Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 - «Інженерія програмного забезпечення»

На тему: Розробка програмного забезпечення оптимізації розкрою листових та рулонних матеріалів в умовах серійного виготовлення продукції

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-22-2
_____ / В.А. Пліско /

Керівник кваліфікаційної роботи _____ / А.М. Гриценко /

Завідувач кафедри _____ / А.М. Стрюк /

Кривий Ріг
2026

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Освітньо-кваліфікаційний рівень: бакалавр

Спеціальність: 121 - Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М. Стрюк

«___»_____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-22-2 Пліску Владиславу Анатолійовичу

1. Тема: Розробка програмного забезпечення оптимізації розкрою листових та рулонних матеріалів в умовах серійного виготовлення продукції затверджено наказом по КНУ № 285 від «28» травня 2026 р.

2. Термін подання студентом закінченого проекту «06» червня 2026 р.

3. Вихідні дані по роботі: *Пояснювальна записка: 117 сторінки, 42 рисунків, 2 додатки, 25 використаних у роботі джерел*

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Аналіз предметної області та постановка задачі оптимізації розкрою. Проектування програмного забезпечення оптимізації розкрою. Програмна реалізація DESCTOP-застосунку. Тестування та наліз результатів роботи програмного забезпечення.

5. Перелік графічного демонстраційного матеріалу: Класифікація програмних засобів оптимізації розкрою. Порівняння існуючих програмних засобів для оптимізації розкрою. Класифікація методів оптимізації розміщення деталей. Функціональна схема CutMap Designer. Архітектура CutMap Designer. Структура бази даних CutMap Designer. Інтерфейси CutMap Designer.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Формулювання мети та задач роботи	18.02.2026-20.02.2026
2	Аналіз інформаційних джерел	21.02.2026-09.03.2026
3	Визначення вимог до програмного забезпечення	10.03.2026-18.03.2026
4	Розробка функціональної схеми	19.03.2026-23.03.2026
5	Розробка архітектури та алгоритмів роботи	24.03.2026-31.03.2026
6	Розробка структури бази даних	01.04.2026-10.04.2026
7	Розробка програмного забезпечення	10.04.2026-13.05.2026
8	Тестування програмного забезпечення	14.05.2026-20.05.2026
9	Оформлення пояснювальної записки	21.05.2026-31.06.2026
10	Розробка демонстраційних матеріалів	01.06.2026-10.06.2026

Дата видачі завдання: « 18 » лютого 2026 р.

Студент _____ / В.А. Пліско /

Керівник роботи _____ / А.М. Гриценко /

РЕФЕРАТ

ОПТИМІЗАЦІЯ РОЗКРОЮ, ЛИСТОВІ МАТЕРІАЛИ, РУЛОННІ МАТЕРІАЛИ, КАРТА РОЗКРОЮ, C#, WPF, SQLITE, DXF, SVG

Пояснювальна записка: 117 с., 17 табл., 42 рис., 1 дод., 24 джерел.

Метою кваліфікаційної роботи є розробка програмного забезпечення CutMap Designer для оптимізації розкрою листових та рулонних матеріалів в умовах серійного виробництва з урахуванням геометрії деталей і технологічних параметрів обробки.

У роботі проведено аналіз предметної області, розглянуто особливості серійного виготовлення продукції, способи подання стандартних і складних деталей, а також виконано огляд існуючих програмних засобів оптимізації розкрою.

Для реалізації програмного забезпечення використано мову програмування C#, технологію WPF та базу даних SQLite. Передбачено підтримку експорту результатів у формати DXF та SVG.

У межах проєктування розроблено функціональну схему системи, архітектуру застосунку, структуру бази даних, модель подання геометрії деталей та алгоритм формування карти розкрою. Архітектура побудована на основі підходу MVVM.

Алгоритм оптимізації включає підготовку вхідних даних, формування геометричної моделі деталей, сортування, генерацію допустимих орієнтацій, пошук позицій розміщення, перевірку перетинів та розрахунок коефіцієнта використання матеріалу.

Розроблене програмне забезпечення забезпечує створення проєктів розкрою, роботу з довідниками матеріалів, інструментів і верстатів, введення деталей, формування карти розкрою, візуалізацію результатів та їх експорт. Проведене тестування підтвердило працездатність системи та коректність роботи алгоритму оптимізації.

ABSTRACT

CUTTING OPTIMIZATION, SHEET MATERIALS, ROLL MATERIALS, CUTTING LAYOUT, C#, WPF, SQLITE, DXF, SVG

Explanatory note: 117 pages, 17 tables, 42 figures, 1 appendix, 24 references.

The purpose of the qualification work is to develop the CutMap Designer software for optimizing the cutting of sheet and roll materials in serial production, taking into account part geometry and technological processing parameters.

The work analyzes the subject area, considers the features of serial production, methods of representing standard and complex-shaped parts, and reviews existing software tools for cutting optimization.

The software was implemented using the C# programming language, WPF technology, and the SQLite database. Support for exporting results in DXF and SVG formats is provided.

During the design stage, the functional scheme of the system, application architecture, database structure, model for representing part geometry, and cutting layout generation algorithm were developed. The architecture is based on the MVVM approach.

The optimization algorithm includes input data preparation, formation of the geometric model of parts, sorting, generation of allowable orientations, search for placement positions, intersection checking, and calculation of the material utilization coefficient.

The developed software provides project creation, work with material, tool, and machine directories, part input, cutting layout generation, visualization of results, and their export. The conducted testing confirmed the system’s operability and the correctness of the optimization algorithm.

ЗМІСТ

ВСТУП

Раціональне використання матеріальних ресурсів є важливим завданням серійного виробництва. Для підприємств, що працюють із листовими та рулонними матеріалами, ефективність розкрою безпосередньо впливає на обсяг відходів, собівартість продукції та тривалість технологічної підготовки виробництва [1, 2].

Особливої актуальності оптимізація розкрою набуває під час виготовлення великої кількості однакових або подібних деталей. Навіть незначне покращення їх розміщення може забезпечити помітний економічний ефект. При цьому необхідно враховувати не лише геометрію деталей, а й ширину різу, технологічні зазори, допуски, параметри інструменту, характеристики обладнання та обмеження орієнтації [1, 2].

Деталі можуть мати стандартну форму або складний контур, що ускладнює ручне формування карти розкрою. Застосування спеціалізованого програмного забезпечення дає змогу автоматизувати розміщення деталей, скоротити час підготовки виробничого завдання та підвищити ефективність використання матеріалу [2].

У межах кваліфікаційної роботи розробляється програмне забезпечення CutMap Designer для формування та оптимізації карт розкрою листових і рулонних матеріалів. Програма реалізується як desktop-застосунок мовою C# із використанням WPF, а для збереження матеріалів, інструментів, верстатів і конфігурацій проєктів застосовується локальна база даних SQLite.

Метою кваліфікаційної роботи є розробка програмного забезпечення CutMap Designer, яке забезпечує створення проєктів розкрою, введення або імпорт геометрії деталей, урахування технологічних параметрів, формування карти розкрою та експорт результатів у формати DXF і SVG.

Для досягнення поставленої мети необхідно:

а) проаналізувати предметну область розкрою листових і рулонних матеріалів;

б) розглянути способи подання стандартних і складних деталей;

в) дослідити існуючі програмні засоби та методи оптимізації розкрою;

г) визначити вимоги до програмного забезпечення;

д) спроєктувати архітектуру WPF-застосунку та структуру бази даних SQLite;

е) розробити алгоритм розміщення деталей з урахуванням ширини різу, зазорів, допусків та обмежень орієнтації;

ж) реалізувати введення, редагування й імпорт геометрії деталей;

и) реалізувати візуалізацію карти розкрою та експорт у формати DXF і SVG;

к) провести тестування програмного забезпечення.

Об’єктом розробки є процес підготовки карт розкрою листових і рулонних матеріалів у серійному виробництві.

Предметом розробки є програмне та алгоритмічне забезпечення оптимізованого розміщення деталей з урахуванням їх геометрії, технологічних параметрів матеріалу, інструменту й обладнання.

Практичне значення роботи полягає у створенні програмного засобу, який скорочує час підготовки проєкту розкрою, забезпечує повторне використання технологічних налаштувань і формує результати, придатні для подальшого використання у спеціалізованих програмах.

Програмне забезпечення може застосовуватися на виробничих підприємствах, у майстернях і проєктно-технологічних підрозділах під час розкрою металу, деревинних плит, пластику, тканини, плівки, паперу та інших листових або рулонних матеріалів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ОПТИМІЗАЦІЇ РОЗКРОЮ

1.1 Характеристика процесу розкрою листових та рулонних матеріалів

Розкрій матеріалів є важливим етапом технологічної підготовки виробництва. Він полягає у розміщенні заданої кількості деталей на вихідному матеріалі та формуванні карти розкрою, яка визначає положення, кількість, орієнтацію деталей і технологічні відстані між ними [1, 2].

Розкрій застосовується для металевих листів, деревинних плит, пластику, скла, тканини, паперу, плівки та інших матеріалів. Загальна задача полягає у розміщенні деталей із мінімальними втратами матеріалу та дотриманням установлених обмежень [1, 2].

Листовий матеріал має фіксовані довжину й ширину, тому деталі повинні повністю розміщуватися в межах заданої області. Для рулонного матеріалу зазвичай фіксованою є ширина, а довжина використаної частини змінюється. У цьому випадку одним із критеріїв оптимізації є мінімізація використаної довжини рулону [2].

Деталі можуть мати стандартну або складну форму. Прямокутники, кола та прості багатокутники задаються основними геометричними параметрами. Деталі складної форми подаються контуром, заданим координатами вершин або імпортованим із файлу. Під час їх розміщення необхідно враховувати фактичну форму, перевіряти перетини контурів і вихід за межі матеріалу [3].

На формування карти розкрою також впливають ширина різу, технологічні зазори, допуски, характеристики інструменту й обладнання. Окремо враховується орієнтація деталей, оскільки для матеріалів із напрямом волокон, текстурою або напрямом прокату їх поворот може бути обмежений.

В умовах серійного виробництва кожна деталь може мати значну кількість екземплярів. Тому програмне забезпечення повинно враховувати

кількість деталей, забезпечувати повторюваність результатів і підтримувати повторне використання технологічних налаштувань.

Ручне формування карти розкрою є трудомістким і не завжди забезпечує ефективне використання матеріалу. Автоматизація дає змогу перевіряти вхідні дані, враховувати технологічні обмеження, формувати різні варіанти розміщення та обирати кращий за коефіцієнтом використання матеріалу.

У межах роботи процес розкрою розглядається як автоматизоване розміщення стандартних і складних деталей на листовому або рулонному матеріалі. Результатом є карта розкрою, яка відображається у CutMap Designer та може бути експортована у формати DXF або SVG.

Таким чином, розкрій є комплексною задачею, що поєднує геометричне моделювання, технологічні обмеження та оптимізацію використання матеріалу.

1.2 Особливості серійного виготовлення продукції

Серійне виробництво передбачає виготовлення продукції партіями з повторенням технологічних операцій і використанням типових налаштувань обладнання [4]. У задачах розкрою це означає багаторазове виготовлення однакових або подібних деталей із повторним застосуванням параметрів матеріалу, інструменту, верстата та технологічних допусків.

Основними вимогами серійного виробництва є стабільність результатів, скорочення часу підготовки та економне використання матеріалу. Навіть незначна перевитрата матеріалу на одній карті розкрою за великої кількості виробів може суттєво збільшити загальні відходи та собівартість продукції. Тому оптимізація розміщення деталей має важливе практичне значення [1, 2].

Стабільність виготовлення забезпечується врахуванням ширини різу, технологічних зазорів, допусків, параметрів інструменту, точності обладнання та властивостей матеріалу. Збереження цих даних у базі дає змогу повторно використовувати типові конфігурації, скорочувати час підготовки проєкту та зменшувати ймовірність помилок.

Під час формування карти розкрою необхідно враховувати комплектність виробу. Якщо один виріб складається з кількох деталей, кількість кожного типу множиться на розмір партії. Тому програмне забезпечення повинно формувати відповідну кількість екземплярів деталей і забезпечувати їх раціональне розміщення.

Виробничі умови можуть змінюватися залежно від наявного матеріалу, обладнання або вимог замовника. Через це система повинна підтримувати як вибір збереженої конфігурації, так і створення нової шляхом введення необхідних параметрів.

Для листових матеріалів основним завданням є ефективне використання площі листа, а для рулонних — зменшення довжини використаної частини рулону [1, 2]. Під час роботи з деталями складної форми додатково враховуються фактичні контури, технологічні відступи та обмеження орієнтації [3].

Результати підготовки розкрою повинні зберігатися та передаватися до інших програмних засобів. У CutMap Designer для цього передбачено візуалізацію карти розкрою та її експорт у формати DXF і SVG.

Таким чином, програмне забезпечення для серійного виробництва повинно забезпечувати повторне використання налаштувань, підтримку значної кількості деталей, урахування технологічних параметрів, стабільність результатів і можливість експорту сформованої карти розкрою.

1.3 Технологічні параметри, що впливають на формування карти розкрою

Формування карти розкрою залежить не лише від геометрії деталей і матеріалу, але й від технологічних параметрів. Вони визначають допустимі відстані між деталями, можливість їх повороту, точність виготовлення та придатність карти для практичного використання.

Під час розкрою необхідно враховувати ширину різу, технологічні зазори, допуски, характеристики інструменту, робочу область верстата та

властивості матеріалу. Параметри процесу різання безпосередньо впливають на ширину різу та якість отриманих контурів [5]. Без їх урахування карта може бути непридатною для виробництва, навіть якщо деталі геометрично не перетинаються.

У CutMap Designer технологічні параметри входять до конфігурації проєкту. Користувач може вибрати збережені налаштування з бази даних SQLite або створити нову конфігурацію відповідно до матеріалу, інструменту, верстата та вимог до деталей.

Основні технологічні параметри, що впливають на формування карти розкрою, наведено в таблиці 1.1.

Таблиця 1.1 – Технологічні параметри, що впливають на формування карти розкрою

Параметр	Характеристика параметра	Вплив на формування карти розкрою
1	2	3
Тип матеріалу	Визначає вид матеріалу, який використовується для розкрою: метал, деревинна плита, фанера, пластик, тканина, плівка, папір тощо	Впливає на вибір інструменту, допустимі зазори, можливість повороту деталей та вимоги до орієнтації
Форма матеріалу	Матеріал може бути листовим або рулонним	Для листового матеріалу враховуються довжина і ширина заготовки, для рулонного — ширина рулону та довжина використаної ділянки
Габарити матеріалу	Довжина, ширина, товщина або робоча область матеріалу	Визначають межі, у яких можуть бути розміщені деталі
Товщина матеріалу	Фізична товщина заготовки	Впливає на вибір інструменту, режим різання, ширину різу та допустимі технологічні параметри

Продовження табл. 1.1

1	2	3
Ширина різу	Ширина матеріалу, яка видаляється інструментом під час обробки	Потребує збільшення відстані між деталями, щоб готові деталі мали необхідний розмір
Технологічний зазор	Мінімальна допустима відстань між сусідніми деталями	Запобігає перетину контурів деталей і забезпечує можливість проходження інструменту
Допуск на обробку	Допустиме відхилення фактичного розміру деталі від заданого	Враховується під час побудови контуру деталі та визначення відступів між деталями
Тип інструменту	Характеризує спосіб обробки: фреза, ніж, лазер, плазмовий різак, пильний диск тощо	Визначає ширину різу, мінімальний радіус обробки, швидкість і особливості формування траєкторії
Мінімальний радіус обробки	Найменший радіус, який може бути виконаний інструментом	Впливає на можливість обробки деталей складної форми та коректність їх контурів

Як видно з таблиці 1.1, ширина різу, технологічний зазор і допуски визначають мінімальну відстань між деталями. Тому під час розміщення враховується не лише фактичний контур, а й додаткова технологічна зона навколо нього.

Габарити матеріалу визначають межі робочої області. Для листового матеріалу деталі повинні повністю розміщуватися в межах листа, а для рулонного важливим критерієм є мінімізація довжини використаної частини [1, 2].

Параметри інструменту та верстата визначають ширину різу, мінімальний радіус обробки, точність позиціонування й допустиму робочу область. Якщо ці обмеження не враховано, сформована карта може бути непридатною для конкретного обладнання.

Окреме значення має орієнтація деталей. Для матеріалів без вираженого напряму поворот може покращити щільність розкладки. Для матеріалів із волокнами, текстурою, декоративним покриттям або напрямом прокату поворот може бути обмежений чи заборонений.

Для деталей складної форми необхідно враховувати фактичний контур, перевіряти його перетини, вихід за межі матеріалу та дотримання технологічного відступу [3]. У CutMap Designer такий контур може вводитися координатами вершин або імпортуватися з файлу.

Таким чином, технологічні параметри забезпечують відповідність геометричної розкладки реальним умовам виробництва. Їх урахування в CutMap Designer дає змогу формувати карти розкрою, придатні для подальшого технологічного використання.

1.4 Способи представлення стандартних деталей і деталей складної форми

Для формування карти розкрою геометрія деталей повинна бути подана у вигляді, придатному для програмної обробки. Модель деталі має містити дані про її форму, розміри, площу, кількість екземплярів, допустимість повороту та напрям орієнтації.

За способом подання деталі можна поділити на стандартні та складної форми. Стандартні фігури описуються основними геометричними параметрами: прямокутник — шириною і висотою, коло — радіусом або діаметром. Для них легко визначити площу, габаритну область і допустимі варіанти повороту.

Деталі складної форми мають виступи, вирізи або ламані контури, тому подаються послідовністю точок із координатами (X) та (Y). Порядок точок визначає послідовність побудови контуру. Контурне подання дає змогу виконувати перевірку перетинів, геометричні перетворення та визначення положення деталі на матеріалі [3, 6].

У CutMap Designer передбачено ручне введення координат вершин та імпорт геометрії з файлу. Це дозволяє працювати як із простими геометричними фігурами, так і з контурами, підготовленими в інших програмних середовищах. Приклади подання стандартних і складних деталей наведено на рисунку 1.1.

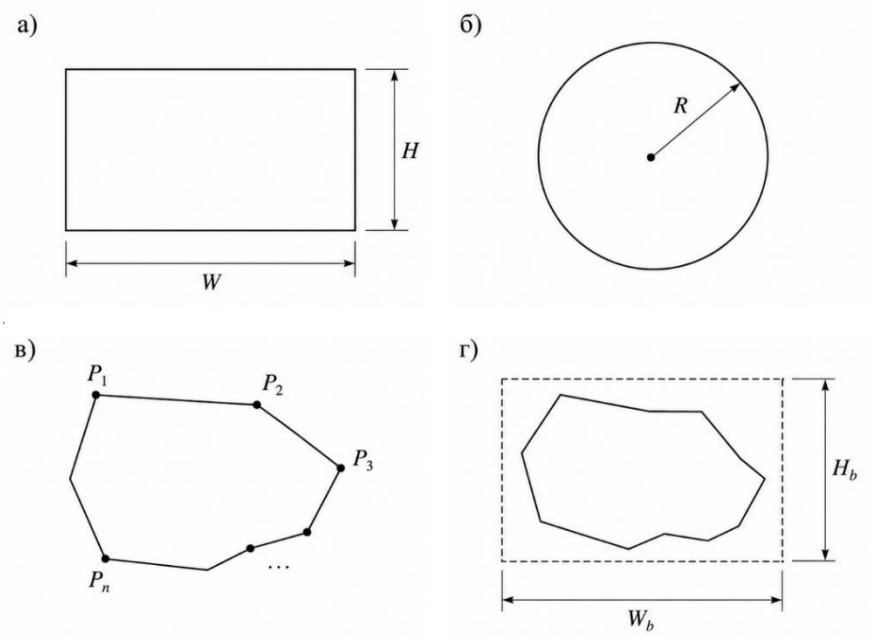

Рисунок 1.1 – Приклади представлення деталей різної форми а) - прямокутну деталь, що задаєтьсс параметрами W та H; б) - кругла деталь, що задається радіусом R; в) - складну деталь у вигляді багатокутного контуру, заданого координатами вершин P1, P2, P3, ..., Pn; г) - приклад габаритного прямокутника, який охоплює складний контур деталі.

Основні способи представлення деталей у програмному забезпеченні наведено в таблиці 1.2. Для стандартних фігур використовується параметричний опис, а для складних деталей — контурне подання, яке точніше відтворює їх фактичну геометрію.

Для внутрішньої обробки всі деталі приводяться до уніфікованої моделі. Наприклад, прямокутник може бути поданий чотирма вершинами, а складний контур — довільною кількістю впорядкованих точок. Такий підхід дозволяє

застосовувати спільні процедури перевірки меж, перетинів, повороту та переміщення [6].

Таблиця 1.2 – Способи представлення стандартних деталей і деталей складної форми

Спосіб представлення	Тип деталей	Дані, що задаються	Переваги	Обмеження
Представлення за габаритами	Прямокутні та прості стандартні деталі	Ширина, висота, кількість, дозвіл на поворот	Простота введення та обробки, висока швидкість розміщення	Не підходить для точного опису складних контурів
Представлення за параметрами фігури	Кола, овали, прості геометричні фігури	Радіус, діаметр, ширина, висота, тип фігури	Зручне для стандартних деталей, не потребує великої кількості даних	Обмежене набором підтримуваних типів фігур
Представлення за координатами вершин	Багатокутні та складні деталі	Послідовність точок контуру з координатами X та Y	Дозволяє описувати складні форми, придатне для перевірки перетинів контурів	Потребує точного введення координат і складнішої обробки
Імпорт геометрії з файлу	Деталі, підготовлені в інших програмах	Контури, лінії або полігони з файлу DXF чи SVG	Зменшує ручне введення, дозволяє використовувати готові креслення	Потребує коректного зчитування та перевірки імпортованих даних
Представлення через габаритний прямокутник складної деталі	Складні деталі на етапі попереднього розміщення	Мінімальна та максимальна координата контуру по X і Y	Спрощує первинну оцінку можливості розміщення	Не враховує реальну форму контуру та може давати менш щільну розкладку

У моделі деталі також зберігаються її кількість і допустимі орієнтації. Якщо поворот дозволений, алгоритм може розглядати кути 0°, 90°, 180° і 270°. Для матеріалів із напрямом волокон, текстурою або конструктивними обмеженнями використовуються лише дозволені орієнтації.

Під час розміщення складної деталі координати її вершин перераховуються відповідно до заданого повороту та положення. Для дотримання ширини різу, зазорів і допусків навколо основного контуру може формуватися розширений технологічний контур.

У програмній реалізації сутність Part містить назву, тип форми, кількість, площу, габарити, параметри повороту та шлях до імпортованого файлу. Координати вершин складного контуру зберігаються у сутності PartPoint. Основні елементи цієї структури наведено в таблиці 1.3.

Таблиця 1.3 – Основні дані представлення деталі

Дані	Призначення
Назва деталі	Використовується для ідентифікації деталі в проєкті
Тип форми	Визначає спосіб подання деталі: прямокутник, коло, багатокутник, імпортований контур
Кількість	Визначає кількість екземплярів деталі для розміщення
Ширина та висота	Використовуються для стандартних деталей і габаритної оцінки
Площа	Використовується для сортування деталей і розрахунку коефіцієнта використання матеріалу
Координати вершин	Описують контур складної деталі
Дозвіл на поворот	Визначає, чи може алгоритм змінювати орієнтацію деталі
Кут орієнтації	Задає обов’язковий або початковий напрям розміщення
Джерело геометрії	Вказує, чи була деталь створена вручну або імпортована з файлу

Таким чином, у CutMap Designer використовується комбінована модель, яка поєднує параметричний опис стандартних деталей, координатне подання складних контурів та імпорт геометрії. Це забезпечує універсальність системи й коректну підготовку деталей до оптимізації, візуалізації та експорту.

1.5 Огляд існуючих програмних засобів для оптимізації розкрою

Для обґрунтування доцільності розробки CutMap Designer необхідно розглянути наявні програмні засоби оптимізації розкрою. Вони відрізняються підтримуваними матеріалами й типами деталей, урахуванням технологічних параметрів, засобами експорту та рівнем інтеграції з виробничим обладнанням.

Існуючі рішення можна умовно поділити на промислові CAD/CAM-системи, програми прямокутного розкрою, системи true shape nesting, онлайн-сервіси та відкриті програмні засоби. Їх узагальнену класифікацію подано на рисунку 1.2.

Рисунок 1.2 – Класифікація програмних засобів оптимізації розкрою

Промислові CAD/CAM-системи забезпечують формування карт розкрою, імпорт креслень, урахування параметрів обладнання та підготовку керуючих даних. Одним із представників цієї групи є SigmaNEST, що

підтримує імпорт деталей із CAD-систем, їх автоматичне групування за матеріалом і верстатом, оптимізацію використання матеріалу та підготовку програм для різального обладнання [7]. Приклад інтерфейсу системи наведено на рисунку 1.3.

