GET або PUT, залежно від контексту. При цьому передача параметрів відбувається у форматі JSON. Для завантаження файлів креслень використовується MultipartFormDataContent.

Аутентифікація та авторизація - кожен запит до API супроводжується JWT-токеном, який зберігається на стороні клієнта та додається до заголовків запиту. Сервер перевіряє підпис токена і роль користувача, щоб надати доступ до потрібного ресурсу або функціоналу.

Для стандартизації обміну між усіма рівнями системи (внутрішня логіка - API - клієнт) реалізована структура відповіді ServiceResponse, яка дозволяє відокремити результат від статусу та помилок.

```
public class ServiceResponse
{
      public bool IsSuccess { get; set; }
      public Error Error { get; set; } = new Error();
}
```

У разі, якщо потрібно повернути результат, використовується generic-варіант:

```
public class ServiceResponse<TResult> : ServiceResponse
{
      public TResult Result { get; set; }
}
```

Це дозволяє уніфікувати обробку всіх результатів, включно з помилками, і на клієнті, і на сервері.

Обробка на сервері (API) - у кожному контролері API реалізовано уніфіковану обробку ServiceResponse через допоміжні методи:

```
protected IActionResult ConvertFromServiceResponse(
      ServiceResponse serviceResponse)
{
      if (serviceResponse.IsSuccess)
            return Ok();

      return BadRequest(serviceResponse.Error);
}

protected IActionResult ConvertFromServiceResponse<T>(
      ServiceResponse<T> serviceResponse)
{
      if (serviceResponse.IsSuccess)
            return Ok(serviceResponse.Result);

      return BadRequest(serviceResponse.Error);
}
```

Це дозволяє API повертати єдину структуру помилок – об’єкт Error, який містить список ValidationErrors та ServiceErrors з читабельними повідомленнями:

```
public class Error
```

```
{
    public List<ValidationError> ValidationErrors { get; set; }
        = new();
    public List<ServiceError> ServiceErrors { get; set; } = new();
}
```

Завдяки цьому, на будь-якому рівні інтерфейсу можна обробити:

– чи був запит успішним;
– вивести повідомлення про помилку;
– відобразити помилки валідації полів.

Переваги підходу:

– централізоване оброблення результатів запитів;
– можливість формувати повноцінну історію помилок;
– гнучка передача повідомлень користувачам;
– легка відладка та тестування;
– сумісність з клієнтськими моделями Blazor.

3.2.5 Валідація запитів

У системі реалізовано централізований і розширюваний механізм валідації запитів та команд за допомогою бібліотеки FluentValidation, яка інтегрована в пайплайн MediatR через реалізований middleware валідації. Такий підхід дозволяє автоматично виконувати перевірку всіх вхідних даних до початку виконання бізнес-логіки, а також забезпечує єдиний формат повідомлень про помилки.

Middleware валідації реалізований через клас ValidationBehaviour<TRequest, TResponse>, який реалізує інтерфейс IPipelineBehavior і автоматично викликається MediatR перед обробкою будь-якої команди або запиту. У цьому класі реалізовано перевірку запиту на наявність валідатора, його виконання та формування відповідного ServiceResponse.

Логіка виконання валідації зосереджена у класі ValidationService, який через IServiceProvider динамічно знаходить відповідний IValidator<T> для об’єкта, що перевіряється:

```
public async Task<ServiceResponse> ValidateAsync<T>(T item)
{
      var validator = _serviceProvider.GetService<IValidator<T>>();
      if (validator is null)
            return ServiceResponseBuilder.Success();

      var validationResult = await validator
            .ValidateAsync(new ValidationContext<T>(item));

      var errors = validationResult.Errors
            .Select(error => new ValidationError
      {
            FieldCode = error.PropertyName,
            ErrorMessage = error.ErrorMessage
      }).ToList();

      return errors.Count > 0
            ? ServiceResponseBuilder.Failure(errors)
            : ServiceResponseBuilder.Success();
}
```

Це дозволяє:

– автоматично підключати будь-який валідатор без жорсткого зв’язування;
– повертати результат у вигляді зручної обгортки ServiceResponse, який вже інтегрований у загальну логіку API.

Формат помилок – валідаційні помилки повертаються у вигляді об'єкта Error, який містить список ValidationErrors. Кожна помилка має поле FieldCode і повідомлення для користувача:

```
public class ValidationError : IDisplayError
{
      public string FieldCode { get; set; }
      public string ErrorMessage { get; set; }
```

```
}
```

Переваги реалізації:

– гнучкість: легко додавати нові валідатори без зміни існуючої логіки;
– безпечність: жодна команда не буде оброблена, якщо дані некоректні;
– уніфікований API-вивід: однакова обробка помилок як у контролері, так і на клієнті;
– відсутність дублювання: валідація не розміщується в обробниках або контролерах.

3.3 Проєктування структури зберігання даних

3.3.1 Загальна модель даних

Модель даних системи побудована на основі реляційної структури з використанням принципів нормалізації. Вона забезпечує збереження інформації про замовлення, матеріали, верстати, листи розкрою, розміщення моделей та історію статусів.

Центральною сутністю є замовлення (Order), яке пов’язане з клієнтом, матеріалом, товщиною та файлом креслення. Замовлення можуть розміщуватись на аркушах (Sheet), які в свою чергу належать до конкретного верстата та матеріалу.

3.3.2 Опис основних сутностей

а) Customer – зберігає дані про замовника (ім’я, email, телефон), має зв’язок з замовленнями;
б) ServiceAdmin – описує адміністратора сервісу (ім’я, email, телефон);
в) Material – визначає матеріал (назва, тип), для якого вказується список доступних товщин (MaterialThickness), має зв’язок із верстатами через проміжну таблицю CuttingMachineMaterial;
г) MaterialThickness – вказує допустиму товщину для конкретного матеріалу;

д) CuttingMachine – описує верстат розкрою (назва, розміри робочого листа, опис), має список матеріалів, які підтримує (через CuttingMachineMaterial);

е) CuttingMachineMaterial – таблиця зв’язку «багато-до-багатьох» між верстатами та матеріалами;

ж) Order – сутність замовлення:

1) клієнт;
2) матеріал;
3) товщина;
4) кількість копій;
5) шлях до файлу DXF;
6) шлях до SVG-прев’ю;
7) статус;
8) історія змін статусу (OrderStatusHistory);

и) OrderStatusHistory – веде історію змін статусів замовлення із зазначенням часу;

к) Sheet – відображає лист розкрою, що містить розміщені моделі замовлень, має прив’язку до верстата, матеріалу та товщини. Містить фінальні файли для обробки (DXF, SVG), статус виконання;

л) SheetOrderPlacement – зв’язує моделі замовлень з листом розкрою, містить координати розміщення (TranslateX, TranslateY) та номер моделі (ModelNumber).

Для візуального представлення структури даних системи створено ER-діаграму, що відображає основні сутності, їх атрибути та зв’язки між ними. Це дозволяє чітко побачити логіку зберігання інформації та взаємозв’язок бізнес-об’єктів у системі.

Побудова ER-діаграми є важливим етапом у процесі проєктування, адже вона дозволяє узгодити модель даних з функціональними вимогами системи. Вона також є базою для створення схеми бази даних та реалізації взаємозв’язаних об’єктів у коді.

Сутності розроблялися на основі сценаріїв використання та передбачають масштабування і розширення в майбутніх релізах.

Результат побудови схеми зображений на рисунку 4.1.