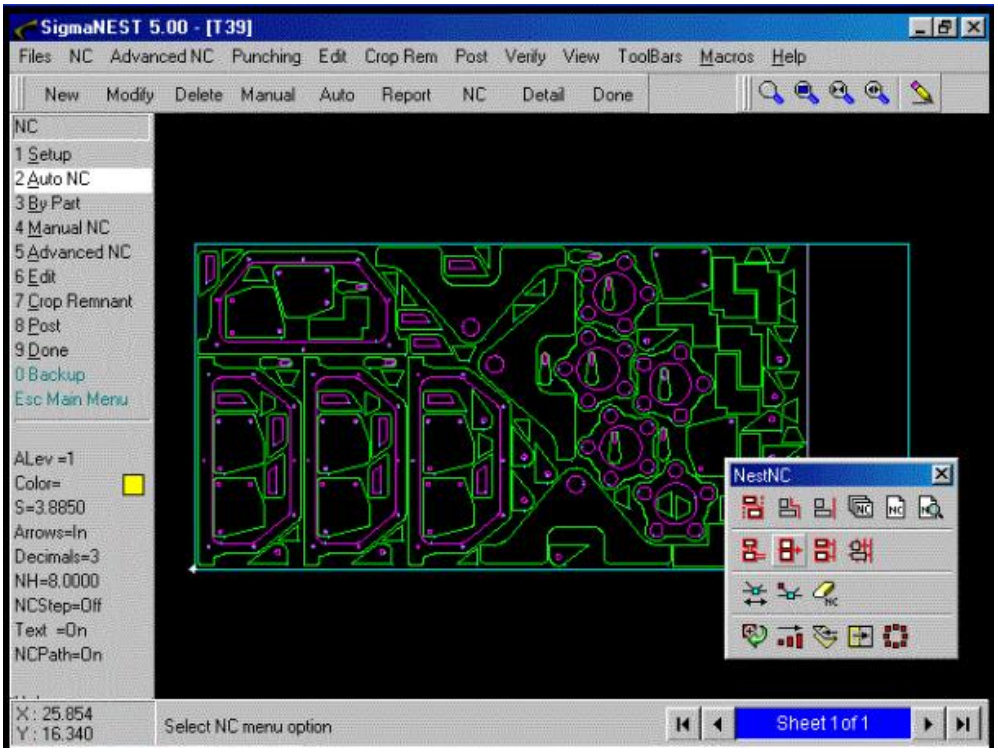

Рисунок 1.3 – Інтерфейс програмного засобу SigmaNEST

Як видно з рисунка 1.3, SigmaNEST орієнтована на професійне використання у виробничому середовищі. Її перевагою є широка функціональність, однак для невеликих виробництв або окремих задач така система може бути надмірно складною.

Ще одним прикладом промислової системи є ProNest, яка використовується для підготовки розкрою при плазмовому, лазерному, газовому та інших типах різання. Програма дозволяє імпортувати креслення деталей, налаштовувати параметри обладнання, формувати карти розкрою та готувати дані для подальшої обробки. Приклад інтерфейсу ProNest наведено на рисунку 1.4.

SigmaNEST орієнтована на професійне виробниче застосування та має широку функціональність, яка для невеликих підприємств або локальних задач може бути надмірною.

Іншим прикладом є ProNest — CAD/CAM-система для автоматизованого плазмового, лазерного, газового та інших видів різання. Вона підтримує імпорт креслень, автоматичне й ручне розміщення деталей, налаштування технологічних параметрів і підготовку даних для обладнання [8]. Інтерфейс ProNest наведено на рисунку 1.4.

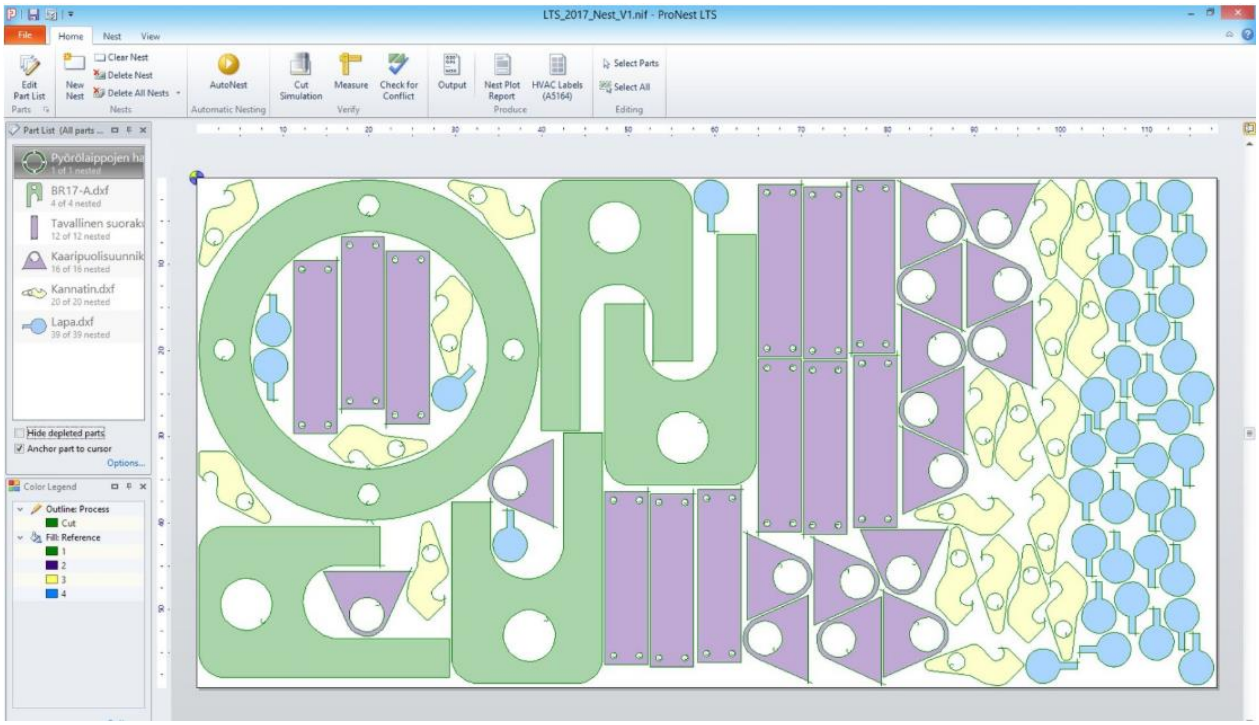

Рисунок 1.4 – Інтерфейс програмного засобу ProNest

Окрему групу становлять програми прямокутного розкрою, які використовуються переважно для деревинних плит, скла, пластику та інших листових матеріалів. Прикладом є CutList Optimizer, що формує оптимізовані схеми розкрою на основі заданих розмірів листів і прямокутних деталей [9]. Приклад інтерфейсу програми наведено на рисунку 1.5. Перевагою такого рішення є простота використання, проте його можливості обмежені деталями прямокутної форми.

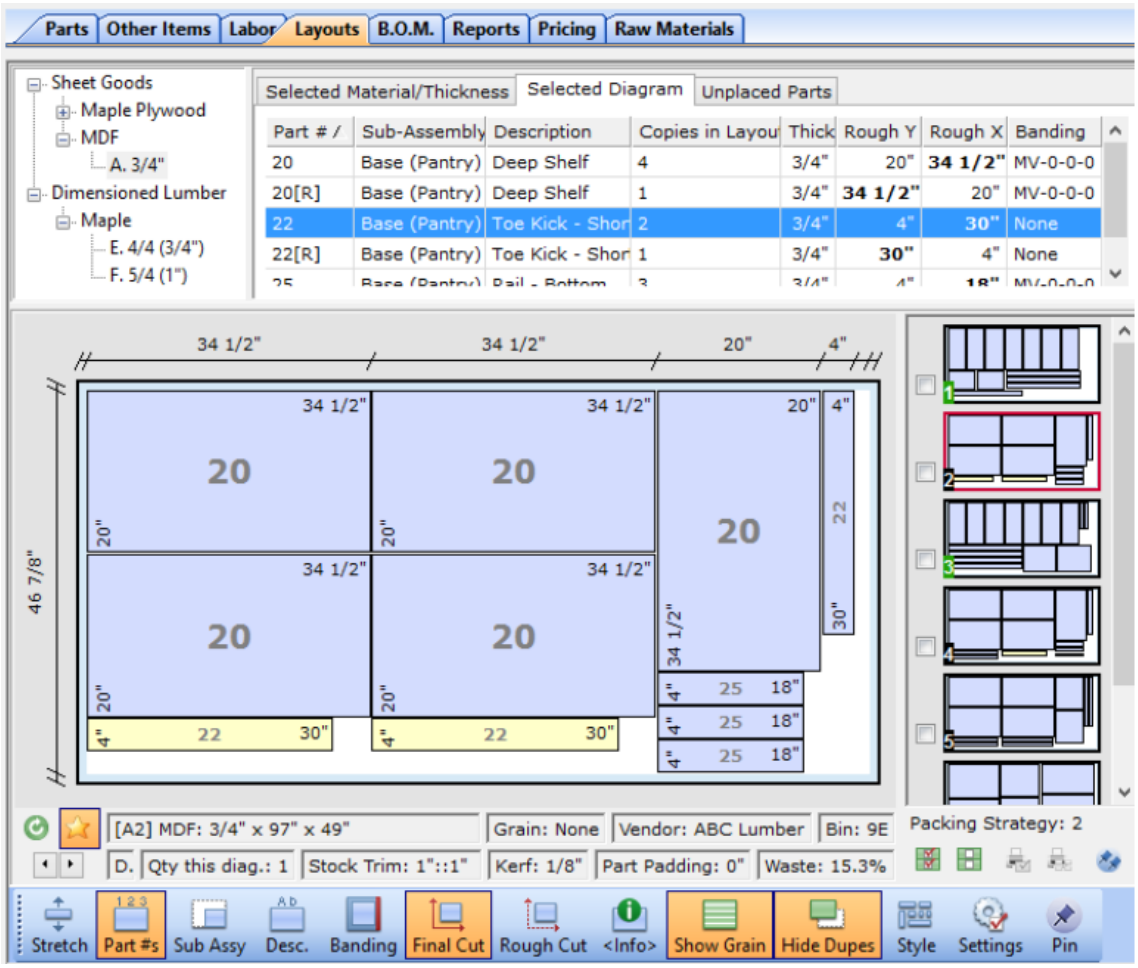

Рисунок 1.5 – Інтерфейс програмного засобу CutList Optimizer

Для розміщення деталей складної форми застосовуються системи true shape nesting. До них належить Deepnest — відкрита desktop-програма для розміщення складних контурів, орієнтована на лазерне, плазмове та інше CNC-різання. Вона підтримує DXF-файли та оптимізує взаємне розташування нерегулярних деталей [10]. Інтерфейс Deepnest наведено на рисунку 1.6.

Перевагою Deepnest є підтримка складної геометрії та відкритість програмного рішення. Водночас система не має розвинених засобів керування довідниками матеріалів, інструментів, верстатів і повторно використовуваних конфігурацій проєктів.

Онлайн-сервіси зручні для швидких розрахунків, але не завжди забезпечують локальне збереження даних, гнучке налаштування технологічних параметрів і повноцінну роботу зі складними контурами.

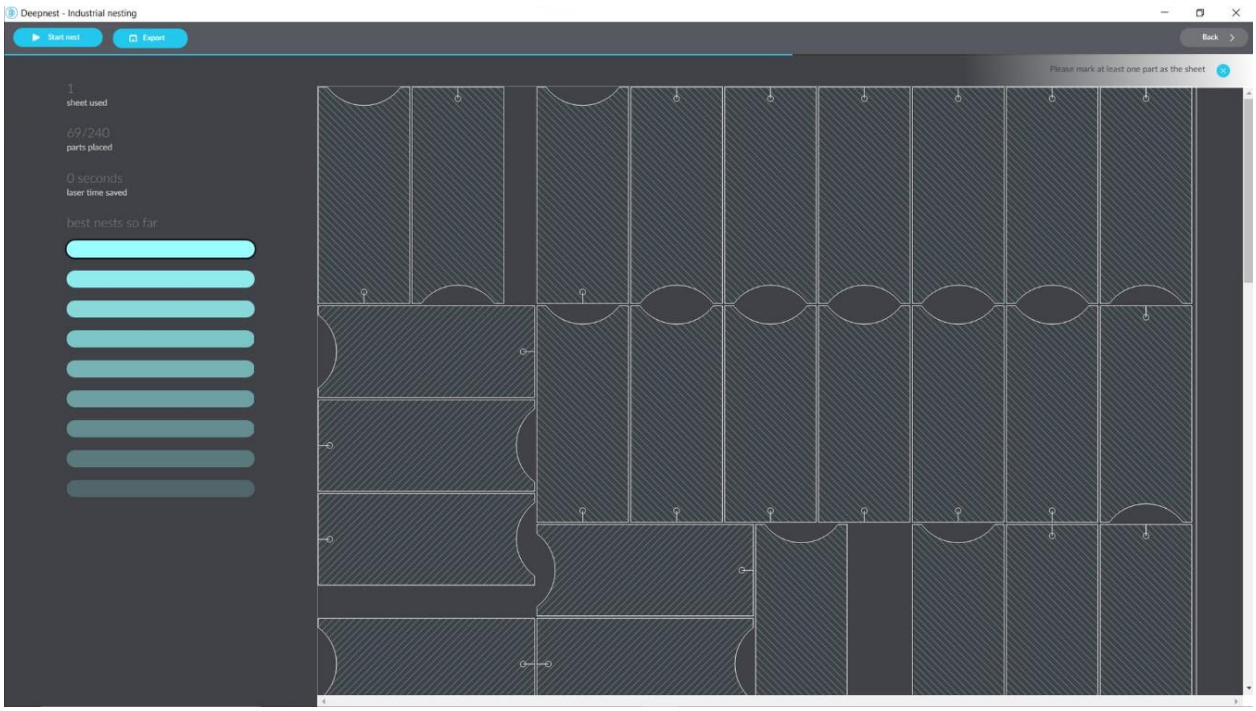

Рисунок 1.6 – Інтерфейс програмного засобу Deepnest

Порівняння розглянутих програмних засобів наведено в таблиці 1.4. Промислові CAD/CAM-системи підтримують складні виробничі процеси, але є складнішими у впровадженні. Програми прямокутного розкрою прості у використанні, проте мають обмеження щодо геометрії деталей. Відкриті nesting-рішення працюють зі складними контурами, але не завжди підтримують керування проєктами й технологічними довідниками.

Таблиця 1.4 – Порівняння існуючих програмних засобів для оптимізації розкрою

Програмний засіб	Основне призначення	Підтримка складних контурів	Робота з технологічними параметрами	Експорт результатів	Особливості
1	2	3	4	5	6
SigmaNEST	Промислова CAD/CAM-система для розкрою	Так	Так	Так	Висока функціональність, інтеграція з виробництвом

Продовження табл. 1.4

1	2	3	4	5	6
ProNest	Підготовка розкрою для різних типів різання	Так	Так	Так	Орієнтація на роботу з виробничим обладнанням
Astra R-Nesting	Прямокутний розкрій листових матеріалів	Обмежено	Частково	Так	Ефективна для простих деталей
MaxCut	Формування карт розкрою панельних матеріалів	Обмежено	Частково	Так	Зручна для меблевого виробництва
CutList Optimizer	Оптимізація панельного розкрою	Ні	Обмежено	Частково	Простий інтерфейс, швидке формування схеми
Cutting Optimization Pro	Одновимірний і двовимірний розкрій	Обмежено	Частково	Так	Універсальність для прямокутних деталей
Deepnest	Автоматичне nesting-розміщення контурів	Так	Обмежено	Так	Підтримка складних форм
SVGnest	Автоматичне розміщення SVG-контурів	Так	Обмежено	SVG	Орієнтація на контурні деталі

Отже, розробка CutMap Designer є доцільною. Система повинна поєднувати зручність desktop-застосунку, локальне збереження конфігурацій, підтримку стандартних і складних деталей, урахування технологічних параметрів та експорт результатів у формати DXF і SVG.

1.6 Аналіз методів оптимізації розміщення деталей

Оптимізація розміщення деталей є центральною задачею формування карти розкрою. Вона полягає у виборі такого розташування деталей на листовому або рулонному матеріалі, за якого забезпечується виготовлення

потрібної кількості елементів із мінімальними втратами та дотриманням технологічних обмежень [1, 2].

Задачі розкрою належать до складних комбінаторних задач, оскільки кількість можливих варіантів швидко зростає зі збільшенням кількості деталей і допустимих орієнтацій. Для деталей складної форми додатково необхідно враховувати фактичні контури та перевіряти їх взаємне перетинання [3, 6].

Методи оптимізації розміщення деталей поділяють на точні, евристичні, метаевристичні та комбіновані. Точні методи спрямовані на знаходження оптимального розв’язку, але потребують значних обчислювальних ресурсів. Евристичні методи швидко формують практично придатний результат, метаевристичні забезпечують ітераційне покращення розкладки, а комбіновані поєднують переваги кількох підходів [11–13].

Узагальнену класифікацію методів оптимізації розміщення деталей наведено на рисунку 1.7. До точних методів належать повний перебір і математичне програмування; до евристичних — жадібні алгоритми, First Fit, Best Fit і Bottom-Left; до метаевристичних — генетичні алгоритми, імітація відпалу, алгоритми мурашиної колонії та рою частинок. Комбіновані методи поєднують конструктивні евристики з процедурами локального або глобального покращення [11–13].

Повний перебір передбачає перевірку всіх можливих позицій та орієнтацій деталей. Він може забезпечити оптимальний результат для малих задач, однак є непридатним для великих наборів даних через швидке зростання кількості комбінацій. Методи математичного програмування описують задачу через змінні, обмеження та цільову функцію, але їх реалізація для складних контурів і довільних поворотів є обчислювально складною [11].

У прикладних системах частіше використовуються евристичні методи. Деталі попередньо сортуються за площею або габаритами, після чого послідовно розміщуються на матеріалі. Підходи First Fit, Best Fit і Bottom-Left мають порівняно просту реалізацію та забезпечують прийнятні результати за невеликий час [11, 12].

Рисунок 1.7 – Класифікація методів оптимізації розміщення деталей

Для деталей складної форми необхідно перевіряти вихід за межі матеріалу та перетин контурів. Щоб скоротити кількість обчислень, спочатку аналізуються габаритні прямокутники. Якщо вони перетинаються, виконується точніша перевірка фактичної геометрії деталей [3, 6].

Метаевристичні методи застосовуються для покращення розв’язків у задачах із великим простором пошуку. Генетичні алгоритми, імітація відпалу та інші підходи можуть забезпечувати якісніші розкладки, однак потребують вибору параметрів і більшого часу обчислення [12]. Комбіновані методи дають змогу поєднати швидке формування початкового розв’язку з його подальшим покращенням [13].

Для CutMap Designer доцільно використати комбінований евристичний підхід, який передбачає підготовку даних, сортування деталей, генерацію допустимих орієнтацій, пошук позицій, перевірку перетинів і вибір кращого варіанта. Порівняльну характеристику основних методів наведено в таблиці 1.5, а блок-схему запропонованого алгоритму — на рисунку 1.8.

Основним критерієм якості розкладки є коефіцієнт використання матеріалу:

$$K = S_д / S_м \cdot 100\ \%,$$

де K — коефіцієнт використання матеріалу, $S_д$ — сумарна площа розміщених деталей, $S_м$ — площа матеріалу або використаної області матеріалу.

Таблиця 1.5 – Порівняльна характеристика методів оптимізації розміщення деталей

Метод	Сутність методу	Переваги	Недоліки
Повний перебір	Перевірка всіх можливих варіантів розміщення деталей	Може знайти оптимальний розв'язок для малих задач	Дуже висока обчислювальна складність
Математич-не програму-вання	Формалізація задачі через змінні, обмеження та цільову функцію	Теоретична точність, чітка постановка задачі	Складна реалізація для деталей складної форми
First Fit	Розміщення деталі у першій допустимій позиції	Простота реалізації, висока швидкість	Не завжди забезпечує ефективне використання матеріалу
Best Fit	Вибір позиції, що дає найкращий локальний результат	Краща якість розкладки порівняно з First Fit	Потребує перевірки більшої кількості позицій

Кращим вважається варіант із більшим коефіцієнтом використання матеріалу, меншою площею відходів і мінімальною кількістю нерозміщених деталей. Додатково можуть ураховуватися кількість листів, довжина використаної частини рулону та складність траєкторії різання.

Для листового матеріалу деталі розміщуються в межах фіксованої прямокутної області. Якщо одного листа недостатньо, система може сформувати кілька карт розкрою. Для рулонного матеріалу основним обмеженням є ширина, а критерієм оптимізації — мінімізація використаної довжини [1, 2].

Алгоритм також повинен ураховувати допустимі орієнтації деталей. Якщо поворот дозволений, можуть перевірятися кути 0°, 90°, 180° і 270°. За наявності напрямку волокон, текстури або інших технологічних обмежень розглядаються лише дозволені орієнтації.

Таким чином, комбінований евристичний алгоритм є найбільш доцільним для CutMap Designer, оскільки поєднує прийнятну швидкість роботи з достатньою якістю розміщення. Він охоплює сортування деталей, генерацію орієнтацій, послідовне розміщення, перевірку обмежень та вибір кращої карти за коефіцієнтом використання матеріалу.

1.7 Постановка задачі кваліфікаційної роботи

На основі аналізу предметної області, технологічних параметрів, способів подання деталей і наявних програмних засобів сформульовано задачу кваліфікаційної роботи. Вона полягає у розробці програмного забезпечення для автоматизованого формування та оптимізації карт розкрою листових і рулонних матеріалів.

Розроблюваний desktop-застосунок CutMap Designer повинен забезпечувати створення проєктів розкрою, налаштування технологічних параметрів, введення або імпорт геометрії деталей, формування карти розкрою, візуалізацію та експорт результатів у формати DXF і SVG.

Система має підтримувати листові та рулонні матеріали. Для листового матеріалу враховуються його габарити й межі робочої області, а для рулонного — фіксована ширина та довжина використаної частини. Розміщені деталі не повинні виходити за межі матеріалу або перетинатися між собою [1–3].

Програмне забезпечення повинно працювати зі стандартними та складними деталями. Стандартні форми задаються геометричними параметрами, а складні — координатами вершин контуру або імпортуються з файлу. Контурне подання дає змогу виконувати перевірку меж, перетинів, поворотів і переміщень деталей [3, 6].

Конфігурація проєкту повинна містити параметри матеріалу, верстата й інструменту, ширину різу, технологічні зазори, допуски, правила орієнтації та формат експорту. Збереження конфігурацій у базі SQLite забезпечує їх повторне використання під час підготовки серійних виробничих завдань.

Основною розрахунковою частиною системи є алгоритм оптимізованого розміщення деталей. Він повинен враховувати їх кількість, геометрію, допустимі орієнтації, технологічні відступи та межі матеріалу. Для складних контурів необхідно передбачити перевірку перетинів, а основним критерієм оцінювання карти доцільно використати коефіцієнт використання матеріалу [11–13].

Для реалізації системи обрано мову програмування C#, технологію WPF та локальну базу даних SQLite. Такий набір технологій дозволяє створити автономний desktop-застосунок із графічним інтерфейсом і локальним збереженням довідкових, проєктних та розрахункових даних.

Результатом роботи програми є карта розкрою з координатами й орієнтацією деталей, показниками використання матеріалу та інформацією про залишки. Сформована карта повинна відображатися в інтерфейсі та експортуватися у формати DXF або SVG.

Для досягнення поставленої мети необхідно:

а) проаналізувати предметну область і технологічні параметри розкрою;

б) розглянути способи подання стандартних і складних деталей;

в) проаналізувати існуючі програмні засоби та методи оптимізації;

г) визначити функціональні й нефункціональні вимоги;

д) спроєктувати архітектуру WPF-застосунку та структуру бази SQLite;

е) розробити алгоритм формування карти розкрою з урахуванням геометрії деталей, зазорів, допусків, ширини різу й обмежень орієнтації;

ж) реалізувати введення, редагування та імпорт деталей;

и) реалізувати візуалізацію й експорт результатів у формати DXF і SVG;

к) провести тестування розробленого програмного забезпечення.

Об’єктом розробки є процес підготовки карт розкрою листових і рулонних матеріалів у серійному виробництві.

Предметом розробки є програмне та алгоритмічне забезпечення оптимізованого розміщення стандартних і складних деталей з урахуванням технологічних параметрів, характеристик обладнання та виробничих обмежень.

Вхіднии даними системи є параметри матеріалу, верстата й інструменту, зазори, допуски, список і кількість деталей, їх геометрія та допустимі орієнтації. Вихідними даними є карта розкрою, показники використання матеріалу, відомості про залишки та файл результату у форматі DXF або SVG.

Таким чином, задача роботи полягає у створенні CutMap Designer, що поєднує керування проєктами, збереження технологічних конфігурацій, подання геометрії деталей, оптимізоване формування карти розкрою, візуалізацію та експорт результатів.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ОПТИМІЗАЦІЇ РОЗКРОЮ

2.1 Визначення вимог до програмного забезпечення

Проєктування програмного забезпечення починається з визначення вимог, які встановлюють призначення системи, її функції, обмеження та умови використання [14]. Для CutMap Designer вони формуються з урахуванням оптимізації розкрою листових і рулонних матеріалів, підтримки стандартних і складних деталей та збереження технологічних параметрів проєкту.

Програмне забезпечення реалізується як desktop-застосунок мовою C# із використанням WPF. Ця технологія призначена для створення графічних застосунків Windows і підтримує форми введення, таблиці, графічні елементи та прив’язування даних [16]. Для локального збереження матеріалів, верстатів, інструментів, конфігурацій і результатів використовується SQLite — автономна безсерверна система керування базами даних [17].

Основним призначенням CutMap Designer є автоматизація підготовки карти розкрою. Система повинна забезпечувати створення проєктів, налаштування параметрів матеріалу, введення деталей, формування та перегляд карти розкрою, а також експорт результатів у формати DXF і SVG.

Вимоги до програмного забезпечення поділяються на функціональні та нефункціональні. Функціональні вимоги визначають дії системи, а нефункціональні — характеристики якості її роботи, зокрема зручність використання, надійність, продуктивність і супроводжуваність [14, 15]. Функціональні вимоги до CutMap Designer наведено в таблиці 2.1.

Функціональні вимоги охоплюють повний процес роботи користувача: створення проєкту, вибір або формування конфігурації, роботу з довідниками, введення деталей, запуск оптимізації, перегляд карти та експорт результатів. Повторне використання конфігурацій скорочує час підготовки типових виробничих завдань.

Таблиця 2.1 – Функціональні вимоги до програмного забезпечення CutMap Designer

№	Вимога	Опис вимоги
1	2	3
1	Створення проєкту розкрою	Система повинна дозволяти створювати новий проєкт розкрою із зазначенням назви, дати створення та основних параметрів
2	Вибір конфігурації проєкту	Користувач повинен мати можливість вибрати раніше збережену конфігурацію або створити нову
3	Робота з матеріалами	Програма повинна дозволяти додавати, редагувати, видаляти та вибирати матеріали з бази даних
4	Робота з верстатами	Система повинна зберігати параметри верстатів, зокрема тип обладнання, робочу область та технологічні обмеження
5	Робота з інструментами	Програма повинна зберігати параметри інструментів, зокрема ширину різу, мінімальний радіус обробки та рекомендовані зазори
6	Введення стандартних деталей	Користувач повинен мати можливість вводити деталі простої форми за основними геометричними параметрами
7	Введення складних деталей	Система повинна підтримувати задання складних деталей за координатами вершин контуру
8	Імпорт геометрії деталей	Програма повинна забезпечувати можливість відкриття геометрії деталей з файлу
9	Задання кількості деталей	Для кожної деталі повинна задаватися кількість екземплярів, які необхідно розмістити на матеріалі
10	Задання орієнтації деталей	Система повинна дозволяти вказувати, чи дозволено поворот деталі, а також задавати напрям орієнтації, якщо він є обов’язковим
11	Урахування технологічних параметрів	Під час формування карти розкрою повинні враховуватися ширина різу, зазори, допуски, параметри матеріалу, інструменту та верстата

Продовження табл. 2.1

1	2	3
12	Формування карти розкрою	Програма повинна автоматично розміщувати деталі на листовому або рулонному матеріалі
13	Перевірка коректності розміщення	Система повинна перевіряти вихід деталей за межі матеріалу та перетини між деталями
14	Розрахунок показників ефективності	Програма повинна розраховувати коефіцієнт використання матеріалу, площу використаного матеріалу та площу відходів
15	Візуалізація результату	Сформована карта розкрою повинна відображатися в інтерфейсі користувача
16	Експорт результатів	Система повинна забезпечувати експорт карти розкрою у формати DXF та SVG
17	Збереження даних	Програма повинна зберігати довідкові дані, конфігурації та результати в локальній базі даних SQLite

Нефункціональні вимоги до програми наведено в таблиці 2.2. До них належать автономність, зрозумілість інтерфейсу, стабільність, коректність обробки даних, достатня швидкодія та можливість подальшого розширення. Такі характеристики відповідають загальним моделям якості програмного забезпечення [15].

Таблиця 2.2 – Нефункціональні вимоги до програмного забезпечення CutMap Designer

№	Вимога	Опис вимоги
1	2	3
1	Зручність використання	Інтерфейс програми повинен бути зрозумілим для користувача, який виконує підготовку карти розкрою
2	Автономність	Програма повинна працювати локально без обов’язкового підключення до мережі Інтернет

Продовження табл. 2.2

3	Надійність збереження даних	Довідкові дані, конфігурації та результати повинні зберігатися в локальній базі SQLite
4	Коректність обробки вхідних даних	Система повинна перевіряти введені параметри та повідомляти користувача про некоректні значення
5	Продуктивність	Час формування карти розкрою повинен бути прийнятним для роботи з виробничими проєктами середньої складності
6	Масштабованість даних	Структура програми повинна дозволяти додавати нові матеріали, інструменти, верстати та формати експорту
7	Зрозумілість результатів	Користувач повинен мати можливість візуально оцінити сформовану карту розкрою та отримати числові показники ефективності
8	Сумісність результатів	Експортовані файли повинні бути придатними для подальшого використання у програмах, що підтримують формати DXF або SVG
9	Стабільність роботи	Програма повинна коректно обробляти типові помилки користувача та не завершувати роботу аварійно
10	Підтримуваність	Код програми має бути структурований за модулями, що спрощує подальше розширення та супровід

Окремо визначаються вхідні та вихідні дані системи. Вхідними даними є параметри матеріалу, інструменту та верстата, технологічні налаштування, геометрія й кількість деталей. Вихідними даними є карта розкрою, показники використання матеріалу та файл експорту. Їх перелік наведено в таблиці 2.3.

Інтерфейс повинен забезпечувати послідовну роботу з проєктом: створення або відкриття, вибір конфігурації, введення деталей, запуск оптимізації, перегляд результатів та експорт. Для матеріалів, інструментів,

верстатів, деталей і результатів доцільно передбачити окремі вікна або сторінки.

Таблиця 2.3 – Вхідні та вихідні дані програмного забезпечення CutMap Designer

Вид даних	Дані	Опис
Вхідні дані	Параметри матеріалу	Тип матеріалу, розміри, товщина, ознака листового або рулонного матеріалу
Вхідні дані	Параметри верстата	Тип обладнання, робоча область, обмеження переміщення та точності
Вхідні дані	Параметри інструменту	Тип інструменту, ширина різу, мінімальний радіус, рекомендований зазор
Вхідні дані	Технологічні налаштування	Допуски, відстані між деталями, правила повороту та орієнтації
Вхідні дані	Список деталей	Назви деталей, кількість, тип форми, геометричні параметри
Вхідні дані	Контури складних деталей	Координати вершин або дані, імпортовані з файлу
Вихідні дані	Карта розкрою	Розміщення деталей на листовому або рулонному матеріалі
Вихідні дані	Показники ефективності	Коефіцієнт використання матеріалу, площа деталей, площа відходів
Вихідні дані	Файл результату	Експортована карта розкрою у форматі DXF або SVG
Вихідні дані	Дані про проєкт	Збережена конфігурація, параметри та результати розкрою

Під час введення даних система повинна перевіряти їх коректність. Розміри матеріалу й деталей та їх кількість мають бути додатними, а ширина різу, зазори й допуски — перебувати в допустимих межах. У разі помилки програма повинна повідомляти користувача та блокувати запуск оптимізації до виправлення даних.

Алгоритм має розміщувати деталі в межах матеріалу, не допускати їх перетинання, враховувати технологічні відступи та допустимі орієнтації. Для стандартних деталей використовуються геометричні параметри, а для складних — контурне подання. Якщо розмістити всі деталі неможливо, система повинна вивести відповідне повідомлення.

Експортований файл DXF або SVG має містити межі матеріалу, контури розміщених деталей, їх координати та кути повороту. Це забезпечує подальше використання карти розкрою у спеціалізованих програмних засобах.

Таким чином, визначені вимоги є основою для проєктування функціональної схеми, архітектури WPF-застосунку, структури бази SQLite, моделі геометрії деталей та алгоритму формування карти розкрою.

2.2 Розробка функціональної схеми системи

Функціональна схема відображає основні складові програмного забезпечення, їх призначення та взаємозв’язки. Для CutMap Designer вона є основою подальшого проєктування архітектури, бази даних, алгоритму розміщення деталей і користувацького інтерфейсу.

Система призначена для створення проєктів розкрою, налаштування технологічних параметрів, введення або імпорту деталей, формування карти розкрою, візуалізації та експорту результатів у формати DXF і SVG. Тому її доцільно будувати за модульним принципом, за якого кожен компонент виконує окрему функцію та взаємодіє з іншими через визначені дані й операції [14].

До складу CutMap Designer входять модулі керування проєктами та конфігураціями, роботи з довідниками, введення й імпорту деталей, обробки геометрії, оптимізації розкрою, візуалізації, експорту результатів і доступу до бази SQLite. Загальну функціональну схему системи наведено на рисунку 2.1.

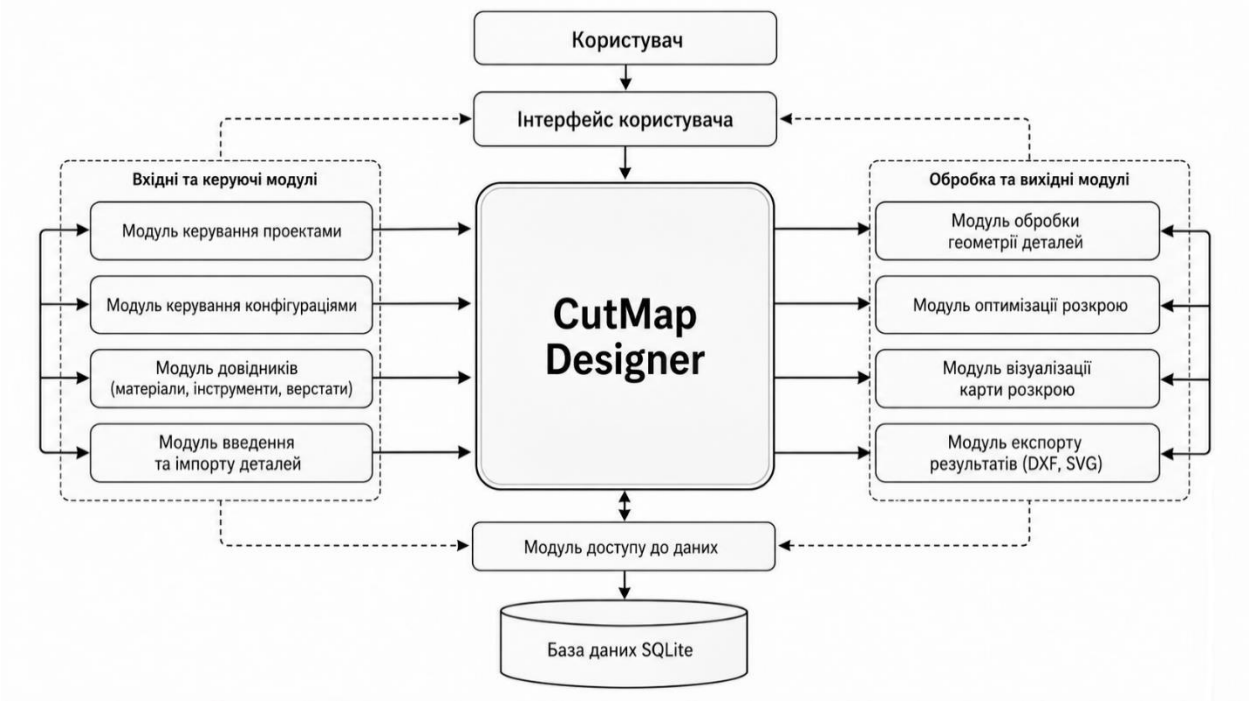

Рисунок 2.1 – Функціональна схема програмного забезпечення CutMap Designer

На рисунку 2.1 показано взаємодію користувача з основними модулями. Через графічний інтерфейс користувач створює або відкриває проєкт, вибирає конфігурацію, додає деталі, запускає оптимізацію, переглядає карту розкрою та експортує результат. Використання WPF забезпечує побудову інтерфейсу з формами введення, таблицями, командами та графічною областю відображення карти [16].

Центральним елементом схеми є модуль керування проєктами, який об’єднує конфігурацію, параметри матеріалу, список деталей і результати розкрою. Він забезпечує створення, відкриття та збереження проєктів.

Модуль конфігурацій відповідає за вибір або створення налаштувань, що містять параметри матеріалу, верстата, інструменту, ширину різу, зазори, допуски, правила повороту деталей і формат експорту. Їх збереження дає змогу повторно використовувати типові параметри в серійному виробництві.

Модуль довідників призначений для керування матеріалами, інструментами та верстатами. У ньому зберігаються характеристики

матеріалів, параметри різального інструменту та обмеження обладнання, які враховуються під час формування карти розкрою.

Модуль введення та імпорту деталей формує список елементів для розміщення. Стандартні деталі задаються геометричними параметрами, а складні — координатами вершин або імпортуються з файлу. Для кожної деталі зберігаються тип форми, кількість, габарити, контур і допустимі орієнтації.

Модуль обробки геометрії формує контури, обчислює площу та габаритні прямокутники, виконує повороти й переміщення, а також враховує технологічні відступи. Підготовлені дані передаються до модуля оптимізації, який розміщує деталі з урахуванням меж матеріалу, зазорів, допусків, ширини різу, допустимих орієнтацій і перетинів контурів.

Модуль візуалізації відображає межі матеріалу, контури та орієнтацію розміщених деталей, а також залишкові області. Модуль експорту формує файли DXF або SVG, що містять межі матеріалу, координати, контури й кути повороту деталей.

Модуль доступу до даних забезпечує збереження та отримання матеріалів, інструментів, верстатів, конфігурацій, проєктів, деталей і результатів. Використання SQLite дає змогу організувати автономне локальне збереження даних без окремого сервера [17].

Взаємодія модулів відбувається послідовно: користувач створює проєкт і конфігурацію, додає деталі, система готує їх геометрію, виконує оптимізацію, відображає результат і за потреби експортує карту. Призначення основних модулів наведено в таблиці 2.4.

Запропонована функціональна схема розділяє систему на логічно самостійні компоненти, що спрощує розроблення, тестування та подальше розширення. Зокрема, нові алгоритми оптимізації, типи деталей або формати експорту можуть додаватися без повної перебудови програми [14].

Таблиця 2.4 – Призначення основних модулів програмного забезпечення CutMap Designer

Модуль	Призначення	Модуль
Модуль керування проєктами	Створення, відкриття, редагування та збереження проєктів розкрою	Модуль керування проєктами
Модуль керування конфігураціями	Вибір збережених налаштувань або створення нових конфігурацій проєкту	Модуль керування конфігураціями
Модуль довідників	Робота з матеріалами, інструментами та верстатами	Модуль довідників
Модуль введення та імпорту деталей	Додавання стандартних і складних деталей, імпорт геометрії з файлів	Модуль введення та імпорту деталей
Модуль обробки геометрії	Формування контурів, обчислення габаритів, площ і технологічних відступів	Модуль обробки геометрії
Модуль оптимізації розкрою	Розміщення деталей на матеріалі з урахуванням технологічних параметрів	Модуль оптимізації розкрою
Модуль візуалізації	Відображення сформованої карти розкрою в інтерфейсі програми	Модуль візуалізації
Модуль експорту	Збереження результату у формати DXF та SVG	Модуль експорту
Модуль доступу до даних	Робота з локальною базою даних SQLite	Модуль доступу до даних

Таким чином, функціональна схема CutMap Designer охоплює керування проєктами й налаштуваннями, введення деталей, геометричну обробку, оптимізацію, візуалізацію, збереження та експорт результатів. Вона є основою для подальшого проєктування архітектури WPF-застосунку і структури бази даних SQLite.

2.3 Проєктування архітектури WPF-застосунку

Архітектура програмного забезпечення визначає загальну структуру системи, склад її основних компонентів, принципи їх взаємодії та спосіб розподілу відповідальності між окремими частинами програми. Для desktop-застосунку CutMap Designer архітектура повинна забезпечувати зручну роботу користувача з проєктами розкрою, збереження даних у локальній базі SQLite, виконання алгоритму оптимізації, візуалізацію карти розкрою та експорт результатів у формати DXF і SVG.

Оскільки програмне забезпечення реалізується мовою C# з використанням технології WPF, доцільним є застосування багаторівневої архітектури. Такий підхід дозволяє розділити інтерфейс користувача, логіку обробки даних, алгоритмічну частину, рівень доступу до бази даних і модуль експорту. Завдяки цьому окремі частини програми можна розробляти, тестувати та змінювати незалежно одна від одної.

Для CutMap Designer доцільно виділити такі архітектурні рівні:

а) рівень представлення;

б) рівень керування станом інтерфейсу;

в) рівень бізнес-логіки;

г) рівень алгоритму оптимізації розкрою;

д) рівень роботи з геометрією;

е) рівень доступу до даних;

ж) рівень експорту результатів.

Загальну архітектуру WPF-застосунку CutMap Designer наведено на рисунку 2.2.

На рисунку 2.2 архітектуру подано як багаторівневу схему. Верхній рівень відповідає за представлення, тобто вікна, сторінки та елементи інтерфейсу WPF. Нижче розташовується рівень ViewModel, який обробляє команди користувача та передає дані до сервісів. Далі розміщуються сервіси бізнес-логіки, модулі роботи з геометрією та алгоритм оптимізації. Окремо

виділяються рівень доступу до даних SQLite і модуль експорту результатів у формати DXF та SVG.

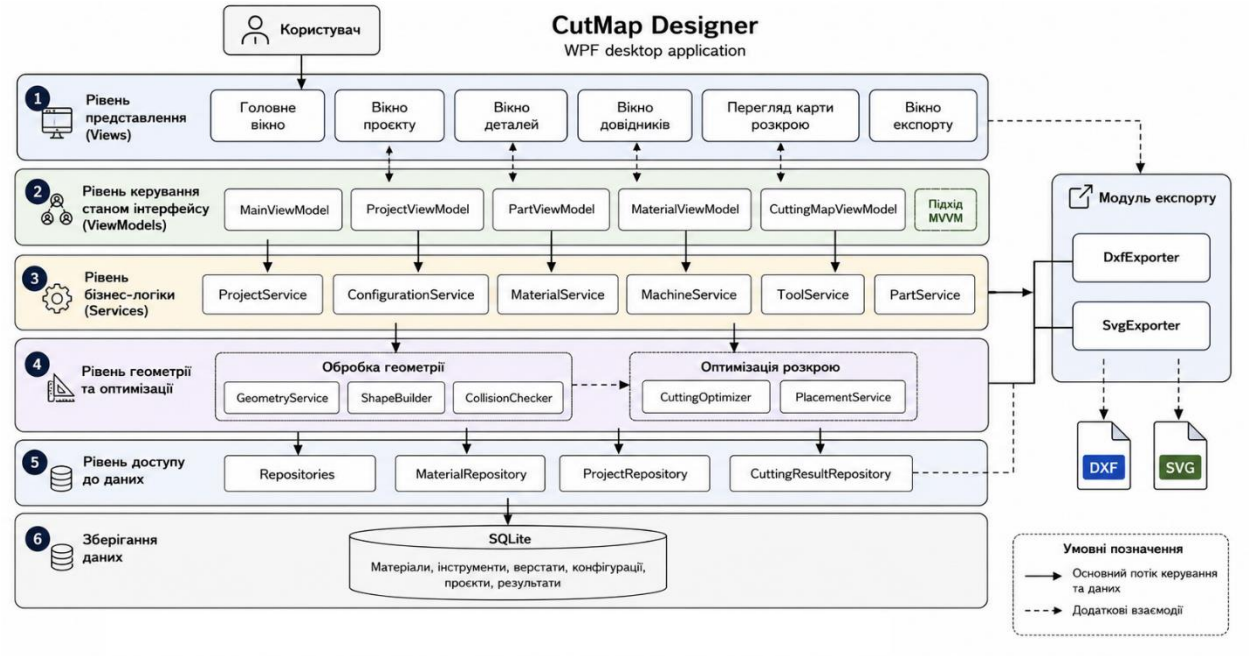

Рисунок 2.2 – Архітектура WPF-застосунку CutMap Designer

Рівень представлення забезпечує взаємодію користувача з програмою: створення проєкту, редагування довідників, введення деталей, перегляд карти розкрою та експорт результатів. Основним завданням цього рівня є відображення даних і прийняття введеної інформації, без безпосереднього виконання розрахунків або роботи з базою даних.

Для WPF-застосунку доцільно використовувати підхід MVVM, який розділяє програму на модель, представлення та модель представлення. Представлення відповідає за інтерфейс, ViewModel містить стан, властивості та команди, а модель описує основні сутності предметної області: матеріали, інструменти, верстати, деталі, проєкти, конфігурації та результати розкрою.

Рівень ViewModel містить класи, що відповідають за окремі вікна або функціональні розділи, наприклад MainViewModel, ProjectViewModel, MaterialViewModel, PartViewModel і CuttingMapViewModel. Такий підхід структурує код і спрощує підтримку програми.

Рівень бізнес-логіки містить сервіси для роботи з проєктами, конфігураціями, матеріалами, інструментами, верстатами та деталями. Вони відповідають за створення, перевірку, збереження та обробку даних.

Окреме місце займає рівень роботи з геометрією, який формує контури деталей, обчислює габаритні прямокутники та площу, виконує повороти, переміщення і перевірку перетинів. Рівень алгоритму оптимізації отримує підготовлені дані, виконує сортування деталей, генерацію орієнтацій, пошук позицій і формує карту розкрою.

Рівень доступу до даних забезпечує роботу з локальною базою SQLite через окремі репозиторії, наприклад MaterialRepository, ToolRepository, MachineRepository, ProjectRepository та CuttingResultRepository. Це дозволяє відокремити збереження даних від бізнес-логіки.

Рівень експорту формує файли DXF та SVG на основі координат розміщених деталей, їх контурів і меж матеріалу. Винесення експорту в окремий модуль спрощує додавання нових форматів у майбутньому.

Запропоновану структуру архітектурних рівнів наведено в таблиці 2.5.

Взаємодія архітектурних рівнів відбувається послідовно. Користувач працює з WPF-інтерфейсом, а дії інтерфейсу передаються до відповідної ViewModel. ViewModel звертається до сервісів бізнес-логіки, які виконують перевірку даних або передають їх до алгоритму оптимізації. Якщо потрібно зберегти або отримати дані, сервіси використовують рівень доступу до бази даних. Після формування карти розкрою результат передається до рівня візуалізації та, за потреби, до модуля експорту.

Для підтримки такої архітектури доцільно передбачити структуру проєкту, у якій класи згруповані за призначенням. Наприклад, каталог Models може містити класи Material, Machine, Tool, Part, PartPoint, CuttingProject, ProjectConfiguration та CuttingResult. Каталог ViewModels містить класи, пов’язані з інтерфейсом. Каталог Services містить бізнес-логіку, геометричну обробку та оптимізацію. Каталог Data відповідає за роботу з SQLite, а каталог Export — за формування DXF та SVG.

Таблиця 2.5 – Архітектурні рівні програмного забезпечення CutMap Designer

Рівень архітектури	Основне призначення	Приклади компонентів
Рівень представлення	Відображення інтерфейсу користувача та прийняття введених даних	WPF-вікна, сторінки, елементи керування
Рівень керування станом інтерфейсу	Обробка команд користувача та передавання даних до сервісів	MainViewModel, ProjectViewModel, PartViewModel
Рівень бізнес-логіки	Реалізація правил роботи системи та перевірка даних	ProjectService, MaterialService, ConfigurationService
Рівень роботи з геометрією	Обробка контурів, координат, площ і перетинів деталей	GeometryService, ShapeBuilder, CollisionChecker
Рівень оптимізації	Формування карти розкрою та вибір ефективного розміщення	CuttingOptimizer, PlacementService
Рівень доступу до даних	Збереження та отримання даних із SQLite	Repository-класи, DbContext, SQLite
Рівень експорту	Формування файлів результату	DxfExporter, SvgExporter

Орієнтовну структуру програмного проєкту наведено в таблиці 2.6.

Перевагою запропонованої архітектури є її зрозумілість, модульність і можливість подальшого розширення. Нові формати експорту, алгоритми оптимізації або типи деталей можуть додаватися через відповідні модулі без повної зміни структури програми.

Використання локальної бази даних SQLite відповідає характеру desktop-застосунку, оскільки забезпечує автономну роботу, збереження проєктів, довідкових даних і повторне використання конфігурацій без підключення до сервера.

Таблиця 2.6 – Орієнтовна структура проєкту CutMap Designer

Каталог проєкту	Призначення
Models	Класи предметної області: матеріали, деталі, інструменти, верстати, проєкти, результати
Views	WPF-вікна та сторінки інтерфейсу користувача
ViewModels	Класи керування станом інтерфейсу та командами користувача
Services	Бізнес-логіка, обробка параметрів, оптимізація та геометричні операції
Data	Класи роботи з базою даних SQLite
Repositories	Операції доступу до конкретних таблиць бази даних
Export	Класи формування файлів DXF та SVG
Helpers	Допоміжні класи, конвертери, перевірки та утиліти

Таким чином, архітектура CutMap Designer побудована за багаторівневим принципом із використанням підходу MVVM. Вона розділяє інтерфейс, стан представлення, бізнес-логіку, геометричну обробку, алгоритм оптимізації, доступ до даних і експорт, що забезпечує зручність розробки, тестування та подальшої підтримки програми.

2.4 Проєктування структури бази даних SQLite

Для збереження довідкових, технологічних і проєктних даних у CutMap Designer використовується локальна база даних SQLite. Такий вибір зумовлений тим, що система є desktop-застосунком і повинна працювати автономно без серверної інфраструктури. База даних зберігається у вигляді локального файлу, що спрощує розгортання, резервне копіювання та перенесення програми між робочими місцями.

У базі даних зберігаються довідники матеріалів, верстатів та інструментів, конфігурації проєктів, проєкти розкрою, перелік деталей, координати вершин складних контурів, результати побудови карти розкрою та дані про фактичне розміщення деталей на матеріалі.

Структура бази даних побудована за реляційним принципом. Кожна основна сутність подається окремою таблицею, а зв’язки між таблицями реалізуються за допомогою первинних і зовнішніх ключів. Це дозволяє уникнути дублювання даних, забезпечити цілісність інформації та спростити подальше розширення системи.

ER-діаграму структури бази даних програмного забезпечення CutMap Designer наведено на рисунку 2.3.

Таблиці Materials, Machines і Tools є довідковими. ProjectConfigurations зберігає типові конфігурації, CuttingProjects описує проєкти розкрою, Parts містить деталі проєкту, PartPoints використовується для подання складних контурів, CuttingResults зберігає результати оптимізації, а PlacedParts — координати та параметри розміщених деталей.

Перелік основних таблиць бази даних наведено в таблиці 2.7.