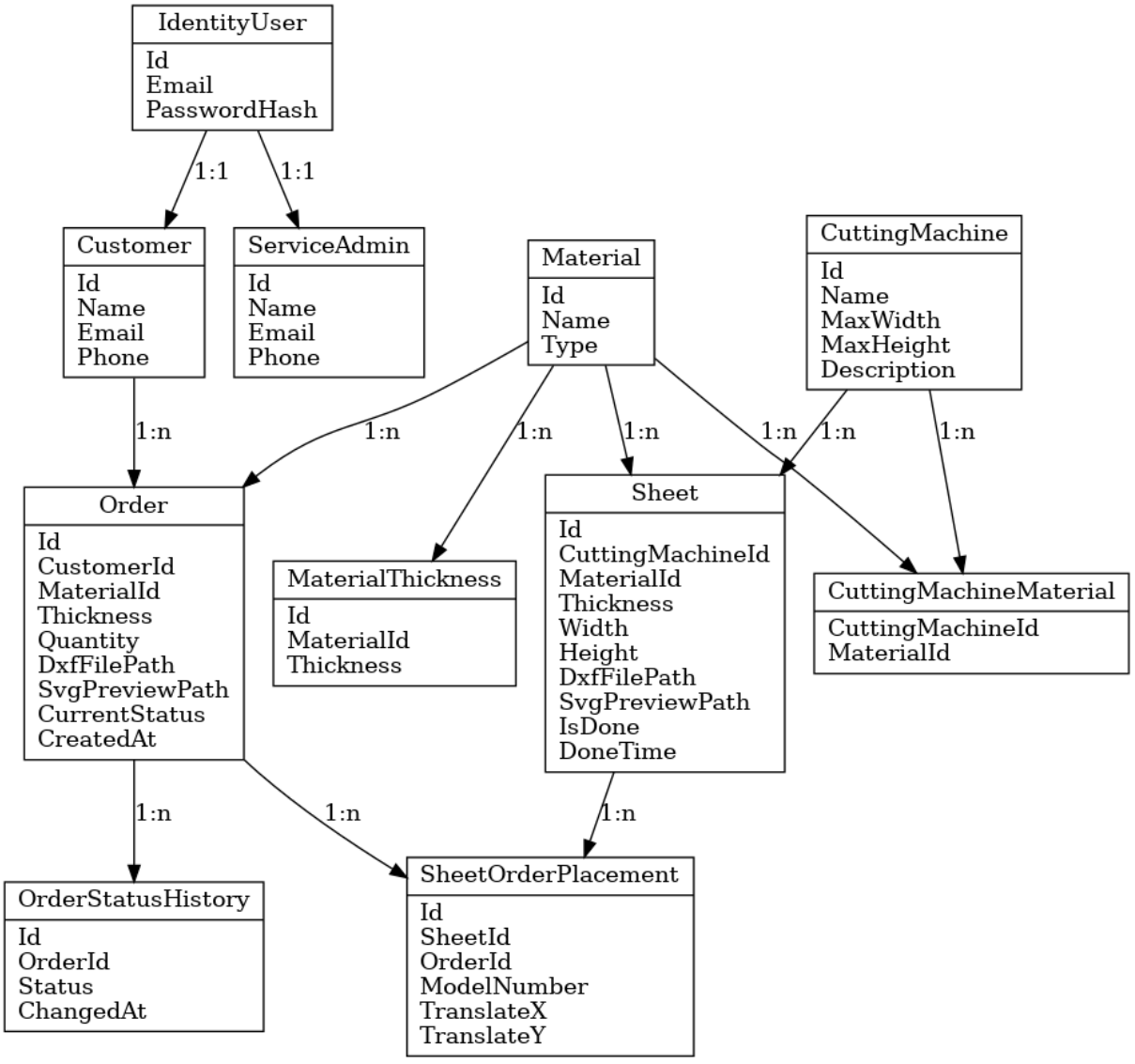

Рисунок 4.1 – ER-діаграма бази даних

Як видно з діаграми, всі сутності побудовані з урахуванням реальних бізнес-процесів, що дозволяє ефективно моделювати структуру даних та забезпечує масштабованість і зручність підтримки системи. Зв’язки між сутностями було сформовано відповідно до принципів нормалізації бази даних, що мінімізує надлишковість даних і підвищує цілісність інформації.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Логіка обробки DXF-файлів

Система підтримує завантаження креслень у форматі DXF, які містять векторну геометрію для подальшого розкрою. Завантажені файли є вихідною інформацією для планування розкрою та генерування виробничих планів.

Послідовність обробки файлу DXF:

1) отримання файлу: файл завантажується клієнтом через веб-інтерфейс;
2) зберігання: здійснюється у файлову систему сервера, шлях будується з унікальною назвою;
3) формування шляху: використовується відокремлений каталог для збереження файлів.

4.2 Конвертація DXF у SVG для перегляду

Оскільки формат DXF не підтримується безпосередньо у веб-браузерах для відображення, виникає необхідність конвертації у формат SVG для візуалізації моделі замовлення у інтерфейсі системи. Усі готові інструменти для читання і редагування dxf є платними і мають високу вартість ліцензій, тож виникла проблема створення своєї функціональності.

Метою функціональності конвертації стало побудувати коректне двовимірне векторне зображення креслення у форматі SVG на основі даних DXF. Для обробки DXF використовується безкоштовна бібліотека netDxf, яка дозволяє завантажити та обробити зміст файлу. Бібліотека дозволяє прочитати зміст та зробити просте редагування координат, але складніші зміни потрібно реалізовувати самостійно.

Алгоритм конвертації:

а) завантаження документа: DXF-документ завантажується у пам’ять для аналізу;

б) розрахунок габаритних розмірів (Bounding Box):

1) визначаються мінімальні та максимальні координати по осях X та Y;
2) враховуються всі геометричні примітиви (лінії, кола, полілінії, сплайни);

3) визначається розмір робочої області для SVG;

```
public static (Vector3 min, Vector3 max) GetBounds(DxfDocument dxf)
{
      var min = new Vector3(double.MaxValue, double.MaxValue, 0);
      var max = new Vector3(double.MinValue, double.MinValue, 0);

      foreach (var entity in dxf.Entities.All)
      {
            switch (entity)
            {
                  case Line line:
                        min         =         VectorHelper.MinVector(min,
                  line.StartPoint);
                        max         =         VectorHelper.MaxVector(max,
                  line.StartPoint);
                        min         =         VectorHelper.MinVector(min,
                  line.EndPoint);
                        max         =         VectorHelper.MaxVector(max,
                  line.EndPoint);
                        break;
                  case Circle circle:
                        ...
                  case Polyline2D poly:
                        ...
                  case Spline spline:
                        foreach (var pt in spline.ControlPoints)
                        {
                              min = VectorHelper.MinVector(min, pt);
                              max = VectorHelper.MaxVector(max, pt);
                        }
                        break;
            }
      }
      return (min, max);
}
```

в) масштабування та трансляція:

1) виконується нормалізація координат у площині відносно верхнього лівого кута;
2) оскільки осі координат у DXF і SVG мають різні напрямки по осі Y, виконується інверсія Y;

```
var bounds = DxfBoundsCalculator.GetBounds(dxf);
var min = bounds.min;
var max = bounds.max;
var translation = new Vector3(min.X, max.Y, 0);
```

г) генерація тегу <svg>:

1) формується атрибут viewBox з урахуванням габаритних розмірів;
2) встановлюється фактична ширина і висота з урахуванням масштабу;

```
double width = (max.X - min.X) * scale;
double height = (max.Y - min.Y) * scale;
var widthString = width.ToString(CultureInfo.InvariantCulture);
var heightString = height.ToString(CultureInfo.InvariantCulture);
var sb = new StringBuilder();
sb.AppendLine($"<svg ... width=\"{widthString}\"
height=\"{heightString}\" viewBox=\"0 0 {widthString}
{heightString}\" ... >");
```

д) конвертація геометричних примітивів у SVG елементи, інвертуючі Y координату:

1) лінії - <line>;
2) кола - <circle>;
3) полілінії - <polyline>;
4) сплайни - дискретизуються у полілінії (Polyline2D) з визначеним рівнем точності;

```
public static string ConvertToSvg(EntityObject entity, double scale =
1.0, Vector3 translation = default)
{
```

```
      var sb = new StringBuilder();

      switch (entity)
      {
            case Line line:
            ...
            case Circle circle:
            ...
            case Polyline2D poly:
            ...
            case Spline spline:
                  var convertedPoly = spline
            .ToPolyline2D(PolyLineConversionPrecision);
                  sb.Append("<polyline points=\"");
                  foreach (var v in convertedPoly.Vertexes)
                  {
                        var x = (v.Position.X - translation.X) * scale;
                        var  y  =  -(v.Position.Y  -  translation.Y)  *
                  scale;
                        sb.AppendFormat(CultureInfo.InvariantCulture,
                  "{0},{1} ", x, y);
                  }
                  sb.AppendLine("\" fill=\"none\" stroke=\"black\"/>");
                  break;
      }
      return sb.ToString();
}
```