Таблиця 2.7 – Основні таблиці бази даних CutMap Designer

№	Назва таблиці	Призначення
1	Materials	Зберігання параметрів матеріалів
2	Machines	Зберігання параметрів верстатів
3	Tools	Зберігання параметрів інструментів
4	ProjectConfigurations	Зберігання конфігурацій проєктів розкрою
5	CuttingProjects	Зберігання інформації про проєкти
6	Parts	Зберігання деталей проєкту
7	PartPoints	Зберігання координат вершин складних деталей
8	CuttingResults	Зберігання результатів побудови карти розкрою
9	PlacedParts	Зберігання даних про розміщені деталі

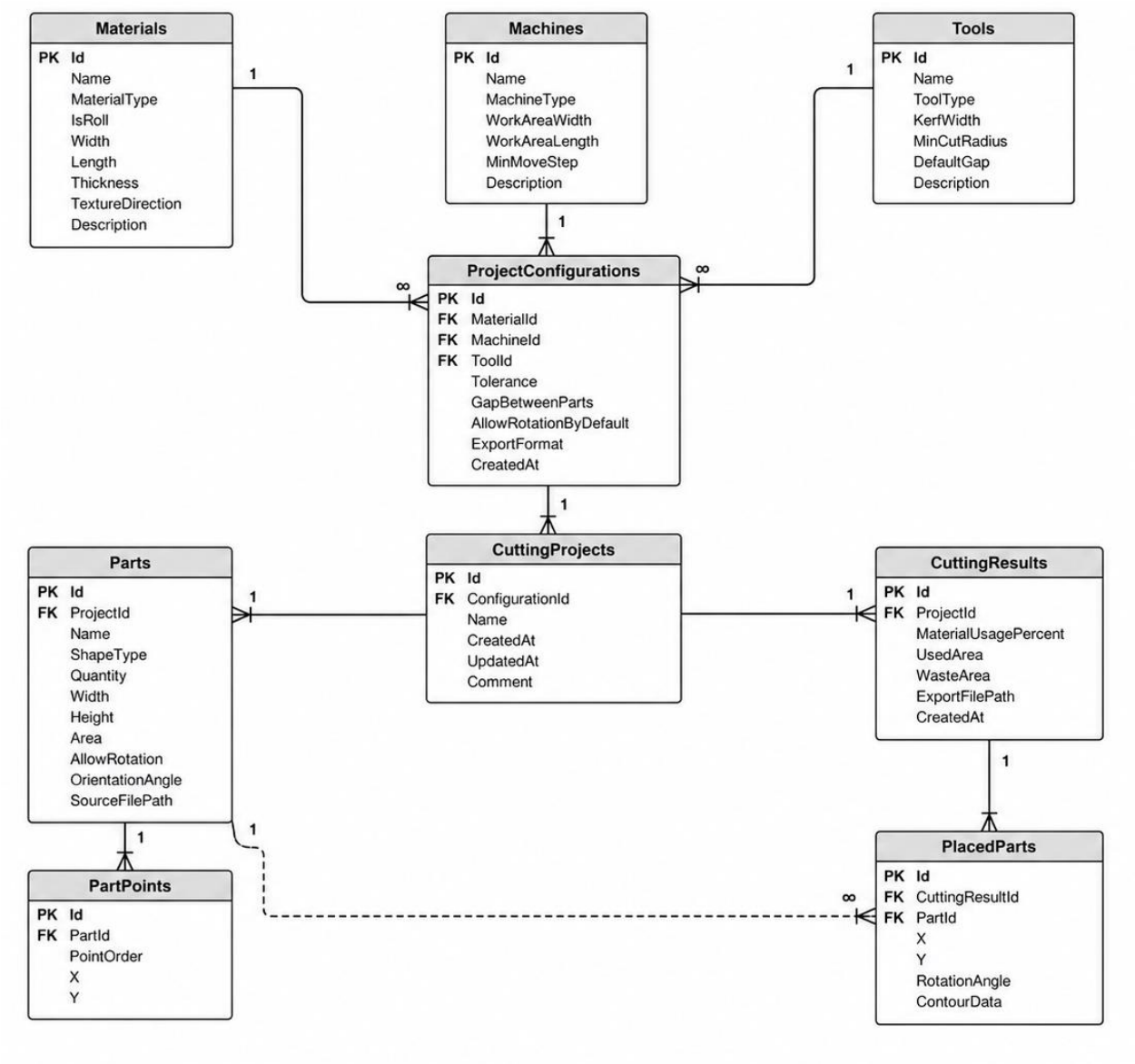

Рисунок 2.3 – ER-діаграма структури бази даних SQLite програмного забезпечення CutMap Designer

Таблиця Materials (див. табл. 2.8) призначена для збереження відомостей про матеріали, які використовуються під час формування карти розкрою. У ній зберігаються тип матеріалу, його геометричні параметри та додаткові характеристики, які можуть впливати на орієнтацію деталей.

Таблиця 2.8 – Структура таблиці Materials

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор матеріалу
2	Name	TEXT	Назва матеріалу
3	MaterialType	TEXT	Тип матеріалу: метал, пластик, фанера, тканина тощо
4	IsRoll	INTEGER	Ознака рулонного матеріалу
5	Width	REAL	Ширина матеріалу
6	Length	REAL	Довжина листа або стандартна довжина заготовки
7	Thickness	REAL	Товщина матеріалу
8	TextureDirection	TEXT	Напрям текстури, волокон або прокату
9	Description	TEXT	Додатковий опис матеріалу

Таблиця Machines (див. табл. 2.9) використовується для збереження параметрів виробничого обладнання, які впливають на можливість застосування сформованої карти розкрою.

Таблиця 2.9 – Структура таблиці Machines

№	Назва поля	Тип	Призначення
1	2	3	4
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор верстата
2	Name	TEXT	Назва верстата
3	MachineType	TEXT	Тип обладнання або спосіб обробки
4	WorkAreaWidth	REAL	Ширина робочої області
5	WorkAreaLength	REAL	Довжина робочої області
6	MinMoveStep	REAL	Мінімальний крок переміщення

Таблиця Tools (див. табл. 2.10) призначена для збереження характеристик інструментів, які використовуються під час розкрою.

Таблиця 2.10 – Структура таблиці Tools

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор інструменту
2	Name	TEXT	Назва інструменту
3	ToolType	TEXT	Тип інструменту
4	KerfWidth	REAL	Ширина різу
5	MinCutRadius	REAL	Мінімальний радіус обробки
6	DefaultGap	REAL	Рекомендований технологічний зазор
7	Description	TEXT	Додатковий опис інструменту

Таблиця ProjectConfigurations (див. табл. 2.11) використовується для збереження конфігурацій проєктів розкрою. Вона дозволяє повторно використовувати типові набори параметрів у серійному виробництві.

Таблиця 2.11 – Структура таблиці ProjectConfigurations

№	Назва поля	Тип	Призначення
1	Id	INTEGER (PK)	Унікальний ідентифікатор конфігурації
2	Name	TEXT	Назва конфігурації
3	MaterialId	INTEGER	Посилання на таблицю Materials
4	MachineId	INTEGER	Посилання на таблицю Machines
5	ToolId	INTEGER	Посилання на таблицю Tools
6	Tolerance	REAL	Допуск на обробку
7	GapBetweenParts	REAL	Мінімальна відстань між деталями
8	AllowRotationByDefault	INTEGER	Ознака дозволу на поворот деталей
9	ExportFormat	TEXT	Формат експорту результату
10	CreatedAt	TEXT	Дата створення конфігурації

Таблиця CuttingProjects (див. табл. 2.12) містить загальні відомості про проєкти розкрою. Кожен проєкт пов’язаний з однією конфігурацією.

Таблиця 2.12 – Структура таблиці CuttingProjects

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор проєкту
2	Name	TEXT	Назва проєкту
3	ConfigurationId	INTEGER	Посилання на таблицю ProjectConfigurations
4	CreatedAt	TEXT	Дата створення проєкту
5	UpdatedAt	TEXT	Дата останнього оновлення
6	Comment	TEXT	Коментар до проєкту

Таблиця Parts (див. табл. 2.13) призначена для збереження деталей, які входять до складу конкретного проєкту розкрою.

Таблиця 2.13 – Структура таблиці Parts

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор деталі
2	ProjectId	INTEGER	Посилання на таблицю CuttingProjects
3	Name	TEXT	Назва деталі
4	ShapeType	TEXT	Тип форми деталі
5	Quantity	INTEGER	Кількість екземплярів деталі
6	Width	REAL	Ширина або габаритна ширина
7	Height	REAL	Висота або габаритна висота
8	Area	REAL	Площа деталі
9	AllowRotation	INTEGER	Ознака дозволу на поворот
10	OrientationAngle	REAL	Кут орієнтації деталі
11	SourceFilePath	TEXT	Шлях до файлу імпортованої геометрії

Таблиця PartPoints (див. табл. 2.14) використовується для опису складних контурів деталей у вигляді набору координат вершин.

Таблиця 2.14 – Структура таблиці PartPoints

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор точки
2	PartId	INTEGER	Посилання на таблицю Parts
3	PointOrder	INTEGER	Порядковий номер вершини контуру
4	X	REAL	Координата точки по осі X
5	Y	REAL	Координата точки по осі Y

Таблиця CuttingResults (див. табл. 2.15) призначена для збереження результатів формування карти розкрою та числових показників ефективності.

Таблиця 2.15 – Структура таблиці CuttingResults

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор результату
2	ProjectId	INTEGER	Посилання на таблицю CuttingProjects
3	MaterialUsagePercent	REAL	Коефіцієнт використання матеріалу
4	UsedArea	REAL	Площа, зайнята деталями
5	WasteArea	REAL	Площа відходів
6	ExportFilePath	TEXT	Шлях до експортованого файлу
7	CreatedAt	TEXT	Дата створення результату

Таблиця PlacedParts (див. табл. 2.16) містить дані про фактичне розміщення деталей на карті розкрою. Ця таблиця потрібна для повторного відображення карти, аналізу результату та експорту у векторні формати.

Запропонована структура бази даних забезпечує збереження всієї інформації, необхідної для роботи програмного забезпечення CutMap Designer. Вона підтримує збереження довідкових даних, конфігурацій, проєктів, деталей стандартної та складної форми, результатів оптимізації та експортованих даних. Крім того, така структура є достатньо гнучкою для подальшого розширення, наприклад додавання нових типів параметрів, форматів експорту або сервісних таблиць.

Таблиця 2.16 – Структура таблиці PlacedParts

№	Назва поля	Тип	Призначення
1	Id	INTEGER PRIMARY KEY	Унікальний ідентифікатор розміщеної деталі
2	CuttingResultId	INTEGER	Посилання на таблицю CuttingResults
3	PartId	INTEGER	Посилання на таблицю Parts
4	X	REAL	Координата розміщення по осі X
5	Y	REAL	Координата розміщення по осі Y
6	RotationAngle	REAL	Кут повороту деталі
7	ContourData	TEXT	Дані контуру після розміщення

Таким чином, спроєктована структура бази даних SQLite є основою для функціонування системи збереження даних у програмному забезпеченні CutMap Designer. Вона забезпечує цілісність, структурованість і зручність доступу до інформації, необхідної для формування та збереження карт розкрою.

2.5 Розробка моделі подання геометрії деталей

Для формування карти розкрою CutMap Designer повинно мати внутрішню модель геометрії деталей. Вона використовується для збереження параметрів, обробки стандартних і складних форм, обчислення площі, перевірки перетинів, виконання поворотів і переміщень, а також підготовки даних до візуалізації та експорту.

У системі підтримуються стандартні деталі та деталі складної форми. Стандартні фігури задаються геометричними параметрами, наприклад шириною, висотою, діаметром або радіусом. Складні деталі подаються контуром у вигляді впорядкованої послідовності точок із координатами (X) та (Y). Контурне подання дає змогу виконувати геометричні перетворення та перевіряти взаємне розташування деталей [3, 6].

Загальну схему моделі подання геометрії деталей у CutMap Designer наведено на рисунку 2.4.

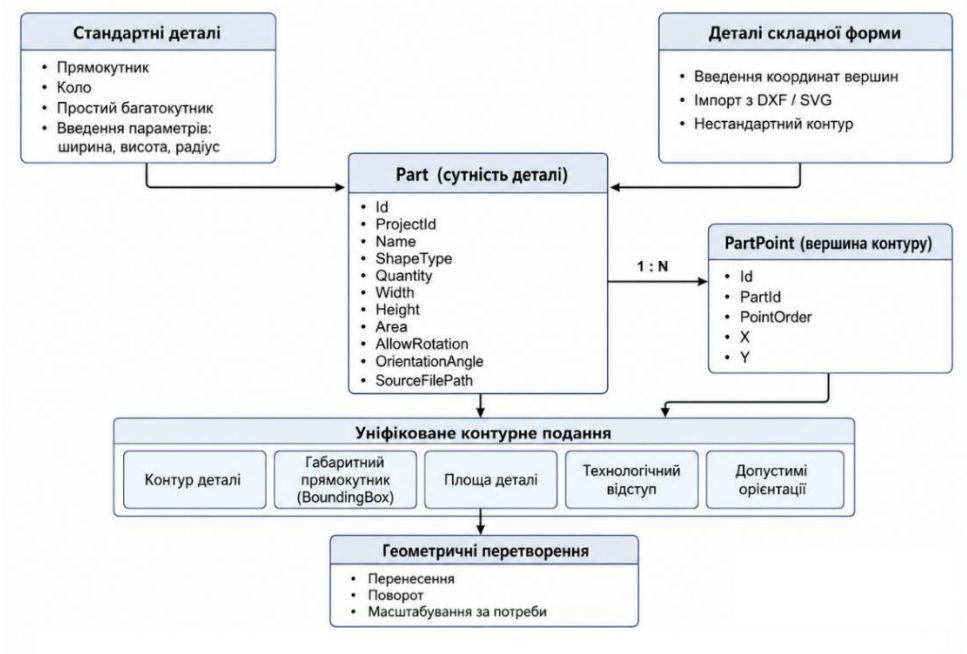

Рисунок 2.4 – Модель подання геометрії деталей у CutMap Designer

Основною сутністю моделі є Part, яка містить назву деталі, тип форми, кількість екземплярів, габарити, площу, параметри повороту та джерело геометрії. Для складних деталей контур описується набором пов’язаних точок.

Параметр ShapeType визначає спосіб формування контуру. Для стандартних деталей можуть використовуватися типи Rectangle, Circle і Polygon, а для складних — CustomContour або ImportedContour. Координати вершин складного контуру зберігаються у сутності PartPoint разом із порядковим номером точки. Правильна послідовність вершин є необхідною для коректної побудови багатокутника [6].

Для внутрішньої обробки деталі приводяться до уніфікованого подання. Прямокутник може бути описаний чотирма вершинами, складний контур — довільною кількістю точок, а коло — параметрично або наближеною полігональною моделлю. Види подання деталей наведено в таблиці 2.17.

Таблиця 2.17 – Види подання деталей у CutMap Designer

№	Вид деталі	Спосіб подання	Дані, що використовуються
1	Прямокутна деталь	Параметричне або контурне подання	Ширина, висота, чотири вершини
2	Кругла деталь	Параметричне подання або наближення контуром	Радіус або діаметр
3	Багатокутна деталь	Контурне подання	Послідовність координат вершин
4	Деталь складної форми	Контурне подання	Набір точок контуру
5	Імпортована деталь	Контур, отриманий з файлу	Дані з DXF або SVG

Для кожної деталі формується габаритний прямокутник, визначений мінімальними та максимальними координатами контуру. Він використовується для сортування, попередньої перевірки розміщення та швидкого виявлення можливих перетинів. Якщо габаритні області перетинаються, виконується точніша перевірка фактичних контурів [3, 6].

Послідовність переходу від вхідного опису деталі до внутрішнього геометричного подання наведено на рисунку 2.5.

Для стандартних деталей контур формується автоматично за введеними параметрами. Складний контур може задаватися координатами вершин або імпортуватися з файлу, після чого приводиться до внутрішнього формату системи.

Після формування контуру обчислюється площа деталі. Для стандартних форм використовуються відповідні геометричні формули, а для багатокутників — координати впорядкованих вершин. Значення площі

застосовується під час сортування деталей і розрахунку коефіцієнта використання матеріалу.

Рисунок 2.5 – Послідовність формування внутрішнього подання геометрії деталі

Модель повинна підтримувати поворот і переміщення деталей. Якщо поворот дозволений, алгоритм може перевіряти орієнтації 0°, 90°, 180° і 270° або інші задані значення. Під час переміщення та повороту координати всіх точок контуру перераховуються відповідно до нового положення.

Окремо формується технологічна зона навколо деталі, яка враховує ширину різу, зазори та допуски. Основний контур описує фактичну геометрію деталі, а розширений використовується для перевірки допустимої відстані між сусідніми елементами.

Таким чином, запропонована модель забезпечує єдиний підхід до стандартних і складних деталей. Вона підтримує параметричні форми,

координатні контури, імпорт геометрії, обчислення площі, формування габаритної області, технологічні відступи, поворот і переміщення деталей.

2.6 Розробка алгоритму оптимізованого формування карти розкрою

Алгоритм формування карти розкрою є основною розрахунковою частиною CutMap Designer. Його призначення полягає у розміщенні заданої кількості деталей на листовому або рулонному матеріалі з урахуванням їх геометрії, допустимих орієнтацій, ширини різу, зазорів, допусків, параметрів інструменту та обмежень обладнання.

Для системи обрано комбінований евристичний підхід. Він не гарантує абсолютного оптимуму, однак дозволяє отримати практично придатний результат за прийнятний час, поєднуючи швидке формування початкової розкладки з оцінюванням і вибором кращого варіанта [11–13].

Алгоритм передбачає підготовку вхідних даних, формування геометрії деталей, створення необхідної кількості їх екземплярів, сортування за площею або габаритами, генерацію допустимих орієнтацій, пошук позицій, перевірку меж і перетинів та оцінювання сформованої карти. Загальну послідовність його роботи наведено на рисунку 1.8, а характеристику основних етапів — у таблиці 2.21.

Вхідними даними є параметри матеріалу, інструменту й верстата, технологічні налаштування, список деталей, їх кількість, контури та правила орієнтації. Вихідними даними є координати розміщених деталей, кути повороту, показники використання матеріалу та дані для візуалізації й експорту у формати DXF або SVG.

На початковому етапі система перевіряє розміри матеріалу, кількість деталей, ширину різу, зазори та допуски. Якщо дані є некоректними або неповними, формування карти не запускається, а користувач отримує відповідне повідомлення.

Таблиця 2.21 – Основні етапи алгоритму формування карти розкрою

№	Етап алгоритму	Призначення
1	Підготовка вхідних даних	Перевірка параметрів матеріалу, верстата, інструменту та списку деталей
2	Формування геометрії деталей	Створення контурів, обчислення площі та габаритних прямокутників
3	Формування екземплярів	Створення потрібної кількості копій кожної деталі
4	Сортування деталей	Визначення порядку розміщення, переважно від більших деталей до менших
5	Генерація орієнтацій	Формування допустимих кутів повороту для кожної деталі
6	Пошук позиції	Підбір координат для розміщення деталі на матеріалі
7	Перевірка допустимості	Перевірка меж матеріалу, перетинів і технологічних відступів
8	Розміщення деталі	Збереження координат, кута повороту та контуру розміщеної деталі
9	Оцінювання карти	Розрахунок площі відходів і коефіцієнта використання матеріалу
10	Вибір результату	Вибір карти з найкращими показниками та передавання її до візуалізації й експорту

Далі стандартні деталі перетворюються на внутрішні контури за заданими параметрами, а складні або імпортовані — на впорядковані набори координат. Для кожної деталі також визначаються площа та габаритний прямокутник, що використовуються під час сортування й попередньої перевірки перетинів [3, 6].

Відповідно до заданої кількості створюються окремі екземпляри деталей. Перед розміщенням вони сортуються за площею або габаритами, оскільки розташування більших елементів на початку зменшує ризик утворення непридатних для них вільних областей [11, 12].

Для кожної деталі формується набір допустимих орієнтацій. Якщо поворот дозволений, перевіряються кути 0°, 90°, 180° і 270° або інші значення,

задані конфігурацією. За наявності напряму волокон, текстури чи інших обмежень використовується лише дозволена орієнтація.

Позиція вважається допустимою, якщо деталь повністю розміщується в межах матеріалу, не перетинається з уже розташованими елементами та зберігає необхідний технологічний відступ. Схему перевірки допустимості позиції наведено на рисунку 2.6.

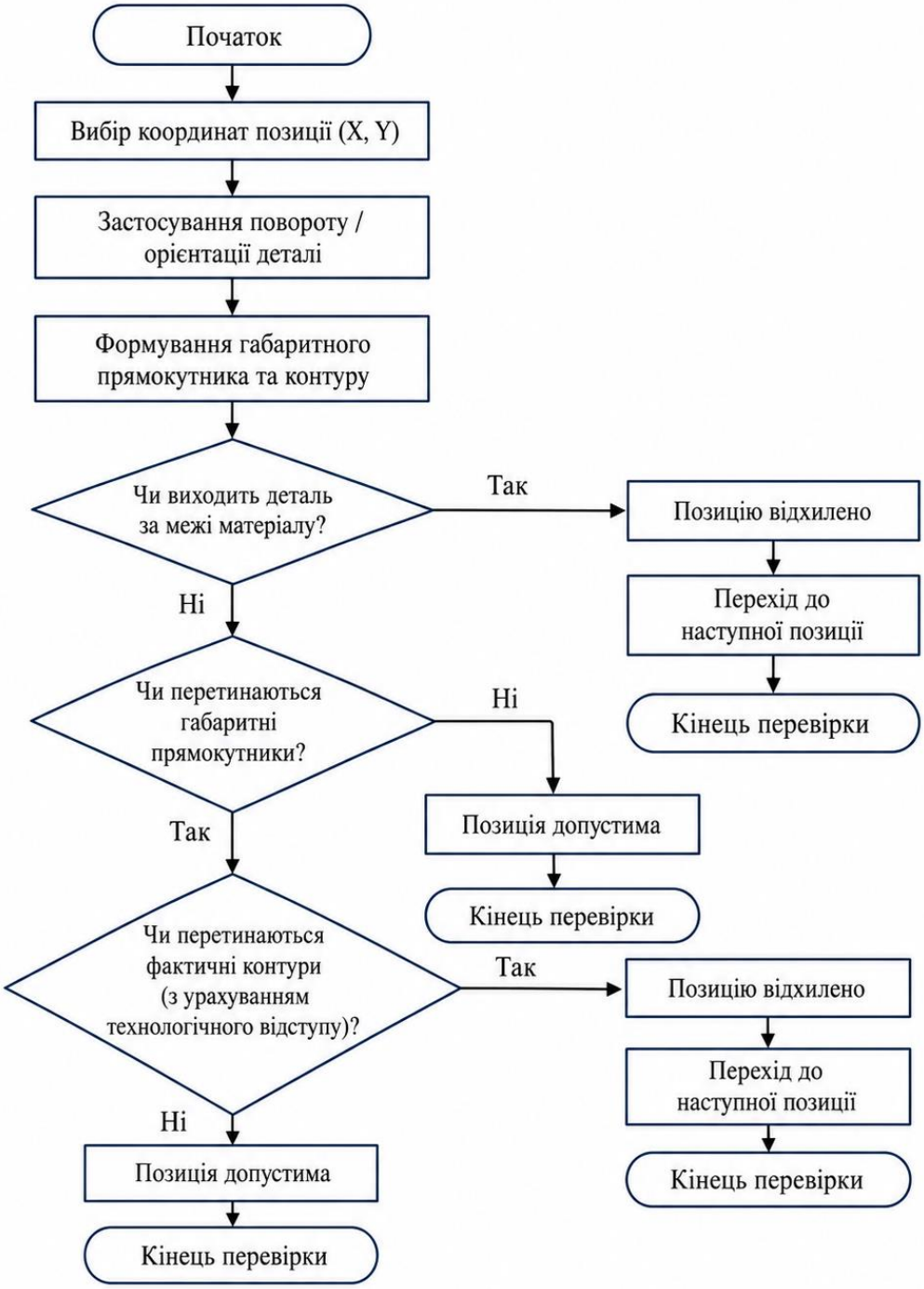

Рисунок 2.6 – Схема перевірки допустимості позиції деталі

Для скорочення кількості обчислень перевірка перетинів виконується у два етапи. Спочатку порівнюються габаритні прямокутники. Якщо вони не перетинаються, точний аналіз не потрібний. У протилежному випадку перевіряються фактичні контури з урахуванням технологічної зони [3, 6].

Після розміщення деталей виконується оцінювання карти розкрою. Основним показником є коефіцієнт використання матеріалу:

$$K = {S_Д}/{S_M} \cdot 100\%,$$

де: K — коефіцієнт використання матеріалу, %; Sд — сумарна площа розміщених деталей; Sм — площа матеріалу або використаної області матеріалу.

Площа відходів визначається за формулою:

$$S_B = S_M - S_Д,$$

де: Sв — площа відходів; Sм — площа матеріалу або використаної області матеріалу; Sд — сумарна площа розміщених деталей.

Для покращення результату система може формувати кілька варіантів розкладки, змінюючи порядок сортування, початкові позиції або допустимі орієнтації. Варіанти порівнюються за коефіцієнтом використання матеріалу, площею відходів і кількістю нерозміщених деталей. Схему вибору кращої карти подано на рисунку 2.7.

У спрощеному вигляді логіка роботи алгоритму включає такі дії: отримання параметрів проєкту, перевірку даних, формування контурів і габаритних прямокутників, створення екземплярів деталей, сортування, генерацію орієнтацій, пошук допустимих позицій, перевірку меж і перетинів, розміщення деталей, розрахунок показників ефективності та передавання результату до модулів візуалізації й експорту.