е) формування кінцевого SVG:

1) побудова готового XML-документу у форматі SVG;

```
foreach (var entity in dxf.Entities.All)
{
      sb.Append(SvgEntityConverter.ConvertToSvg(entity,          scale,
translation));
}
sb.AppendLine("</svg>");
```

2) збереження результату у файлі з ідентичною назвою (заміна розширення DXF на SVG);
3) збереження у тій же директорії, що й оригінал DXF.

У результаті:

– генерується коректне векторне зображення креслення для відображення у редакторі;
– збереження геометричної точності та пропорцій, відповідно до вихідного файлу DXF;
– підготовлений SVG використовується для подальшого планування розкрою.

5.3 Редактор аркушів розкрою

Редактор аркушів розкрою є центральною частиною системи, яка дозволяє працівнику сервісу планувати розміщення моделей замовлень на листі верстата. Основна задача редактора - забезпечити зручне та точне формування плану розкрою з урахуванням матеріалу, товщини та габаритів листа.

Функціональні задачі редактора:

– розміщення моделей замовлень на листі з вказанням координат;
– збереження плану розкрою для подальшого виробництва;
– генерація актуального файлу розкрою (DXF) та прев’ю (SVG).

Алгоритм обробки розміщення моделей:

1. після розташування моделей на аркуші Sheet в клієнті передаються координати розташування кожної моделі на листі (Placement);
2. сервер отримує список розміщень та ідентифікатор аркуша;
3. система очищує попередні розміщення для цього аркуша і додає нові;
4. після збереження, для всіх замовлень на аркуші оновлюється статус на Processing;
5. далі запускається команда, яка фізично формує новий файл аркуша розкрою;

```
protected override async Task<ServiceResponse> PerformLogicAsync(
      SaveSheetLayoutCommand request,
      CancellationToken cancellationToken)
{
      var sheet = await _context.Sheets
            .Include(s => s.OrderPlacements)
            .FirstOrDefaultAsync(
                  s => s.Id == request.SheetId,
                  cancellationToken);

      if (sheet == null)
            return ServiceResponseBuilder
                  .Failure(ServerError.NotFound);

      sheet.OrderPlacements.Clear();

      foreach (var placement in request.Placements)
      {
            sheet.OrderPlacements.Add(new SheetOrderPlacement
            {
                  OrderId = placement.OrderId,
                  ModelNumber = placement.ModelNumber,
                  TranslateX = placement.TranslateX,
                  TranslateY = placement.TranslateY
            });
      }

      await _context.SaveChangesAsync(cancellationToken);

      await _orderStatusService.UpdateStatusesAsync(
            sheet.OrderPlacements,
            OrderStatus.Processing,
            CancellationToken.None);

      var updateResponse = await _mediator
            .Send(new UpdateSheetWithPlacementsCommand
      {
            SheetId = request.SheetId
      }, CancellationToken.None);
```

```
        return ServiceResponseBuilder.Success();
}
```

Логіка оновлення листа розкрою:

а) базовий DXF-файл листа завантажується в пам’ять;

б) для кожного розміщення виконується вставка моделі замовлення (Order DXF) у відповідне місце на аркуш;

в) вставка реалізується через клас створений DxfPlacer, який:

1) обчислює зміщення для вирівнювання моделі;
2) виконує трансляцію координат із урахуванням перевороту по осі Y;
3) додає всі об'єкти моделі до цільового документа (аркуша);

```
public static void Place(
        DxfDocument targetDoc,
        DxfDocument insertDoc,
        Vector3 location)
{
        var parentHeight = GetDocumentHeight(targetDoc);
        var (childMin, childMax) =
                DxfBoundsCalculator.GetBounds(insertDoc);
        var moveToOrigin = new Vector3(childMin.X, childMax.Y, 0);
        var flipY = new Vector3(0, parentHeight, 0);
        var finalOffset = new Vector3(location.X, -location.Y, 0);

        foreach (var entity in insertDoc.Entities.All)
        {
                var clone = (EntityObject)entity.Clone();
                var combinedTranslation = flipY - moveToOrigin
                        + finalOffset;
                DxfEntityTransformer.MoveEntity(clone,
                        combinedTranslation);
                targetDoc.Entities.Add(clone);
        }
}
```

```
private static double GetDocumentHeight(DxfDocument doc)
{
      var bounds = DxfBoundsCalculator.GetBounds(doc);
      return bounds.max.Y - bounds.min.Y;
}
```

г) переміщення геометричних об’єктів виконується через створений DxfEntityTransformer, який підтримує: лінії, кола, полілінії, сплайни, текстові об’єкти, цей клас методом MoveEntity(EntityObject entity, Vector3 translation) застосовує переміщення по вектору до всіх вкладених entity;

д) після вставки всіх моделей, аркуш зберігається у вигляді оновленого DXF-файлу;

е) додатково формується SVG-прев’ю для відображення результату у системі;

Результат:

– отриманий оновлений файл розкрою готовий для передачі у виробництво;
– усі зміни збережені у базі даних та відображаються у через актуальне SVG-прев’ю.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У процесі реалізації програмної системи було застосовано комбінований підхід до тестування функціональності, що охоплює як ручну перевірку користувацького інтерфейсу та графічних результатів, так і можливість юніт-тестування серверної логіки.

5.1 Ручне тестування UI та DXF-файлів

Візуальні частини системи, а саме веб-інтерфейс клієнта та редактор розкрою, тестуються вручну в процесі інтерактивної роботи в браузері. Тестування полягає у перевірці коректності:

- створення замовлень;
- вибору матеріалів і верстатів;
- розміщення моделей на аркуші;
- збереження результатів та відображення SVG-прев’ю.

Оскільки відображення та редагування DXF-файлів безпосередньо у браузері є обмеженим, для перевірки коректності сформованих файлів розкрою використовується стороннє ПЗ, зокрема онлайн-сервіси перегляду, як-от: https://allinpdf.com/dwg-viewer.

Подібні сервіси дозволяють візуально впевнитися, що згенерований файл DXF містить усі моделі у правильному масштабі та позиції.

5.2 Можливості юніт-тестування бізнес-логіки

Архітектура бекенду спроектована з урахуванням модульності та тестованості. Зокрема:

- кожен сервіс реалізовано у вигляді інтерфейсу та окремої реалізації;
- уся бізнес-логіка ізольована в MediatR-обробниках (Handlers), що працюють із запитами та командами;
- інфраструктура збереження даних не жорстко зв'язана із логікою.

Це дозволяє легко створювати юніт-тести, де залежності (наприклад, репозиторії, сервіси збереження, обробники файлів) замінюються на моки або

стаби. Таким чином, можна перевірити логіку переходів статусів, валідації, побудови аркуша, не залучаючи реальну базу даних або файлову систему.

Для реалізації юніт-тестів обрано фреймворк xUnit [14] – один із найпопулярніших інструментів для тестування у .NET-екосистемі. Його буде використано для:

- тестування сервісів (наприклад, логіки формування статусів замовлень);
- перевірки обробників команд/запитів (наприклад, SaveSheetLayoutCommand);
- валідаційних сценаріїв (через FluentValidation [15] у тестовому середовищі).

ВИСНОВКИ

У результаті виконаної роботи було розроблено веб орієнтовану програмну систему для обліку замовлень та планування розкрою листових матеріалів, що відповідає сучасним потребам сервісів з надання таких послуг.