Перевагою запропонованого алгоритму є його практична придатність для реалізації в desktop-застосунку. Він не потребує надмірно складних математичних обчислень, але дозволяє враховувати форму деталей,

технологічні зазори, ширину різу, допуски, параметри обладнання та обмеження орієнтації.

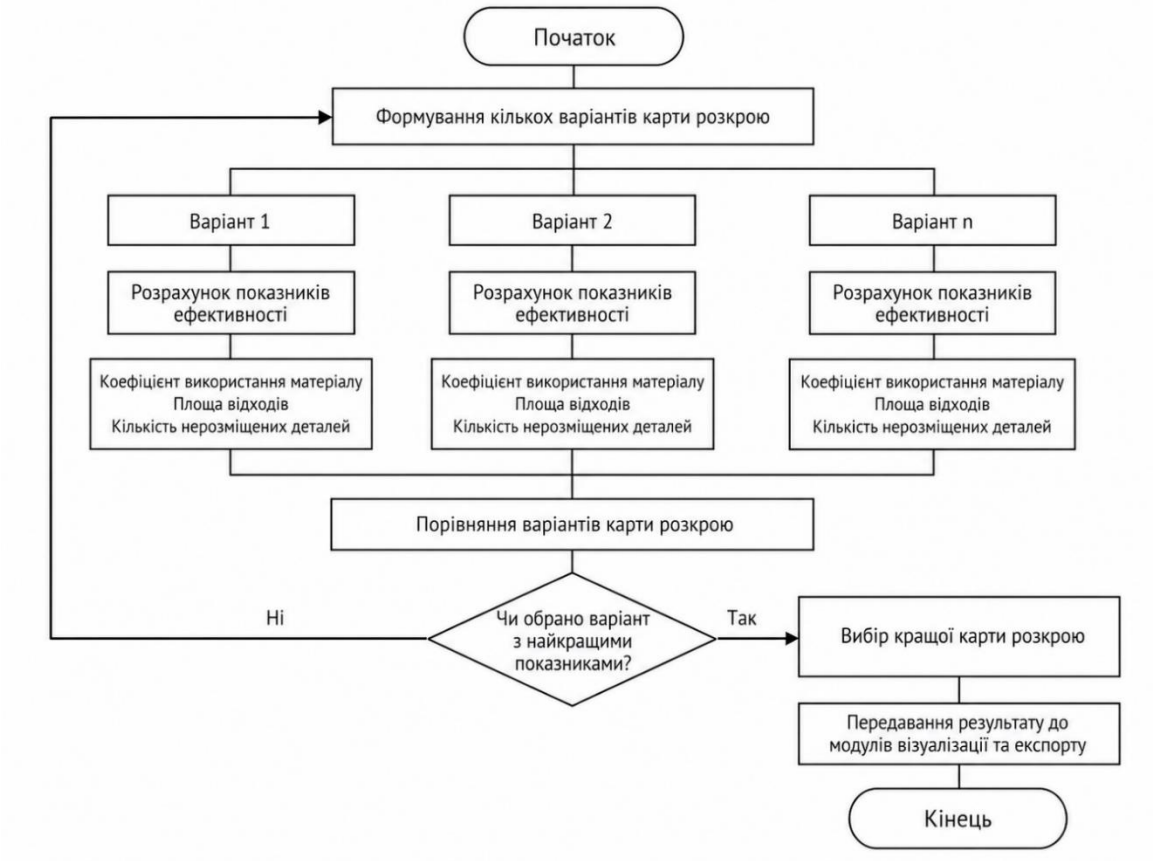

Рисунок 2.7 – Схема вибору кращої карти розкрою

Таким чином, запропонований алгоритм охоплює перевірку вхідних даних, підготовку геометрії, сортування деталей, генерацію орієнтацій, пошук допустимих позицій, перевірку обмежень і вибір кращої карти розкрою. Комбінований евристичний підхід забезпечує прийнятний баланс між швидкістю обчислень та ефективністю використання матеріалу [12, 13].

2.7 Проєктування інтерфейсу користувача

Інтерфейс користувача є важливою частиною програмного забезпечення CutMap Designer, оскільки через нього користувач створює проєкт розкрою, задає технологічні параметри, вводить або імпортує деталі, запускає оптимізацію, переглядає карту та виконує експорт результатів. Тому

інтерфейс повинен бути зрозумілим, послідовним і не перевантаженим зайвими елементами [15].

Для реалізації інтерфейсу обрано технологію WPF, яка дозволяє створювати desktop-застосунки з формами введення, таблицями, панелями навігації та графічними областями для візуалізації [16]. Це є доцільним для CutMap Designer, оскільки програма повинна не лише приймати параметри, але й відображати карту розкрою з контурами деталей.

Інтерфейс доцільно побудувати за принципом послідовного робочого процесу: створення або відкриття проєкту, вибір конфігурації, задання матеріалу, інструменту та верстата, додавання деталей, формування карти розкрою, перегляд результату та експорт. Основні вікна й елементи інтерфейсу наведено в таблиці 2.22.

Таблиця 2.22 – Основні елементи інтерфейсу користувача CutMap Designer

№	Елемент інтерфейсу	Призначення
1	Головне вікно	Забезпечує доступ до основних функцій програми
2	Вікно створення проєкту	Дозволяє створити новий проєкт розкрою або відкрити існуючий
3	Вікно конфігурації проєкту	Використовується для вибору збережених налаштувань або введення нових параметрів
4	Вікно довідників	Забезпечує роботу з матеріалами, інструментами та верстатами
5	Вікно введення деталей	Дозволяє вводити стандартні деталі, складні контури або імпортувати геометрію
6	Вікно карти розкрою	Відображає сформовану карту розміщення деталей на матеріалі
7	Панель результатів	Показує коефіцієнт використання матеріалу, площу відходів і кількість розміщених деталей
8	Вікно експорту	Дозволяє вибрати формат DXF або SVG і зберегти результат у файл

Для уточнення логіки роботи користувача розроблено UML-діаграму переходів між інтерфейсними вікнами, наведену на рисунку 2.8. Вона показує основний сценарій роботи, а також додаткові переходи, пов’язані з редагуванням даних, виправленням помилок або повторним застосуванням налаштувань. Використання UML дає змогу візуально подати стани системи та переходи між ними [19].

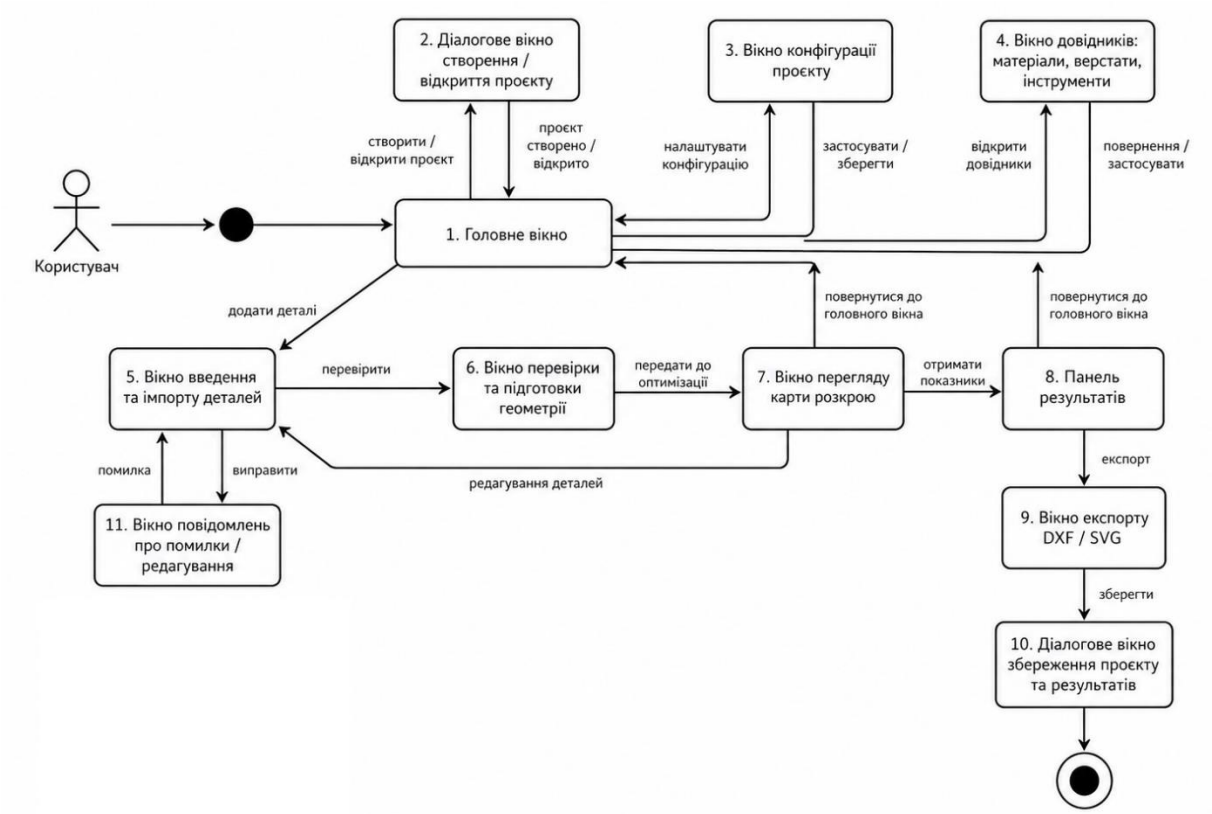

Рисунок 2.8 – UML-діаграма переходів між інтерфейсними вікнами CutMap Designer

Головне вікно є початковою точкою роботи користувача. У ньому розміщується меню або панель навігації для створення чи відкриття проєкту, роботи з довідниками, запуску оптимізації та експорту. Також у головному вікні відображаються дані про поточний проєкт, вибрану конфігурацію та стан виконання операцій.

Вікно створення проєкту призначене для введення назви, вибору конфігурації та зазначення коментаря. Якщо користувач обирає збережену

конфігурацію, програма автоматично підтягує параметри матеріалу, верстата, інструменту та технологічні налаштування.

Вікно конфігурації дозволяє задавати параметри, що впливають на формування карти розкрою: матеріал, верстат, інструмент, ширину різу, технологічний зазор, допуск, дозвіл на поворот деталей і формат експорту. Також передбачається можливість збереження конфігурації для повторного використання.

Вікно довідників використовується для керування матеріалами, інструментами та верстатами. Для кожного довідника доцільно передбачити таблицю записів і форму редагування вибраного елемента.

Вікно введення деталей дає змогу додавати стандартні деталі, задавати їх розміри, кількість, дозвіл на поворот і кут орієнтації. Для складних деталей передбачається введення координат вершин контуру або імпорт геометрії з файлу.

Вікно карти розкрою призначене для візуального відображення результату роботи алгоритму. У ньому показуються межі матеріалу, контури розміщених деталей, їх орієнтація та залишкові області. Для зручності користувача доцільно передбачити масштабування, переміщення області перегляду та виділення окремої деталі.

Панель результатів відображає основні показники сформованої карти: коефіцієнт використання матеріалу, сумарну площу деталей, площу відходів, кількість розміщених і нерозміщених деталей. Вікно експорту дозволяє вибрати формат DXF або SVG, зазначити шлях збереження та сформувати файл результату.

Схему розташування основних елементів головного вікна CutMap Designer наведено на рисунку 2.9. У верхній частині розміщується головне меню, ліворуч — панель навігації, у центрі — область візуалізації карти розкрою, праворуч — панель параметрів і результатів, а внизу — статусний рядок і кнопки основних дій.

Інтерфейс повинен забезпечувати перевірку введених даних. Програма не повинна дозволяти запуск оптимізації без матеріалу, інструменту або списку деталей. Також необхідно перевіряти числові поля: розміри, кількість, зазори та допуски. У разі помилки система повинна повідомити користувача, яке саме поле потрібно виправити. Механізми прив’язування даних WPF підтримують синхронізацію елементів інтерфейсу з даними та передавання повідомлень про помилки перевірки [20].

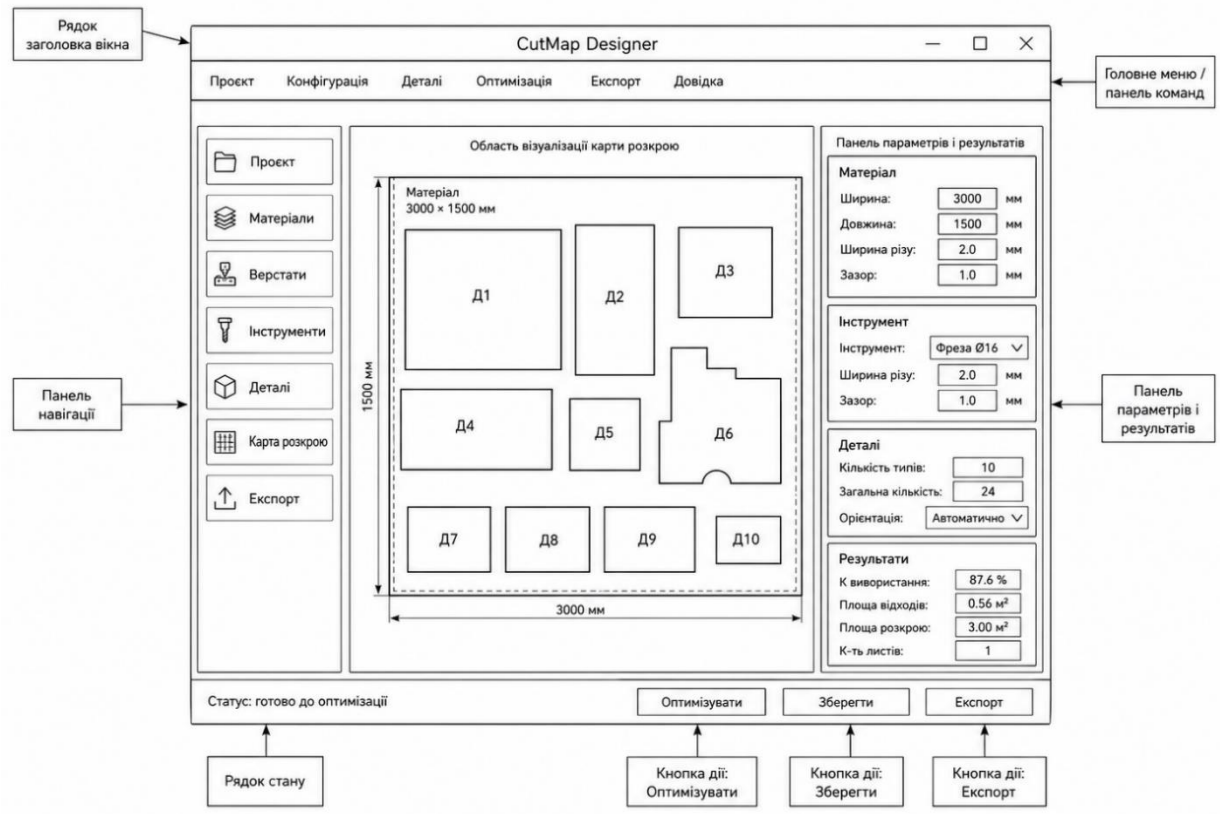

Рисунок 2.9 – Схема розташування елементів головного вікна CutMap Designer

Проєктування інтерфейсу має враховувати архітектуру WPF-застосунку. Вікна належать до рівня представлення, а їхній стан і команди керуються відповідними ViewModel-класами. Наприклад, вікно введення деталей взаємодіє з PartViewModel, а вікно карти розкрою — з CuttingMapViewModel. Такий підхід відповідає шаблону MVVM і відокремлює логіку інтерфейсу від бізнес-логіки [20].

Таким чином, інтерфейс користувача CutMap Designer забезпечує повний цикл роботи з картою розкрою: створення проєкту, налаштування параметрів, введення або імпорт деталей, запуск алгоритму, перегляд результатів та експорт. Запропонована структура робить програмне забезпечення зручним для використання й відповідає функціональним вимогам.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ DESKTOP-ЗАСТОСУНКУ

3.1 Обґрунтування вибору C#, WPF та SQLite

Для реалізації програмного забезпечення CutMap Designer було обрано мову програмування C#, технологію побудови інтерфейсу WPF та локальну базу даних SQLite. Такий набір технологій є доцільним для desktop-застосунку, який повинен працювати автономно, мати зручний графічний інтерфейс, виконувати розрахунки оптимізації та зберігати довідкові й проєктні дані.

Мова C# є однією з основних мов платформи .NET і використовується для створення різних типів прикладного програмного забезпечення. Вона підтримує об’єктно-орієнтоване програмування та надає засоби роботи з файлами, колекціями, графічними даними й базами даних [21]. Для CutMap Designer це важливо, оскільки система повинна обробляти геометричні дані, працювати зі списками деталей, виконувати алгоритм розміщення та формувати файли експорту.

Технологія WPF обрана для створення графічного інтерфейсу користувача. Вона дозволяє розробляти desktop-застосунки з гнучкою структурою вікон, формами введення, таблицями, панелями навігації та областями візуалізації [16]. Крім того, механізми прив’язування даних WPF дають змогу реалізувати підхід MVVM і відокремити інтерфейс від бізнес-логіки програми [20].

База даних SQLite використовується для локального збереження матеріалів, верстатів, інструментів, конфігурацій проєктів, деталей і результатів розкрою. Вона не потребує окремого сервера та зберігає дані у локальному файлі, що дозволяє користувачу працювати з програмою автономно [17].

Таким чином, вибір C#, WPF та SQLite є обґрунтованим, оскільки ці технології відповідають вимогам до CutMap Designer: автономність, зручний

інтерфейс, підтримка роботи з геометричними даними, локальне збереження налаштувань і можливість подальшого розширення функціональності.

3.2 Реалізація рівня доступу до даних

Рівень доступу до даних у програмному забезпеченні CutMap Designer призначений для взаємодії з локальною базою даних SQLite. Він забезпечує збереження, отримання, оновлення та видалення даних про матеріали, верстати, інструменти, конфігурації проєктів, деталі, координати контурів, результати оптимізації та розміщені деталі.

Виділення окремого рівня доступу до даних дозволяє відокремити роботу з базою від інтерфейсу користувача та бізнес-логіки. Застосування шаблону Repository забезпечує абстрагування операцій збереження даних і зменшує залежність інших компонентів системи від конкретної реалізації бази даних [22]. Завдяки цьому WPF-вікна не виконують SQL-запити напряму, а звертаються до відповідних сервісів або репозиторіїв, що спрощує супровід програми та зменшує дублювання коду.

У структурі програмного забезпечення цей рівень доцільно розмістити в окремому каталозі Data або Repositories. У ньому реалізуються класи для роботи з окремими таблицями, наприклад MaterialRepository, MachineRepository, ToolRepository, ProjectRepository, PartRepository та CuttingResultRepository. Кожен репозиторій інкапсулює операції доступу до відповідної групи даних і передає результати сервісам бізнес-логіки [22].Основні функції рівня доступу до даних наведено в таблиці 3.1.

Під час запуску програми виконується перевірка наявності файлу бази даних. Якщо файл відсутній, система створює нову базу даних і формує необхідні таблиці. Якщо база даних уже існує, програма підключається до неї та використовує збережені дані. Створення структури бази даних виконується за допомогою SQL-запитів CREATE TABLE, які відповідають структурі таблиць, спроєктованій у підрозділі 2.4.

Таблиця 3.1 – Основні функції рівня доступу до даних

№	Функція	Призначення
1	Створення бази даних	Формування файлу SQLite та необхідних таблиць
2	Додавання даних	Запис нових матеріалів, інструментів, верстатів, деталей і проєктів
3	Отримання даних	Завантаження довідників, конфігурацій, проєктів і результатів
4	Оновлення даних	Зміна параметрів уже збережених записів
5	Видалення даних	Видалення непотрібних або помилково створених записів
6	Підтримка зв’язків	Використання зовнішніх ключів між пов’язаними таблицями
7	Збереження результатів	Запис сформованої карти розкрою та даних про розміщені деталі

Спочатку створюються довідкові таблиці Materials, Machines і Tools, оскільки вони використовуються в таблиці конфігурацій проєктів. Приклад SQL-запитів для створення довідкових таблиць:

```
CREATE TABLE IF NOT EXISTS Materials (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    MaterialType TEXT,
    IsRoll INTEGER NOT NULL DEFAULT 0,
    Width REAL NOT NULL,
    Length REAL,
    Thickness REAL,
    TextureDirection TEXT,
    Description TEXT
);

CREATE TABLE IF NOT EXISTS Machines (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    MachineType TEXT,
    WorkAreaWidth REAL,
    WorkAreaLength REAL,
    MinMoveStep REAL,
    Description TEXT
);

CREATE TABLE IF NOT EXISTS Tools (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
```

```
    ToolType TEXT,
    KerfWidth REAL NOT NULL,
    MinCutRadius REAL,
    DefaultGap REAL,
    Description TEXT
).
```

Таблиця Materials зберігає параметри матеріалів, які можуть використовуватися в проєктах розкрою. Поле IsRoll визначає, чи є матеріал рулонним. Таблиця Machines містить дані про верстати та їхню робочу область. Таблиця Tools зберігає характеристики інструментів, зокрема ширину різу, мінімальний радіус обробки та рекомендований технологічний зазор.

Після створення довідкових таблиць формується таблиця ProjectConfigurations. Вона містить посилання на матеріал, верстат та інструмент, а також параметри, які впливають на формування карти розкрою: допуск, зазор між деталями, дозвіл на поворот та формат експорту.

SQL-запит для створення таблиці конфігурацій має вигляд:

```
CREATE TABLE IF NOT EXISTS ProjectConfigurations (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    MaterialId INTEGER NOT NULL,
    MachineId INTEGER NOT NULL,
    ToolId INTEGER NOT NULL,
    Tolerance REAL DEFAULT 0,
    GapBetweenParts REAL DEFAULT 0,
    AllowRotationByDefault INTEGER NOT NULL DEFAULT 1,
    ExportFormat TEXT DEFAULT 'DXF',
    CreatedAt TEXT NOT NULL,

    FOREIGN KEY (MaterialId) REFERENCES Materials(Id),
    FOREIGN KEY (MachineId) REFERENCES Machines(Id),
    FOREIGN KEY (ToolId) REFERENCES Tools(Id)
);
```

Таблиця CuttingProjects використовується для збереження проєктів розкрою. Кожен проєкт пов’язується з певною конфігурацією. Це дозволяє повторно використовувати однакові налаштування для різних виробничих завдань.

SQL-запит для створення таблиці проєктів має вигляд:

```
CREATE TABLE IF NOT EXISTS CuttingProjects (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    ConfigurationId INTEGER NOT NULL,
```

```
        CreatedAt TEXT NOT NULL,
        UpdatedAt TEXT,
        Comment TEXT,

        FOREIGN KEY (ConfigurationId) REFERENCES
ProjectConfigurations(Id)
    );
```

Для збереження деталей проєкту використовується таблиця Parts. Вона містить загальні параметри деталі: назву, тип форми, кількість, габарити, площу, дозвіл на поворот, кут орієнтації та шлях до файлу імпортованої геометрії. Якщо деталь має складну форму, її вершини додатково зберігаються в таблиці PartPoints.

SQL-запити для створення таблиць Parts і PartPoints має вигляд:

```
    CREATE TABLE IF NOT EXISTS Parts (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        ProjectId INTEGER NOT NULL,
        Name TEXT NOT NULL,
        ShapeType TEXT NOT NULL,
        Quantity INTEGER NOT NULL DEFAULT 1,
        Width REAL,
        Height REAL,
        Area REAL,
        AllowRotation INTEGER NOT NULL DEFAULT 1,
        OrientationAngle REAL DEFAULT 0,
        SourceFilePath TEXT,

        FOREIGN KEY (ProjectId) REFERENCES
CuttingProjects(Id)
    );

    CREATE TABLE IF NOT EXISTS PartPoints (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        PartId INTEGER NOT NULL,
        PointOrder INTEGER NOT NULL,
        X REAL NOT NULL,
        Y REAL NOT NULL,

        FOREIGN KEY (PartId) REFERENCES Parts(Id)
    );
```

Таблиця PartPoints є важливою для роботи з деталями складної форми. Вона дозволяє зберігати контур деталі як послідовність координат вершин. Поле PointOrder визначає порядок обходу точок, що потрібно для правильного формування контуру під час візуалізації, перевірки перетинів і експорту.

Після виконання алгоритму оптимізації потрібно зберегти результат розкрою. Для цього використовуються таблиці CuttingResults і PlacedParts. Перша таблиця містить загальні показники сформованої карти, а друга — координати та кут повороту кожної розміщеної деталі.

SQL-запити для створення таблиць результатів виглядає наступним чином:

```
CREATE TABLE IF NOT EXISTS CuttingResults (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    ProjectId INTEGER NOT NULL,
    MaterialUsagePercent REAL,
    UsedArea REAL,
    WasteArea REAL,
    ExportFilePath TEXT,
    CreatedAt TEXT NOT NULL,

    FOREIGN KEY (ProjectId) REFERENCES
CuttingProjects(Id)
);

CREATE TABLE IF NOT EXISTS PlacedParts (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    CuttingResultId INTEGER NOT NULL,
    PartId INTEGER NOT NULL,
    X REAL NOT NULL,
    Y REAL NOT NULL,
    RotationAngle REAL DEFAULT 0,
    ContourData TEXT,

    FOREIGN KEY (CuttingResultId) REFERENCES
CuttingResults(Id),
    FOREIGN KEY (PartId) REFERENCES Parts(Id)
);
```

Для роботи з даними в програмі використовуються типові операції додавання, отримання, оновлення та видалення записів. Наприклад, під час додавання нового матеріалу користувач вводить його параметри в інтерфейсі, після чого дані передаються до відповідного репозиторію, який виконує SQL-запит INSERT.

Приклад запиту для додавання нового матеріалу виглядає:

```
INSERT INTO Materials (
    Name,
    MaterialType,
    IsRoll,
    Width,
    Length,
```

```
    Thickness,
    TextureDirection,
    Description
)
VALUES (
    @Name,
    @MaterialType,
    @IsRoll,
    @Width,
    @Length,
    @Thickness,
    @TextureDirection,
    @Description
);.
```

У запиті використовуються параметри, а не пряме підставлення значень у текст SQL-команди. Це є правильним підходом, оскільки дозволяє уникнути помилок форматування та підвищує безпеку роботи з даними.

Для отримання списку матеріалів використовується запит SELECT. Отримані дані відображаються в довіднику матеріалів або використовуються під час створення конфігурації проєкту Приклад запиту виглядає наступним чином:

```
SELECT
    Id,
    Name,
    MaterialType,
    IsRoll,
    Width,
    Length,
    Thickness,
    TextureDirection,
    Description
FROM Materials
ORDER BY Name;
```

Під час відкриття проєкту програмі потрібно отримати не тільки сам проєкт, але й пов’язану з ним конфігурацію, матеріал, інструмент і верстат. Для цього можуть використовуватися запити з об’єднанням таблиць.

Приклад запиту для отримання даних проєкту разом із конфігурацією:

```
SELECT
    cp.Id AS ProjectId,
    cp.Name AS ProjectName,
    cp.CreatedAt,
    cp.UpdatedAt,
    pc.Name AS ConfigurationName,
    m.Name AS MaterialName,
```

```
        t.Name AS ToolName,
        mc.Name AS MachineName,
        pc.Tolerance,
        pc.GapBetweenParts,
        pc.AllowRotationByDefault,
        pc.ExportFormat
    FROM CuttingProjects cp
    JOIN ProjectConfigurations pc ON cp.ConfigurationId =
pc.Id
    JOIN Materials m ON pc.MaterialId = m.Id
    JOIN Tools t ON pc.ToolId = t.Id
    JOIN Machines mc ON pc.MachineId = mc.Id
    WHERE cp.Id = @ProjectId;
```

Для збереження результату оптимізації спочатку додається запис у таблицю CuttingResults, а потім для кожної розміщеної деталі створюється запис у таблиці PlacedParts. Такий підхід дозволяє зберігати декілька результатів для одного проєкту та порівнювати їх між собою:

```
INSERT INTO CuttingResults (
    ProjectId,
    MaterialUsagePercent,
    UsedArea,
    WasteArea,
    ExportFilePath,
    CreatedAt
)
VALUES (
    @ProjectId,
    @MaterialUsagePercent,
    @UsedArea,
    @WasteArea,
    @ExportFilePath,
    @CreatedAt
);
```

Дані про фактичне розміщення кожної деталі зберігаються окремо. Це дозволяє повторно відкрити результат, відобразити карту розкрою в інтерфейсі або сформувати файл експорту без повторного запуску алгоритму оптимізації. Запит на збереження розміщеної деталі виглядаэ наступним чином:

```
INSERT INTO PlacedParts (
    CuttingResultId,
    PartId,
    X,
    Y,
    RotationAngle,
```

```
        ContourData
    )
    VALUES (
        @CuttingResultId,
        @PartId,
        @X,
        @Y,
        @RotationAngle,
        @ContourData
    );
```

Для реалізації рівня доступу до даних у C# може використовуватися клас, який відповідає за створення підключення до бази даних. У межах такого класу зберігається рядок підключення, відкривається з'єднання та виконуються SQL-команди. Репозиторії використовують цей клас для виконання конкретних операцій з таблицями.

У спрощеному вигляді структура рівня доступу до даних може містити такі класи:

– DatabaseContext — створення підключення до SQLite;

– DatabaseInitializer — створення таблиць бази даних під час першого запуску;

– MaterialRepository — робота з матеріалами;

– MachineRepository — робота з верстатами;

– ToolRepository — робота з інструментами;

– ProjectRepository — робота з проєктами та конфігураціями;

– PartRepository — робота з деталями та координатами контурів;

– CuttingResultRepository — збереження результатів оптимізації.

Приклад структури класів рівня доступу до даних наведено в таблиці 3.2.

Таким чином, рівень доступу до даних забезпечує взаємодію програмного забезпечення CutMap Designer з базою даних SQLite. Він відповідає за створення структури бази даних, виконання SQL-запитів, збереження довідкових і проєктних даних, а також отримання інформації для подальшої обробки, візуалізації та експорту. Відокремлення цього рівня від інтерфейсу користувача та бізнес-логіки робить програму більш структурованою, зручною для супроводу та подальшого розширення.

Таблиця 3.2 – Класи рівня доступу до даних

№	Клас	Призначення
1	DatabaseContext	Відкриття підключення до бази даних SQLite
2	DatabaseInitializer	Створення таблиць бази даних під час першого запуску
3	MaterialRepository	Додавання, отримання, редагування та видалення матеріалів
4	MachineRepository	Робота з даними про верстати
5	ToolRepository	Робота з даними про інструменти
6	ProjectRepository	Робота з проєктами та конфігураціями
7	PartRepository	Робота з деталями і координатами вершин
8	CuttingResultRepository	Збереження та отримання результатів розкрою

3.3 Реалізація класу взаємодії з користувачем

У програмному забезпеченні CutMap Designer взаємодія користувача із системою реалізується через графічний інтерфейс WPF. Користувач може створювати проєкти розкрою, задавати конфігурацію, працювати з довідниками, вводити деталі, запускати побудову карти розкрою та виконувати експорт результатів.

Для зв’язку між інтерфейсом і програмною логікою використовується підхід MVVM. WPF-вікна відповідають за відображення елементів інтерфейсу, ViewModel-класи обробляють дії користувача, а Model-класи описують основні сутності предметної області. Такий підхід дозволяє відокремити візуальну частину програми від логіки обробки даних.

У межах реалізації інтерфейсу доцільно виділити такі основні ViewModel-класи: MainViewModel, ProjectViewModel, ConfigurationViewModel, MaterialViewModel, ToolViewModel, MachineViewModel, PartViewModel, CuttingMapViewModel та ExportViewModel. Вони відповідають за головне вікно, роботу з проєктами, конфігураціями, довідниками, деталями, картою розкрою та експортом. Загальний вигляд структури проєкту у Visual Studio наведено на рисунку 3.1.

Основним класом взаємодії з головним вікном є MainViewModel. Він містить дані про поточний проєкт, вибрану конфігурацію, список деталей, результат розкрою та команди, що викликаються через інтерфейс: створення й відкриття проєкту, додавання деталей, запуск оптимізації, збереження та експорт.

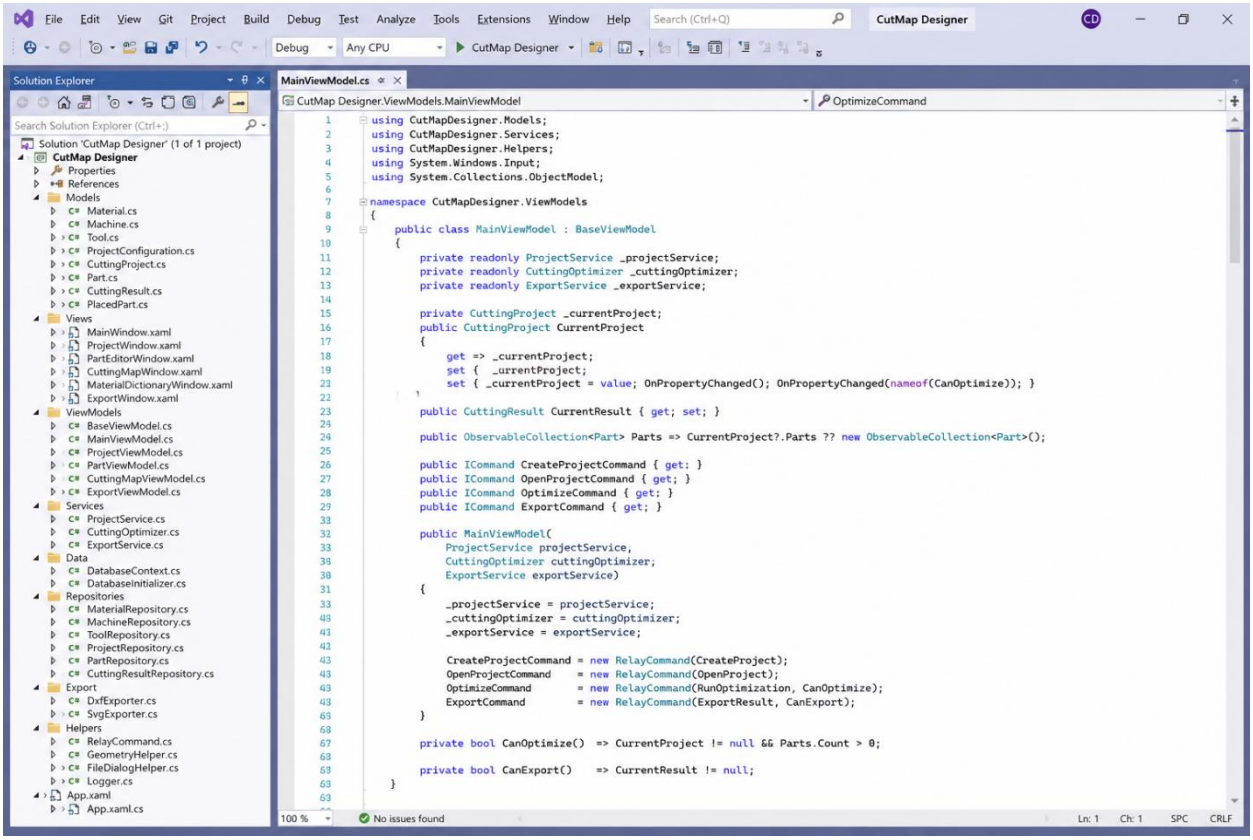

Рисунок 3.1 – Структура проєкту CutMap Designer у середовищі Visual Studio

Фрагмент класу MainViewModel:

```
public class MainViewModel : BaseViewModel
{
    public CuttingProject CurrentProject { get; set; }
    public CuttingResult CurrentResult { get; set; }

    public ICommand CreateProjectCommand { get; }
    public ICommand OpenProjectCommand { get; }
    public ICommand OptimizeCommand { get; }
    public ICommand ExportCommand { get; }

    public MainViewModel(
```

```
            ProjectService projectService,
            CuttingOptimizer cuttingOptimizer,
            ExportService exportService)
        {
            CreateProjectCommand = new
RelayCommand(CreateProject);
            OpenProjectCommand = new
RelayCommand(OpenProject);
            OptimizeCommand = new
RelayCommand(RunOptimization, CanRunOptimization);
            ExportCommand = new RelayCommand(ExportResult,
CanExportResult);
        }

        private bool CanRunOptimization()
        {
            return CurrentProject != null &&
                   CurrentProject.Parts != null &&
                   CurrentProject.Parts.Count > 0;
        }
    }
```

У наведеному фрагменті команди CreateProjectCommand, OpenProjectCommand, OptimizeCommand та ExportCommand пов’язуються з кнопками або пунктами меню. Метод CanRunOptimization перевіряє, чи можна запускати оптимізацію: проєкт має бути створений, а список деталей — не порожній.

Для оновлення інтерфейсу після зміни даних використовується механізм INotifyPropertyChanged, який забезпечує автоматичне оновлення елементів WPF після зміни властивостей у ViewModel програмна реалізація має вигляд:

```
    public class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler
PropertyChanged;

        protected void OnPropertyChanged(string
propertyName)
        {
            PropertyChanged?.Invoke(
                this,
                new
PropertyChangedEventArgs(propertyName));
        }
    }
```

Команди користувача реалізуються через клас RelayCommand, який зв’язує дії в інтерфейсі з методами у ViewModel. Наприклад, кнопка «Оптимізувати» викликає метод запуску алгоритму, а кнопка «Експорт» — метод збереження результату у файл.

Перед запуском оптимізації система перевіряє коректність введених даних: наявність проєкту, конфігурації, списку деталей і допустимість числових параметрів.

```
    private bool ValidateProject()
    {
        if (CurrentProject == null)
            return false;

        if (CurrentProject.Configuration == null)
            return false;

        if (CurrentProject.Parts == null ||
CurrentProject.Parts.Count == 0)
            return false;

        return CurrentProject.Parts.All(p => p.Quantity >
0);
    }
```

Після успішної перевірки ViewModel передає поточний проєкт до сервісу оптимізації. Отриманий результат зберігається у властивості CurrentResult і передається до інтерфейсу для відображення.

```
    private void RunOptimization()
    {
        if (!ValidateProject())
        {
            ShowMessage("Перевірте параметри проєкту та
список деталей.");
            return;
        }

        CurrentResult =
_cuttingOptimizer.BuildCuttingMap(CurrentProject);
        ShowMessage("Карту розкрою успішно сформовано.");
    }
```

Зв’язок між WPF-вікном і ViewModel реалізується через прив’язування даних (див. рис. 3.2). Завдяки цьому інтерфейс не містить складної логіки

обробки, а лише відображає інформацію та передає дії користувача до відповідних команд.

Таким чином, реалізація класів взаємодії з користувачем у CutMap Designer базується на підході MVVM. ViewModel-класи забезпечують зв’язок між WPF-інтерфейсом і бізнес-логікою, обробляють команди користувача, перевіряють вхідні дані, запускають алгоритм оптимізації та передають результати до інтерфейсу.

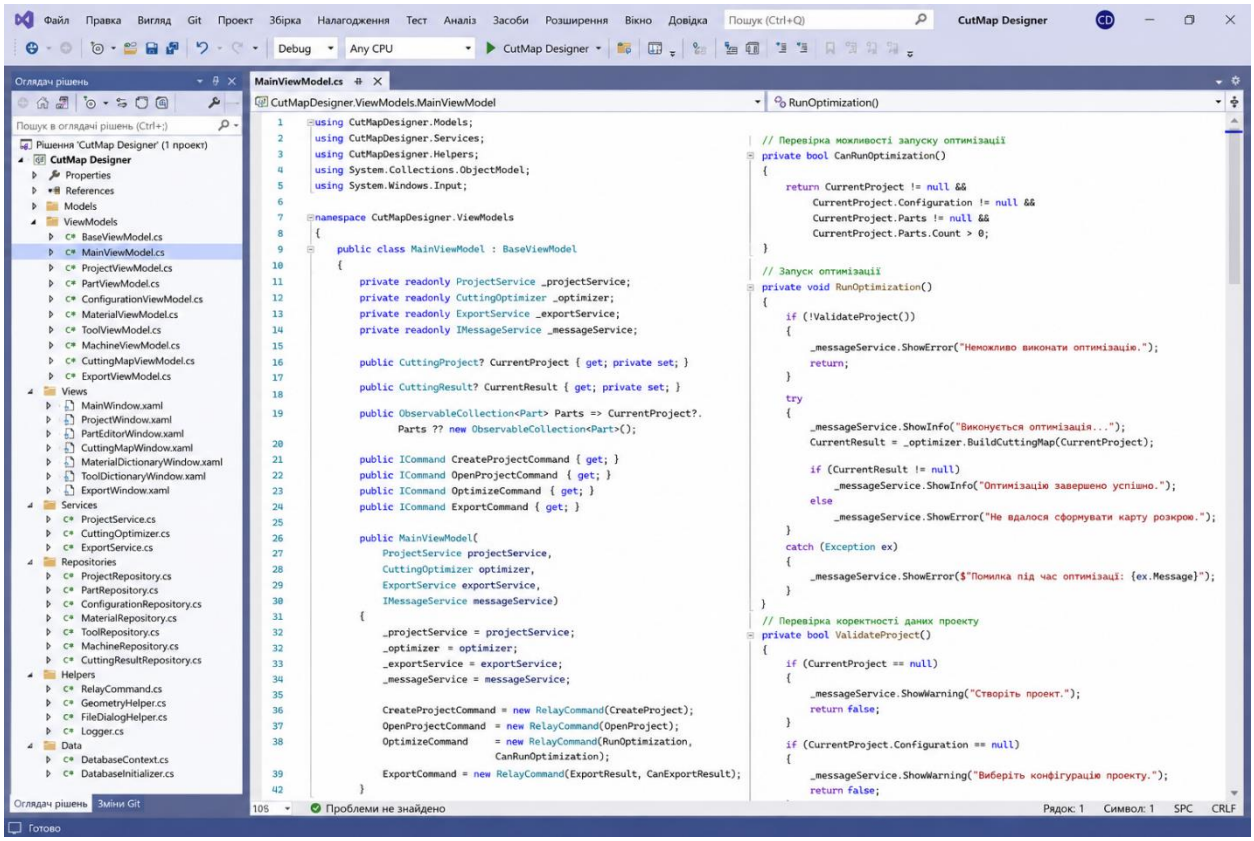

Рисунок 3.2 – Фрагмент програмного коду класу взаємодії з користувачем у Visual Studio

Для зручності користувача в інтерфейсі передбачено повідомлення про результати виконання дій. Якщо операція виконана успішно, програма інформує про це користувача, а в разі помилки вказує, які параметри потрібно виправити. Це зменшує ймовірність некоректного запуску алгоритму та підвищує зручність роботи з програмою.

Зв’язок між WPF-вікном і ViewModel реалізується через прив’язування даних. Наприклад, кнопка запуску оптимізації може бути пов’язана з командою OptimizeCommand, а текстові поля — з властивостями поточного проєкту або конфігурації.

Завдяки такому підходу інтерфейс не містить складної логіки обробки даних, а лише відображає інформацію та передає дії користувача до ViewModel. Перевірка даних, запуск алгоритму, збереження та експорт виконуються на рівні відповідних класів і сервісів.

Таким чином, реалізація взаємодії з користувачем у CutMap Designer базується на підході MVVM. ViewModel-класи забезпечують зв’язок між WPF-інтерфейсом і бізнес-логікою, обробляють команди, перевіряють вхідні дані, запускають оптимізацію та передають результати до інтерфейсу.

3.8 Реалізація експорту результатів

Однією з важливих функцій програмного забезпечення CutMap Designer є експорт сформованої карти розкрою у файл. Це необхідно для подальшого використання результатів у спеціалізованих програмах розкрою, CAD/CAM-системах або засобах перегляду векторної графіки. У межах розробки передбачено підтримку форматів DXF та SVG.

Після завершення оптимізації програма формує набір даних про карту розкрою: розміри матеріалу, координати розміщених деталей, кути повороту, контури деталей і показники ефективності використання матеріалу. Ці дані передаються до модуля експорту, який формує файл у вибраному користувачем форматі.

Для реалізації експорту доцільно виділити окремі класи: ExportService — загальний сервіс експорту, DxfExporter — формування DXF-файлу, SvgExporter — формування SVG-файлу, FileDialogHelper — вибір місця збереження. Такий розподіл дозволяє відокремити логіку експорту від алгоритму розкрою та інтерфейсу користувача.

Формат SVG використовується для збереження карти розкрою у вигляді векторного зображення, яке можна переглядати у браузері або графічному редакторі. Формат DXF призначений для передавання геометрії до CAD/CAM-систем і спеціалізованих програм підготовки розкрою.

Фрагмент методу експорту результату має вигляд:

```
     public void Export(CuttingResult result, string
filePath, ExportFormat format)
     {
         if (result == null)
             throw new
ArgumentNullException(nameof(result));

         if (string.IsNullOrWhiteSpace(filePath))
             throw new ArgumentException("Не задано шлях
збереження файлу.");

         switch (format)
         {
             case ExportFormat.Dxf:
                 _dxfExporter.Export(result, filePath);
                 break;

             case ExportFormat.Svg:
                 _svgExporter.Export(result, filePath);
                 break;

             default:
                 throw new NotSupportedException("Формат
експорту не підтримується.");
         }
     }
```

Перед виконанням експорту програма перевіряє, чи сформовано карту розкрою та чи вибрано шлях збереження файлу. Якщо результат оптимізації відсутній, користувач отримує повідомлення про необхідність спочатку побудувати карту. Після успішного створення файлу система повідомляє про завершення експорту.

Шлях до експортованого файлу зберігається в таблиці CuttingResults, що дозволяє надалі відкрити проєкт і переглянути місце збереження результату. Таким чином, механізм експорту забезпечує практичне використання сформованої карти розкрою у зовнішніх програмних засобах.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Методика тестування програмного забезпечення

Тестування програмного забезпечення CutMap Designer виконувалося з метою перевірки коректності роботи основних функцій, стабільності обробки вхідних даних та правильності формування карти розкрою. Основну увагу було приділено повному сценарію роботи користувача: від створення проєкту до експорту сформованого результату. Такий підхід відповідає процесному розгляду тестування, що охоплює планування, проєктування, виконання та оцінювання результатів перевірки програмного продукту [23, 24].

Методика тестування передбачала перевірку типових дій користувача: створення проєкту, вибір або створення конфігурації, роботу з довідниками матеріалів, інструментів і верстатів, введення деталей, запуск оптимізації, перегляд карти розкрою та експорт результатів. Перевірка наскрізних сценаріїв дає змогу оцінити взаємодію окремих компонентів системи та відповідність програми визначеним функціональним вимогам [23, 24].

Під час тестування перевірялися такі функції: створення і відкриття проєкту, повторне використання конфігурацій, робота з базою SQLite, введення стандартних і складних деталей, перевірка коректності параметрів, формування карти розкрою, розрахунок показників використання матеріалу та експорт у формати DXF і SVG.

Для тестування використовувалися контрольні набори даних із параметрами матеріалу, інструменту, верстата та деталями різної форми. До них входили прямокутні елементи, прості багатокутники та складні деталі, задані координатами вершин. Використання різних наборів вхідних даних дозволяє перевірити поведінку програмного забезпечення за типовими та граничними умовами [24].

Окремо перевірялася обробка помилкових або неповних даних: відсутність матеріалу, порожній список деталей, нульова або від’ємна кількість, некоректні розміри та запуск оптимізації без повної конфігурації. Негативне тестування застосовується для перевірки реакції системи на недопустимі вхідні дані та помилкові дії користувача [24]. У таких випадках програма повинна повідомляти користувача про помилку та не завершувати роботу аварійно.

Критеріями успішного тестування були: коректне створення і збереження проєкту, правильна робота з базою SQLite, відсутність аварійного завершення, розміщення деталей у межах матеріалу без перетинів, урахування зазорів і ширини різу, правильний розрахунок коефіцієнта використання матеріалу та успішний експорт у DXF і SVG. Такі критерії дають змогу оцінити функціональну придатність, надійність і зручність взаємодії з програмним продуктом [15].

Таким чином, обрана методика дозволяє перевірити працездатність CutMap Designer за основними сценаріями використання. Результати тестування основних режимів роботи наведено в наступному підрозділі.

4.2 Тестування основних режимів роботи програми

Тестування основних режимів роботи CutMap Designer виконувалося відповідно до методики, наведеної у підрозділі 4.1. Основна увага приділялася перевірці повного сценарію роботи користувача: створенню проєкту, налаштуванню параметрів, введенню деталей, запуску оптимізації, перегляду карти розкрою та експорту результатів.

Першим етапом було перевірено запуск програми та відображення головного вікна. Воно забезпечує доступ до основних функцій системи: створення або відкриття проєкту, роботу з довідниками, введення деталей, формування карти розкрою та експорт. Загальний вигляд головного вікна наведено на рисунку 4.1. Під час тестування встановлено, що основні функції

згруповані логічно та доступні користувачу в послідовності, зручній для підготовки карти розкрою.

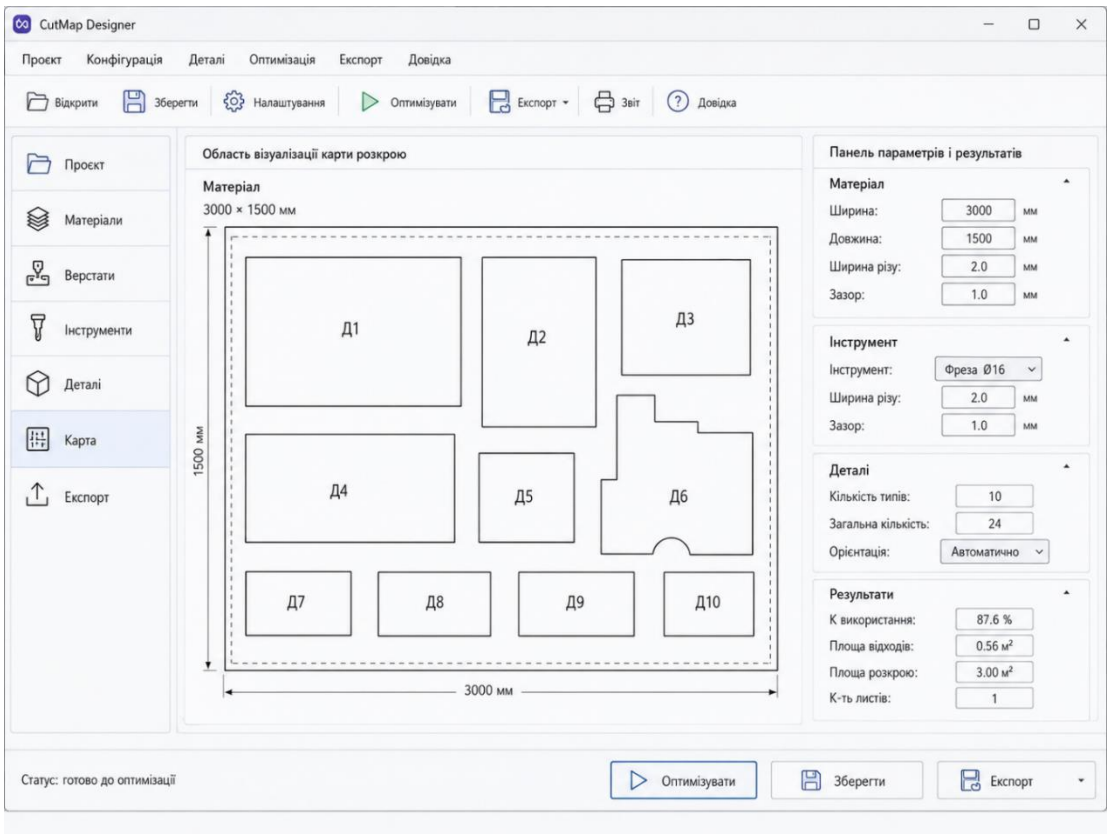

Рисунок 4.1 – Головне вікно програмного забезпечення CutMap Designer

Наступним етапом перевірялася робота з проєктом і конфігурацією. Користувач може створити новий проєкт, вибрати збережену конфігурацію або задати нові параметри матеріалу, інструменту та верстата. Вікно налаштування конфігурації проєкту наведено на рисунку 4.2. Під час тестування перевірялося збереження параметрів, повторне використання конфігурацій та реакція програми на некоректні значення.