На етапі аналізу предметної області було досліджено поточний стан ринку програмного забезпечення для розкрою. Визначено, що більшість існуючих рішень зосереджені на геометричній оптимізації розміщення моделей на листі, не враховуючи потребу в менеджменті замовлень, управлінні ресурсами та виробничими чергами. Виявлені бізнес-вимоги, потреби ринку та основні ризики дозволили сформувати обґрунтовану концепцію рішення.

Було поставлено задачу створення програмної системи, яка забезпечує:

- прийом замовлень із завантаженням креслень у форматі DXF;
- планування розкрою аркушів із урахуванням матеріалів, товщини та габаритів верстатів;
- управління чергами обробки;
- контроль життєвого циклу замовлень.

У рамках архітектурного проєктування побудовано клієнт-серверну модель системи, з використанням Blazor WebAssembly на клієнтській частині та ASP.NET Core Web API на сервері. Розроблено модульну структуру бізнес-логіки з використанням патернів CQRS та Mediator. Визначено структуру зберігання даних у реляційній базі PostgreSQL, розроблено сутності та їх зв’язки, представлено ER-діаграму та UML-діаграму випадків використання.

Особлива увага приділена реалізації логіки обробки DXF-файлів, конвертації у SVG для візуалізації у браузері та розробці редактора аркушів розкрою. Редактор дозволятиме працівнику вручну розміщувати моделі на листі, зберігати план розкрою та формувати актуальні файли для виробництва.

Таким чином, програмна система у рамках версії MVP повністю виконуватиме поставлені задачі, дозволяючи автоматизувати процес обліку замовлень та ручного планування розкрою для сервісів малого та середнього бізнесу. Архітектура системи забезпечуватиме можливість подальшого розвитку:

інтеграцію алгоритмів оптимізації розміщення, підключення CAM/ERP систем та розширення функціоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Madsen D. A. Engineering Drawing and Design. — Cengage Learning, 2011. — 1056 p.

2. Schoonmaker S. J. The CAD Guidebook. — CRC Press, 2002. — 312 p.

3. CutLogic 2D [Електронний ресурс]. – URL: https://www.tmachines.com/ (дата звернення: 27.05.2025).

4. NestFab [Електронний ресурс]. – URL: https://www.nestfab.com (дата звернення: 27.05.2025).

5. Cutting Optimization Pro [Електронний ресурс]. – URL: https://cutting-optimization.optimalprograms.com/ (дата звернення: 27.05.2025).

6. Smart2DCutting [Електронний ресурс]. – URL: https://www.rasterweq.com/ (дата звернення: 27.05.2025).

7. Autodesk. AutoCAD DXF Reference [Електронний ресурс]. – URL: https://help.autodesk.com/view/OARX/2024/ENU/?guid=GUID-235B22E0-A567-4CF6-92D3-38A2306D73F3 (дата звернення: 27.05.2025).

8. Microsoft Docs. ASP.NET Core Web API [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/ (дата звернення: 27.05.2025).

9. Microsoft Docs. Blazor WebAssembly [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-8.0 (дата звернення: 27.05.2025).

10. PostgreSQL. Version 15 Documentation [Електронний ресурс]. – URL: https://www.postgresql.org/docs/15/index.html (дата звернення: 27.05.2025).

11. Microsoft Learn. CQRS Pattern [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs (дата звернення: 27.05.2025).

12. Young G. CQRS Documents [Електронний ресурс]. – URL: https://cqrs.wordpress.com/documents/ (дата звернення: 27.05.2025).

13. Martin R. C. Clean Code: A Handbook of Agile Software Craftsmanship. — Prentice Hall, 2008. — 464 p.

14. xUnit.net. Unit Testing Tool for .NET [Електронний ресурс]. – URL:

https://xunit.net (дата звернення: 27.05.2025).

15. FluentValidation. Validation for .NET [Електронний ресурс]. – URL: https://docs.fluentvalidation.net/en/latest/ (дата звернення: 27.05.2025).

16. Посилання на GitHub, де розташовані всі електронні матеріали до кваліфікаційної роботи [Електронний ресурс]. – URL: https://github.com/Wednumi/2025_B_Sereda_M_I