Окремо тестувалася робота з довідниками матеріалів, інструментів і верстатів. Перевірялося додавання нових записів, редагування створених параметрів і використання цих даних під час формування конфігурації проєкту. Приклад роботи з довідником матеріалів наведено на рисунку 4.3.

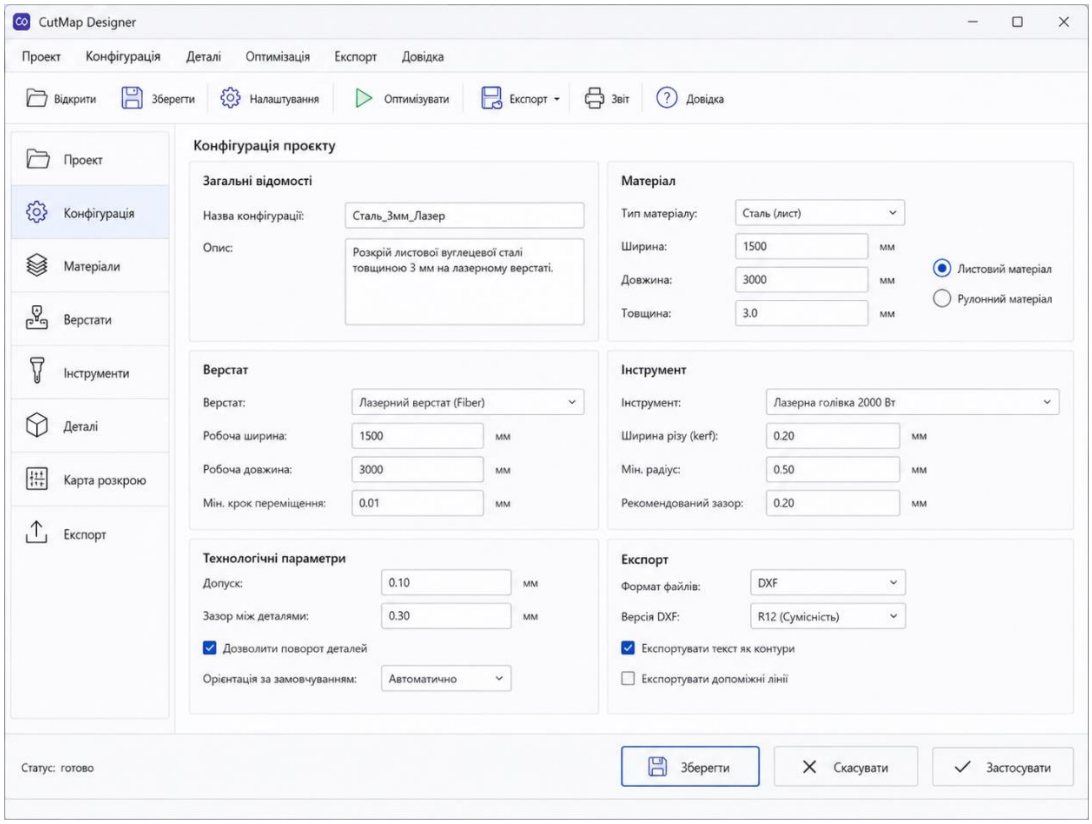

Рисунок 4.2 – Вікно налаштування конфігурації проєкту розкрою

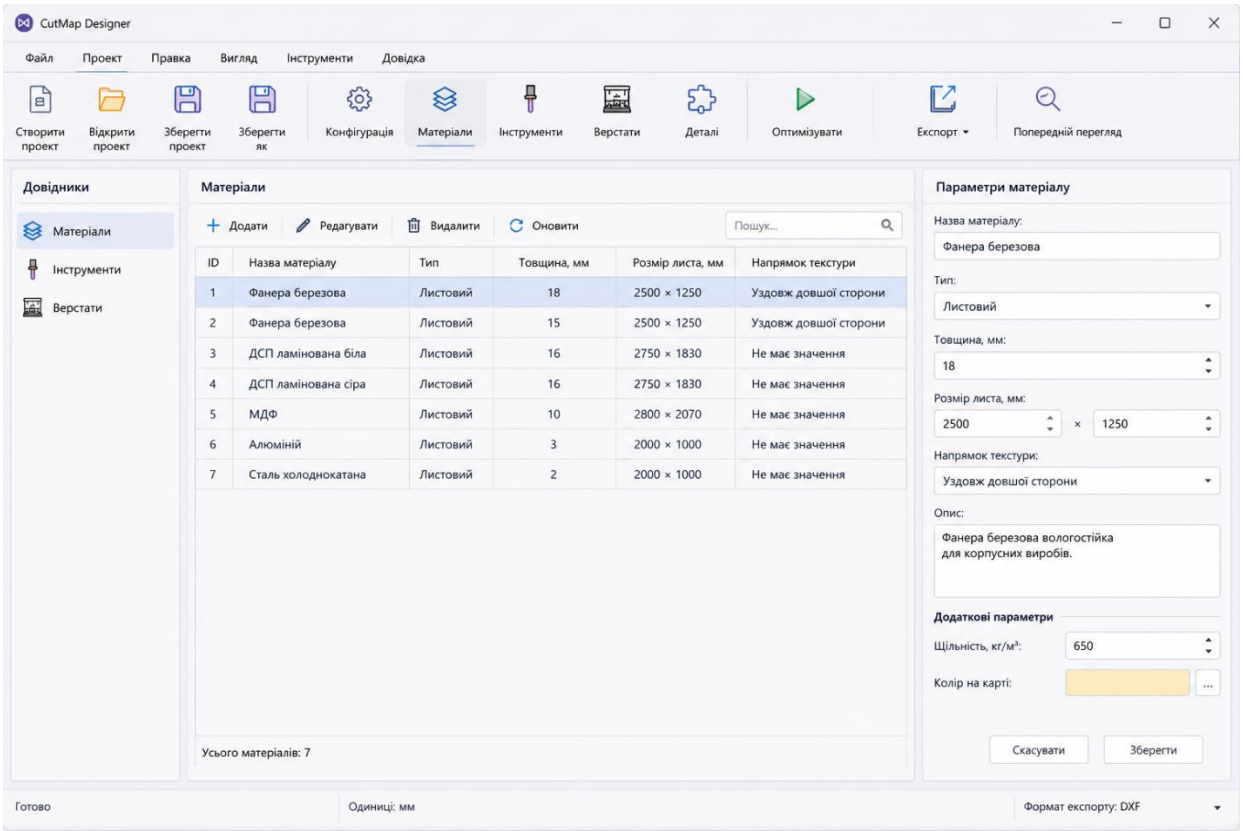

Рисунок 4.3 – Вікно роботи з довідником матеріалів

Після перевірки налаштувань було протестовано введення деталей для розкрою. Програма підтримує введення стандартних деталей за габаритними параметрами, а також деталей складної форми за координатами вершин або шляхом імпорту геометрії з файлу. Вікно введення деталей наведено на рисунку 4.4. У процесі тестування перевірялося збереження кількості деталей, урахування дозволу на поворот, обробка координат складного контуру та реакція програми на помилкові дані.

Після введення коректних даних було виконано запуск алгоритму оптимізації. Програма сформувала карту розкрою з урахуванням параметрів матеріалу, інструменту, технологічних зазорів і обмежень орієнтації. Приклад сформованої карти наведено на рисунку 4.5. Під час перевірки встановлено, що деталі розміщуються в межах матеріалу, не перетинаються між собою, а панель результатів відображає коефіцієнт використання матеріалу, площу відходів і кількість розміщених деталей.

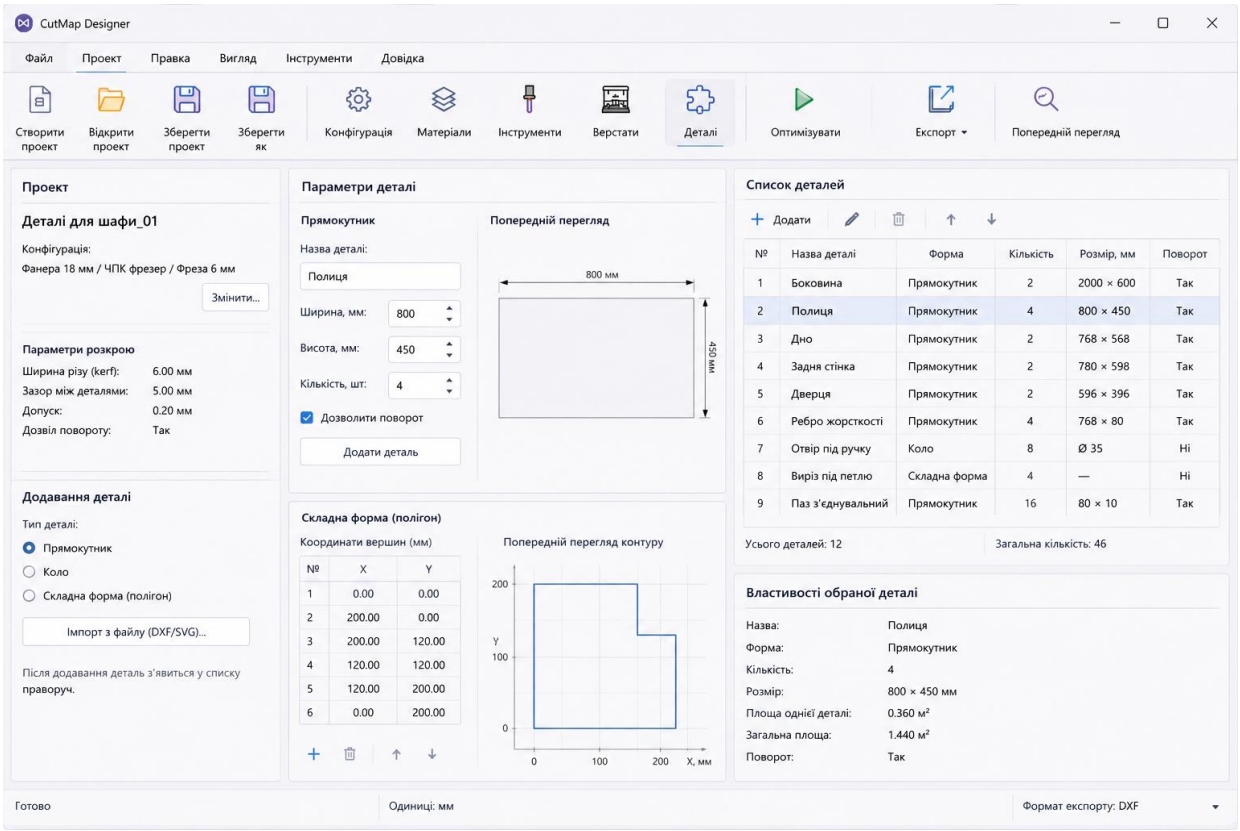

Рисунок 4.4 – Вікно введення деталей для формування карти розкрою

Завершальним етапом була перевірка експорту результатів. Користувач може вибрати формат DXF або SVG, зазначити шлях збереження та сформувати вихідний файл. Вікно експорту наведено на рисунку 4.6. Було перевірено, що у файл передаються контури деталей, їх координати, кут повороту та межі матеріалу, що дозволяє використовувати результат у спеціалізованих програмах розкрою або засобах перегляду векторної графіки.

Таким чином, тестування основних режимів показало, що CutMap Designer забезпечує повний цикл підготовки карти розкрою: створення проєкту, налаштування параметрів, роботу з довідниками, введення деталей, побудову карти та експорт результатів.

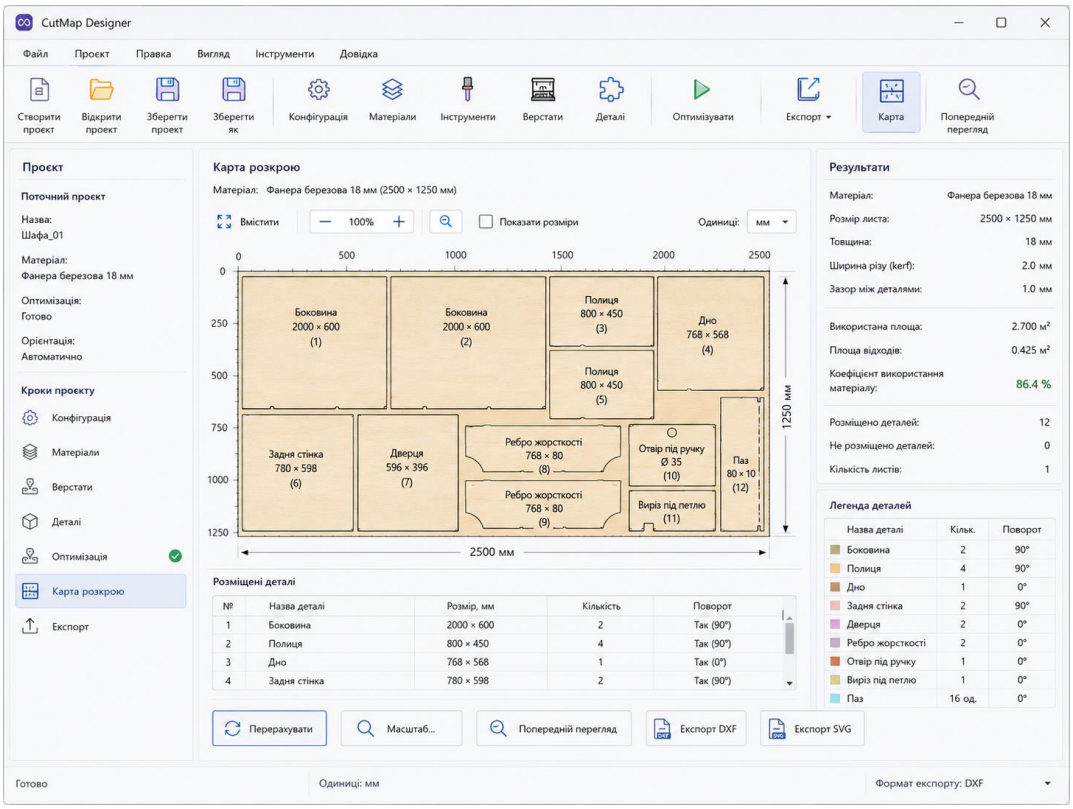

Рисунок 4.5 – Вікно перегляду сформованої карти розкрою

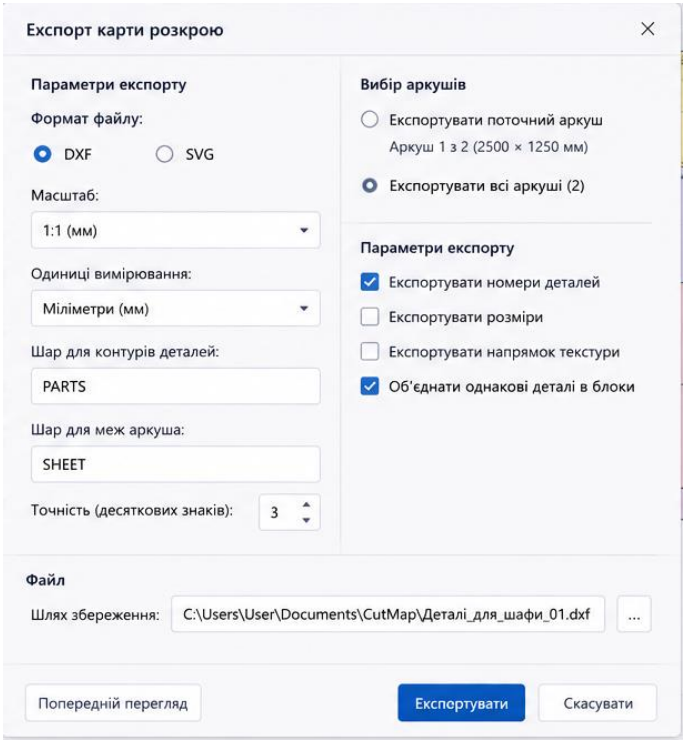

Рисунок 4.6 – Вікно експорту карти розкрою у формат DXF або SVG

Перевірка інтерфейсу підтвердила відповідність основних функцій вимогам, визначеним на етапі проєктування.

4.3 Аналіз результатів формування карти розкрою та експорту даних

Після перевірки основних режимів роботи CutMap Designer було виконано аналіз результатів формування карти розкрою та експорту даних. Метою цього етапу було підтвердження того, що програма не лише коректно приймає вхідні дані, але й формує результат, придатний для подальшої технологічної підготовки виробництва.

Для перевірки було створено тестовий проєкт із заданими параметрами листового матеріалу, інструменту, технологічного зазору, ширини різу та набору деталей. До тестового набору входили деталі різного розміру, зокрема прямокутні елементи та деталі складнішої форми, для яких було задано кількість екземплярів і допустиму орієнтацію.

Після запуску алгоритму оптимізації програма сформувала карту розкрою, у якій деталі розміщені в межах робочої області матеріалу. Під час аналізу перевірялося, чи не виходять деталі за межі матеріалу, чи не перетинаються між собою, чи враховано технологічні зазори та кути повороту. Приклад результату побудови карти розкрою наведено на рисунку 4.7.

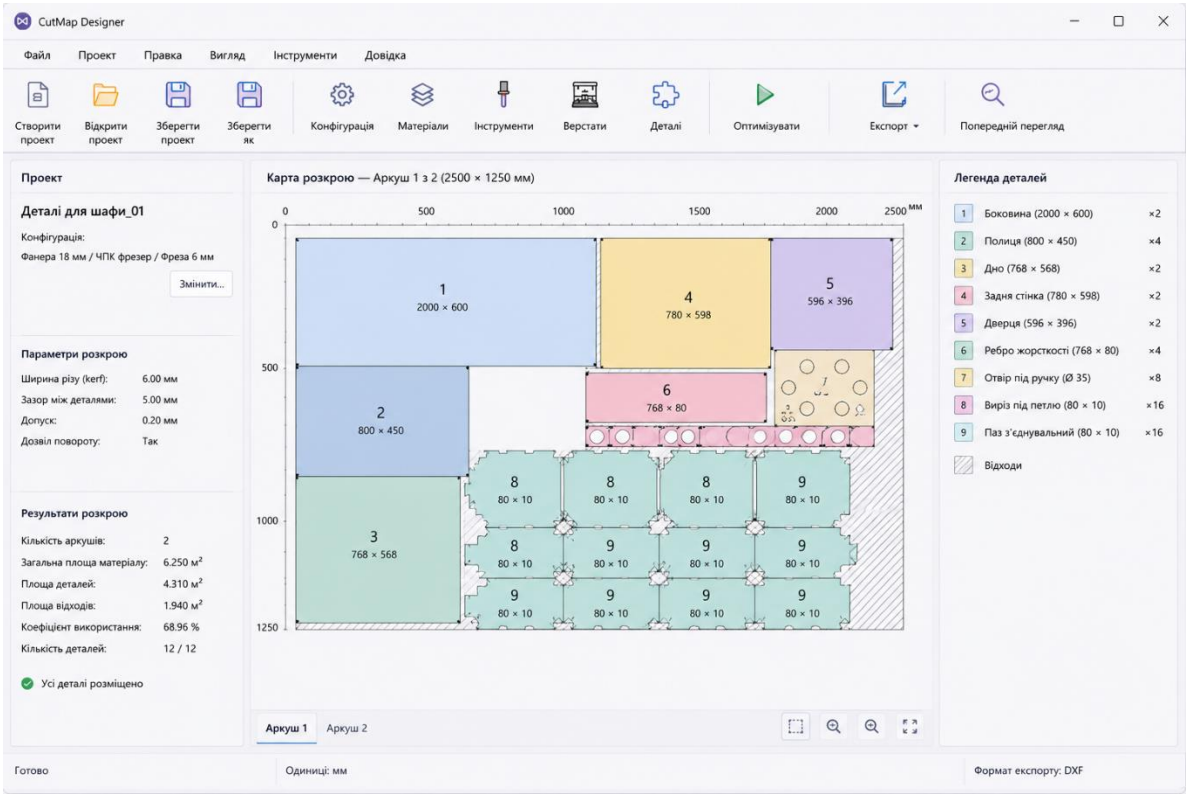

Рисунок 4.7 – Результат побудови карти розкрою в CutMap Designer

Аналіз карти показав, що програма коректно розміщує деталі відповідно до параметрів конфігурації проєкту. Деталі не накладаються одна на одну, а між ними зберігається необхідний технологічний відступ. Це підтверджує працездатність реалізованого алгоритму.

Окрім візуальної перевірки, було проаналізовано числові показники: сумарну площу розміщених деталей, площу використаної області матеріалу, площу відходів і коефіцієнт використання матеріалу. Ці значення дозволяють оцінити ефективність карти розкрою та порівнювати різні варіанти розкладки.

Після формування карти було перевірено експорт результатів. Програма дозволяє зберігати карту у форматах DXF та SVG. Формат SVG зручний для перегляду векторного зображення, а DXF може використовуватися для передавання геометрії до CAD/CAM-систем або спеціалізованих програм розкрою.

Під час експорту перевірялося, чи коректно передаються у файл межі матеріалу, контури деталей, координати їх розташування та кути повороту. Після збереження файл було відкрито для перевірки вмісту. Приклад експортованого результату наведено на рисунку 4.8.

У результаті тестування експорту встановлено, що сформований файл містить основну геометричну інформацію про карту розкрою та може бути використаний для подальшого перегляду або обробки. Це підтверджує практичну цінність програмного забезпечення, оскільки результат роботи не обмежується лише відображенням у вікні програми, а може бути переданий у зовнішні програмні засоби.

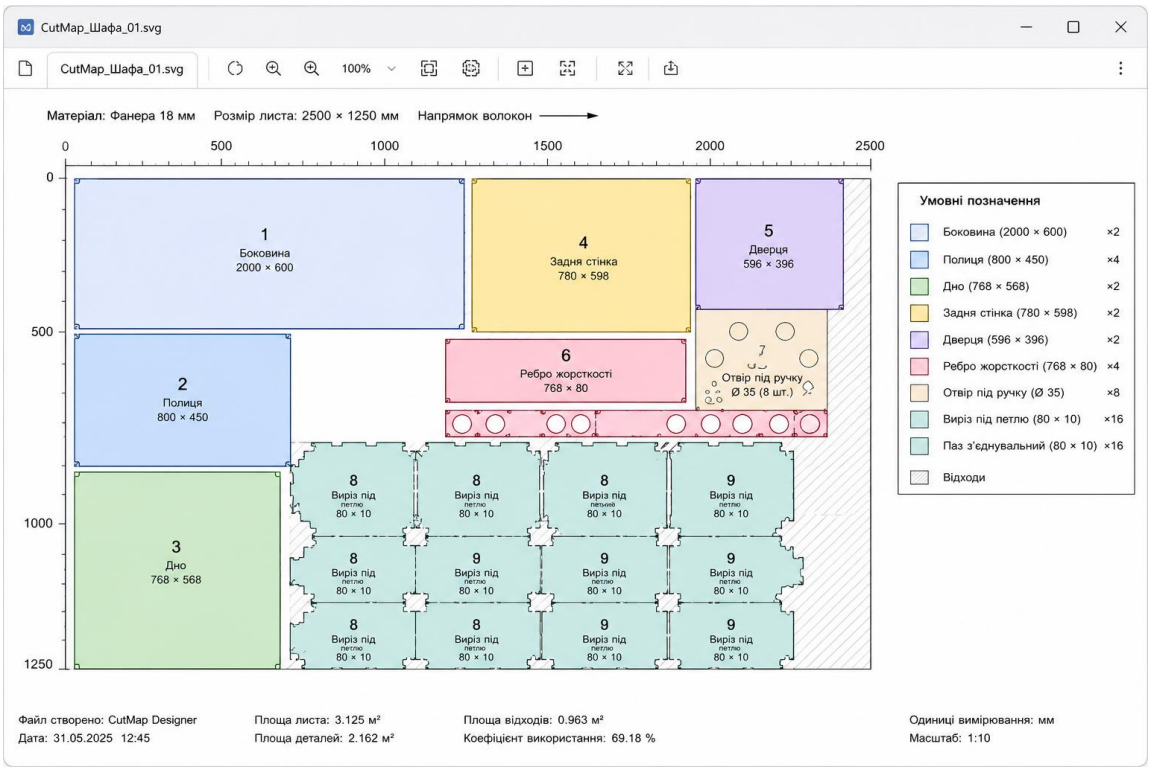

Рисунок 4.8 – Приклад експортованої карти розкрою у форматі SVG

Таким чином, аналіз результатів формування карти розкрою показав, що програмне забезпечення CutMap Designer виконує поставлені функції: формує карту розкрою з урахуванням параметрів матеріалу, інструменту, технологічних зазорів і обмежень орієнтації, розраховує показники ефективності використання матеріалу та забезпечує експорт результатів у векторні формати DXF і SVG. Отримані результати підтверджують працездатність розробленого програмного продукту та його придатність для використання в задачах підготовки розкрою листових і рулонних матеріалів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмне забезпечення CutMap Designer, призначене для оптимізації розкрою листових та рулонних матеріалів в умовах серійного виготовлення продукції. Система забезпечує створення проєктів розкрою, налаштування технологічних параметрів, роботу з довідниками, введення стандартних і складних деталей, формування карти розкрою та експорт результатів у формати DXF і SVG.

Проведено аналіз предметної області, розглянуто особливості розкрою листових і рулонних матеріалів, визначено технологічні параметри, що впливають на побудову карти розкрою: габарити матеріалу, товщину, ширину різу, зазори, допуски, параметри інструменту, характеристики верстата та обмеження на поворот деталей. Розглянуто способи подання стандартних і складних деталей. Стандартні деталі описуються параметрично, а складні — через контурне подання у вигляді координат вершин.

Огляд існуючих програмних засобів показав, що промислові CAD/CAM-системи мають широку функціональність, але можуть бути надмірно складними для окремих задач, а прості онлайн-рішення не завжди підтримують локальне збереження конфігурацій, складні контури та повне врахування технологічних параметрів. Це підтвердило доцільність розробки власного програмного забезпечення.

На етапі проєктування сформульовано функціональні та нефункціональні вимоги до системи. Визначено, що програма повинна підтримувати створення проєктів, роботу з конфігураціями та довідниками, введення й імпорт деталей, формування карти розкрою, візуалізацію результатів і експорт у файли.

У процесі проєктування розроблено функціональну схему програмного забезпечення, архітектуру WPF-застосунку, структуру бази даних SQLite, модель подання геометрії деталей та алгоритм формування карти розкрою. Архітектура побудована за багаторівневим принципом із використанням

підходу MVVM, що дозволило відокремити інтерфейс користувача від бізнес-логіки, рівня доступу до даних, алгоритму оптимізації та модуля експорту.

Алгоритм формування карти розкрою реалізовано як комбінований евристичний підхід. Він передбачає підготовку вхідних даних, формування списку екземплярів деталей, сортування за площею або габаритами, генерацію допустимих орієнтацій, пошук позицій, перевірку меж матеріалу та перетинів між деталями. Результатом роботи алгоритму є карта розкрою з координатами деталей, кутами повороту та показниками використання матеріалу.

У програмному забезпеченні реалізовано рівень доступу до даних, інтерфейс користувача та механізм експорту. База SQLite забезпечує збереження матеріалів, верстатів, інструментів, конфігурацій, проєктів, деталей і результатів розкрою. Інтерфейс WPF містить панель навігації, область візуалізації карти розкрою, панелі параметрів і результатів. Експорт у формати DXF і SVG забезпечує можливість подальшого використання карти у спеціалізованих програмах або CAD/CAM-системах.

На завершальному етапі було виконано тестування програмного забезпечення. Перевірялися створення проєкту, налаштування конфігурації, робота з довідниками, введення деталей, запуск оптимізації, перегляд карти розкрою та експорт результатів. Тестування показало, що система виконує основні функції, коректно обробляє дані, реагує на помилкові або неповні вхідні дані та формує карту розкрою з урахуванням заданих технологічних параметрів.

Таким чином, поставлена мета кваліфікаційної роботи була досягнута. Розроблено програмне забезпечення CutMap Designer, яке забезпечує автоматизацію підготовки карт розкрою листових і рулонних матеріалів, підтримку стандартних і складних деталей, збереження технологічних налаштувань, візуалізацію результатів та експорт даних у поширені векторні формати. Розроблена система може бути основою для подальшого розвитку, зокрема додавання нових алгоритмів оптимізації, розширення форматів імпорту та експорту й покращення роботи зі складними контурами.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Dyckhoff H. A typology of cutting and packing problems. European Journal of Operational Research. 1990. Vol. 44, no. 2. P. 145–159. DOI: 10.1016/0377-2217(90)90350-K.
2. Wäscher G., Haußner H., Schumann H. An improved typology of cutting and packing problems. European Journal of Operational Research. 2007. Vol. 183, no. 3. P. 1109–1130. DOI: 10.1016/j.ejor.2005.12.047.
3. Bennell J. A., Oliveira J. F. A tutorial in irregular shape packing problems. Journal of the Operational Research Society. 2009. Vol. 60, suppl. 1. P. S93–S105. DOI: 10.1057/jors.2008.169.
4. Groover M. P. Fundamentals of Modern Manufacturing: Materials, Processes, and Systems. 7th ed. Hoboken : John Wiley & Sons, 2020. 816 p.
5. Alsaadawy M., Dewidar M., Said A. та ін. A comprehensive review of studying the influence of laser cutting parameters on surface and kerf quality of metals. The International Journal of Advanced Manufacturing Technology. 2024. Vol. 130. P. 1039–1074. DOI: 10.1007/s00170-023-12768-1.
6. de Berg M., Cheong O., van Kreveld M., Overmars M. Computational Geometry: Algorithms and Applications. 3rd ed. Berlin ; Heidelberg : Springer, 2008. 386 p. DOI: 10.1007/978-3-540-77974-
7. SigmaNEST. Best Nesting Software [Електронний ресурс]. URL: https://www.sigmanest.com/en/sigmanest (дата звернення: 09.06.2026).
8. ProNest CNC Part Nesting Software for Plasma, Laser, Waterjet and Oxyfuel [Електронний ресурс] / Hypertherm Associates. URL: https://www.hypertherm.com/hypertherm/pronest/pronest-cadcam-nesting-software/ (дата звернення: 09.06.2026).
9. CutList Optimizer : online panel cutting optimization software [Електронний ресурс]. URL: https://www.cutlistoptimizer.com/ (дата звернення: 09.06.2026).

10. Deepnest : open source nesting application [Електронний ресурс]. URL: https://github.com/Jack000/Deepnest (дата звернення: 09.06.2026).
11. Lodi A., Martello S., Monaci M. Two-dimensional packing problems: a survey. European Journal of Operational Research. 2002. Vol. 141, no. 2. P. 241–252. DOI: 10.1016/S0377-2217(02)00123-6.
12. Hopper E., Turton B. C. H. A review of the application of meta-heuristic algorithms to 2D strip packing problems. Artificial Intelligence Review. 2001. Vol. 16. P. 257–300. DOI: 10.1023/A:1012590107280.
13. Blum C., Puchinger J., Raidl G. R., Roli A. Hybrid metaheuristics in combinatorial optimization: a survey. Applied Soft Computing. 2011. Vol. 11, no. 6. P. 4135–4151. DOI: 10.1016/j.asoc.2011.02.032.
14. Sommerville I. Software Engineering. 10th ed. Boston : Pearson, 2016. 816 p.
15. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмних засобів. Вимоги до якості систем і програмних засобів та її оцінювання (SQuaRE). Моделі якості системи та програмних засобів. Чинний від 2018-01-01. Київ : УкрНДНЦ, 2017.
16. What is Windows Presentation Foundation [Електронний ресурс] / Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/ (дата звернення: 09.06.2026).
17. About SQLite [Електронний ресурс] / SQLite. URL: https://www.sqlite.org/about.html (дата звернення: 09.06.2026).
18. Codd E. F. A relational model of data for large shared data banks. Communications of the ACM. 1970. Vol. 13, no. 6. P. 377–387. DOI: 10.1145/362384.362685.
19. C# language documentation [Електронний ресурс] / Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 09.06.2026).
20. Designing the infrastructure persistence layer [Електронний ресурс] / Microsoft. URL: https://learn.microsoft.com/en-

us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design (дата звернення: 09.06.2026).

21. C# language documentation [Електронний ресурс] / Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 09.06.2026).
22. Designing the infrastructure persistence layer [Електронний ресурс] / Microsoft. URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-design (дата звернення: 09.06.2026).
23. ISO/IEC/IEEE 29119-2:2021. Software and systems engineering — Software testing — Part 2: Test processes. Geneva : International Organization for Standardization, 2021.
24. Certified Tester Foundation Level Syllabus. Version 4.0.1 [Електронний ресурс] / International Software Testing Qualifications Board. 2024. URL: https://istqb.org/?download_id=3345&sdm_process_download=1 (дата звернення: 09.06.2026).

Додаток А

Лістинг програми

Файл Models/Part.cs

```
using System.Collections.Generic;

namespace CutMapDesigner.Models
{
    public enum ShapeType
    {
        Rectangle,
        Circle,
        Polygon,
        CustomContour,
        ImportedContour
    }

    public class Part
    {
        public int Id { get; set; }

        public int ProjectId { get; set; }

        public string Name { get; set; }

        public ShapeType ShapeType { get; set; }

        public int Quantity { get; set; }

        public double Width { get; set; }

        public double Height { get; set; }

        public double Area { get; set; }

        public bool AllowRotation { get; set; }

        public double OrientationAngle { get; set; }

        public string SourceFilePath { get; set; }

        public List<PartPoint> Points { get; set; }

        public Part()
        {
            Quantity = 1;
            AllowRotation = true;
            Points = new List<PartPoint>();
        }
```

```
    }
}
```

Файл Models/ProjectConfiguration.cs

```
using System;

namespace CutMapDesigner.Models
{
    public class ProjectConfiguration
    {
        public int Id { get; set; }

        public string Name { get; set; }

        public Material Material { get; set; }

        public Machine Machine { get; set; }

        public Tool Tool { get; set; }

        public double Tolerance { get; set; }

        public double GapBetweenParts { get; set; }

        public bool AllowRotationByDefault { get; set; }

        public ExportFormat ExportFormat { get; set; }

        public DateTime CreatedAt { get; set; }

        public ProjectConfiguration()
        {
            AllowRotationByDefault = true;
            ExportFormat = ExportFormat.Dxf;
            CreatedAt = DateTime.Now;
        }
    }

    public enum ExportFormat
    {
        Dxf,
        Svg
    }
}
```

Файл Data/DatabaseContext.cs

```
using Microsoft.Data.Sqlite;

namespace CutMapDesigner.Data
{
```

```
    public class DatabaseContext
    {
        private readonly string _connectionString;

        public DatabaseContext(string databasePath)
        {
            _connectionString = $"Data Source={databasePath}";
        }

        public SqliteConnection CreateConnection()
        {
            var connection = new
SqliteConnection(_connectionString);
            connection.Open();

            using var command = connection.CreateCommand();
            command.CommandText = "PRAGMA foreign_keys = ON;";
            command.ExecuteNonQuery();

            return connection;
        }
    }
}
```

Файл Data/DatabaseInitializer.cs

```
using Microsoft.Data.Sqlite;

namespace CutMapDesigner.Data
{
    public class DatabaseInitializer
    {
        private readonly DatabaseContext _context;

        public DatabaseInitializer(DatabaseContext context)
        {
            _context = context;
        }

        public void Initialize()
        {
            using var connection = _context.CreateConnection();

            CreateMaterialsTable(connection);
            CreateMachinesTable(connection);
            CreateToolsTable(connection);
            CreateProjectConfigurationsTable(connection);
            CreateCuttingProjectsTable(connection);
            CreatePartsTable(connection);
            CreatePartPointsTable(connection);
            CreateCuttingResultsTable(connection);
            CreatePlacedPartsTable(connection);
```

```
        }

        private void CreateMaterialsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS Materials (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                Name TEXT NOT NULL,
                MaterialType TEXT,
                IsRoll INTEGER NOT NULL DEFAULT 0,
                Width REAL NOT NULL,
                Length REAL,
                Thickness REAL,
                TextureDirection TEXT,
                Description TEXT
            );";

            command.ExecuteNonQuery();
        }

        private void CreateMachinesTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS Machines (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                Name TEXT NOT NULL,
                MachineType TEXT,
                WorkAreaWidth REAL,
                WorkAreaLength REAL,
                MinMoveStep REAL,
                Description TEXT
            );";

            command.ExecuteNonQuery();
        }

        private void CreateToolsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS Tools (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
                Name TEXT NOT NULL,
                ToolType TEXT,
                KerfWidth REAL NOT NULL,
                MinCutRadius REAL,
                DefaultGap REAL,
                Description TEXT
            );";

            command.ExecuteNonQuery();
        }

        private void
CreateProjectConfigurationsTable(SqliteConnection connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS ProjectConfigurations (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                Name TEXT NOT NULL,
                MaterialId INTEGER NOT NULL,
                MachineId INTEGER NOT NULL,
                ToolId INTEGER NOT NULL,
                Tolerance REAL DEFAULT 0,
                GapBetweenParts REAL DEFAULT 0,
                AllowRotationByDefault INTEGER NOT NULL DEFAULT
1,
                ExportFormat TEXT DEFAULT 'DXF',
                CreatedAt TEXT NOT NULL,

                FOREIGN KEY (MaterialId) REFERENCES
Materials(Id),
                FOREIGN KEY (MachineId) REFERENCES Machines(Id),
                FOREIGN KEY (ToolId) REFERENCES Tools(Id)
            );";

            command.ExecuteNonQuery();
        }

        private void CreateCuttingProjectsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS CuttingProjects (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                Name TEXT NOT NULL,
                ConfigurationId INTEGER NOT NULL,
                CreatedAt TEXT NOT NULL,
                UpdatedAt TEXT,
```

```
                Comment TEXT,

                FOREIGN KEY (ConfigurationId)
                REFERENCES ProjectConfigurations(Id)
            );";

            command.ExecuteNonQuery();
        }

        private void CreatePartsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS Parts (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                ProjectId INTEGER NOT NULL,
                Name TEXT NOT NULL,
                ShapeType TEXT NOT NULL,
                Quantity INTEGER NOT NULL DEFAULT 1,
                Width REAL,
                Height REAL,
                Area REAL,
                AllowRotation INTEGER NOT NULL DEFAULT 1,
                OrientationAngle REAL DEFAULT 0,
                SourceFilePath TEXT,

                FOREIGN KEY (ProjectId) REFERENCES
CuttingProjects(Id)
            );";

            command.ExecuteNonQuery();
        }

        private void CreatePartPointsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS PartPoints (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                PartId INTEGER NOT NULL,
                PointOrder INTEGER NOT NULL,
                X REAL NOT NULL,
                Y REAL NOT NULL,

                FOREIGN KEY (PartId) REFERENCES Parts(Id)
            );";
```

```
            command.ExecuteNonQuery();
        }

        private void CreateCuttingResultsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS CuttingResults (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                ProjectId INTEGER NOT NULL,
                MaterialUsagePercent REAL,
                UsedArea REAL,
                WasteArea REAL,
                ExportFilePath TEXT,
                CreatedAt TEXT NOT NULL,

                FOREIGN KEY (ProjectId) REFERENCES
CuttingProjects(Id)
            );";

            command.ExecuteNonQuery();
        }

        private void CreatePlacedPartsTable(SqliteConnection
connection)
        {
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            CREATE TABLE IF NOT EXISTS PlacedParts (
                Id INTEGER PRIMARY KEY AUTOINCREMENT,
                CuttingResultId INTEGER NOT NULL,
                PartId INTEGER NOT NULL,
                X REAL NOT NULL,
                Y REAL NOT NULL,
                RotationAngle REAL DEFAULT 0,
                ContourData TEXT,

                FOREIGN KEY (CuttingResultId)
                REFERENCES CuttingResults(Id),

                FOREIGN KEY (PartId) REFERENCES Parts(Id)
            );";

            command.ExecuteNonQuery();
        }
    }
}
```

Файл Repositories/MaterialRepository.cs

```
using System.Collections.Generic;
using CutMapDesigner.Data;
using CutMapDesigner.Models;
using Microsoft.Data.Sqlite;

namespace CutMapDesigner.Repositories
{
    public class MaterialRepository
    {
        private readonly DatabaseContext _context;

        public MaterialRepository(DatabaseContext context)
        {
            _context = context;
        }

        public void Add(Material material)
        {
            using var connection = _context.CreateConnection();
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            INSERT INTO Materials (
                Name,
                MaterialType,
                IsRoll,
                Width,
                Length,
                Thickness,
                TextureDirection,
                Description
            )
            VALUES (
                @Name,
                @MaterialType,
                @IsRoll,
                @Width,
                @Length,
                @Thickness,
                @TextureDirection,
                @Description
            );";

            command.Parameters.AddWithValue("@Name",
material.Name);
            command.Parameters.AddWithValue("@MaterialType",
material.MaterialType);
```

```
            command.Parameters.AddWithValue("@IsRoll",
material.IsRoll ? 1 : 0);
            command.Parameters.AddWithValue("@Width",
material.Width);
            command.Parameters.AddWithValue("@Length",
material.Length);
            command.Parameters.AddWithValue("@Thickness",
material.Thickness);
            command.Parameters.AddWithValue("@TextureDirection",
material.TextureDirection);
            command.Parameters.AddWithValue("@Description",
material.Description);

            command.ExecuteNonQuery();
        }

        public List<Material> GetAll()
        {
            var materials = new List<Material>();

            using var connection = _context.CreateConnection();
            using var command = connection.CreateCommand();

            command.CommandText =
            @"
            SELECT
                Id,
                Name,
                MaterialType,
                IsRoll,
                Width,
                Length,
                Thickness,
                TextureDirection,
                Description
            FROM Materials
            ORDER BY Name;";

            using SqliteDataReader reader =
command.ExecuteReader();

            while (reader.Read())
            {
                materials.Add(new Material
                {
                    Id = reader.GetInt32(0),
                    Name = reader.GetString(1),
                    MaterialType = reader.GetString(2),
                    IsRoll = reader.GetInt32(3) == 1,
                    Width = reader.GetDouble(4),
                    Length = reader.IsDBNull(5) ? 0 :
reader.GetDouble(5),
```

```
                    Thickness = reader.IsDBNull(6) ? 0 :
reader.GetDouble(6),
                    TextureDirection = reader.IsDBNull(7) ? "" :
reader.GetString(7),
                    Description = reader.IsDBNull(8) ? "" :
reader.GetString(8)
                });
            }

            return materials;
        }
    }
}
```

Файл Services/GeometryService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using CutMapDesigner.Models;

namespace CutMapDesigner.Services
{
    public class GeometryService
    {
        public List<PartPoint> BuildContour(Part part)
        {
            if (part.ShapeType == ShapeType.Rectangle)
                return BuildRectangleContour(part.Width,
part.Height);

            if (part.ShapeType == ShapeType.CustomContour ||
                part.ShapeType == ShapeType.ImportedContour)
                return part.Points.OrderBy(p =>
p.PointOrder).ToList();

            return new List<PartPoint>();
        }

        public List<PartPoint> BuildRectangleContour(
            double width,
            double height)
        {
            return new List<PartPoint>
            {
                new PartPoint { PointOrder = 1, X = 0, Y = 0 },
                new PartPoint { PointOrder = 2, X = width, Y = 0
},
                new PartPoint { PointOrder = 3, X = width, Y =
height },
```

```
                new PartPoint { PointOrder = 4, X = 0, Y =
height }
            };
        }

        public double CalculatePolygonArea(List<PartPoint>
points)
        {
            if (points == null || points.Count < 3)
                return 0;

            double area = 0;

            for (int i = 0; i < points.Count; i++)
            {
                PartPoint current = points[i];
                PartPoint next = points[(i + 1) % points.Count];

                area += current.X * next.Y - next.X * current.Y;
            }

            return Math.Abs(area) / 2.0;
        }

        public BoundingBox GetBoundingBox(List<PartPoint>
points)
        {
            return new BoundingBox
            {
                MinX = points.Min(p => p.X),
                MinY = points.Min(p => p.Y),
                MaxX = points.Max(p => p.X),
                MaxY = points.Max(p => p.Y)
            };
        }

        public List<PartPoint> Rotate(
            List<PartPoint> points,
            double angleDegrees)
        {
            double angle = angleDegrees * Math.PI / 180.0;
            double cos = Math.Cos(angle);
            double sin = Math.Sin(angle);

            return points.Select(p => new PartPoint
            {
                PointOrder = p.PointOrder,
                X = p.X * cos - p.Y * sin,
                Y = p.X * sin + p.Y * cos
            }).ToList();
        }

        public List<PartPoint> Move(
```

```
            List<PartPoint> points,
            double offsetX,
            double offsetY)
        {
            return points.Select(p => new PartPoint
            {
                PointOrder = p.PointOrder,
                X = p.X + offsetX,
                Y = p.Y + offsetY
            }).ToList();
        }
    }
}
```

Файл Models/BoundingBox.cs

```
namespace CutMapDesigner.Models
{
    public class BoundingBox
    {
        public double MinX { get; set; }

        public double MinY { get; set; }

        public double MaxX { get; set; }

        public double MaxY { get; set; }

        public double Width => MaxX - MinX;

        public double Height => MaxY - MinY;

        public bool Intersects(BoundingBox other)
        {
            return !(MaxX < other.MinX ||
                     MinX > other.MaxX ||
                     MaxY < other.MinY ||
                     MinY > other.MaxY);
        }
    }
}
```

Файл Services/CuttingOptimizer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using CutMapDesigner.Models;

namespace CutMapDesigner.Services
{
```

```
    public class CuttingOptimizer
    {
        private readonly GeometryService _geometryService;

        public CuttingOptimizer(GeometryService geometryService)
        {
            _geometryService = geometryService;
        }

        public CuttingResult BuildCuttingMap(CuttingProject
project)
        {
            ValidateProject(project);

            var material = project.Configuration.Material;
            var placedParts = new List<PlacedPart>();

            var partInstances =
CreatePartInstances(project.Parts);

            partInstances = partInstances
                .OrderByDescending(p => p.Area)
                .ThenByDescending(p => p.Width * p.Height)
                .ToList();

            foreach (var part in partInstances)
            {
                bool placed = TryPlacePart(
                    part,
                    material,
                    project.Configuration,
                    placedParts);

                if (!placed)
                {
                    part.IsPlaced = false;
                }
            }

            return BuildResult(project, placedParts);
        }

        private void ValidateProject(CuttingProject project)
        {
            if (project == null)
                throw new
ArgumentNullException(nameof(project));

            if (project.Configuration == null)
                throw new InvalidOperationException("Не задано
конфігурацію.");
```

```
            if (project.Parts == null || project.Parts.Count ==
0)
                throw new InvalidOperationException("Список
деталей порожній.");
        }

        private List<Part> CreatePartInstances(List<Part> parts)
        {
            var result = new List<Part>();

            foreach (var part in parts)
            {
                for (int i = 0; i < part.Quantity; i++)
                {
                    result.Add(part);
                }
            }

            return result;
        }

        private bool TryPlacePart(
            Part part,
            Material material,
            ProjectConfiguration configuration,
            List<PlacedPart> placedParts)
        {
            var orientations = GetAllowedOrientations(part,
configuration);

            foreach (double angle in orientations)
            {
                for (double y = 0; y <= material.Length; y += 5)
                {
                    for (double x = 0; x <= material.Width; x +=
5)
                    {
                        if (CanPlace(part, x, y, angle,
material, placedParts))
                        {
                            placedParts.Add(new PlacedPart
                            {
                                Part = part,
                                X = x,
                                Y = y,
                                RotationAngle = angle
                            });

                            return true;
                        }
                    }
                }
            }
```

```
            return false;
        }

        private List<double> GetAllowedOrientations(
            Part part,
            ProjectConfiguration configuration)
        {
            if (!part.AllowRotation)
                return new List<double> { part.OrientationAngle
};

            if (!configuration.AllowRotationByDefault)
                return new List<double> { 0 };

            return new List<double> { 0, 90, 180, 270 };
        }

        private bool CanPlace(
            Part part,
            double x,
            double y,
            double angle,
            Material material,
            List<PlacedPart> placedParts)
        {
            var contour = _geometryService.BuildContour(part);
            var rotated = _geometryService.Rotate(contour,
angle);
            var moved = _geometryService.Move(rotated, x, y);
            var box = _geometryService.GetBoundingBox(moved);

            if (box.MinX < 0 || box.MinY < 0)
                return false;

            if (box.MaxX > material.Width || box.MaxY >
material.Length)
                return false;

            foreach (var placed in placedParts)
            {
                var placedContour =
_geometryService.BuildContour(placed.Part);
                var placedRotated = _geometryService.Rotate(
                    placedContour,
                    placed.RotationAngle);

                var placedMoved = _geometryService.Move(
                    placedRotated,
                    placed.X,
                    placed.Y);
```

```
                var placedBox =
_geometryService.GetBoundingBox(placedMoved);

                if (box.Intersects(placedBox))
                    return false;
            }

            return true;
        }

        private CuttingResult BuildResult(
            CuttingProject project,
            List<PlacedPart> placedParts)
        {
            double materialArea =
                project.Configuration.Material.Width *
                project.Configuration.Material.Length;

            double usedArea = placedParts.Sum(p => p.Part.Area);
            double usage = materialArea > 0
                ? usedArea / materialArea * 100.0
                : 0;

            return new CuttingResult
            {
                ProjectId = project.Id,
                PlacedParts = placedParts,
                UsedArea = usedArea,
                WasteArea = materialArea - usedArea,
                MaterialUsagePercent = usage,
                CreatedAt = DateTime.Now
            };
        }
    }
}
```

Файл ViewModels/MainViewModel.cs

```
using System.Windows.Input;
using CutMapDesigner.Models;
using CutMapDesigner.Services;

namespace CutMapDesigner.ViewModels
{
    public class MainViewModel : BaseViewModel
    {
        private readonly CuttingOptimizer _cuttingOptimizer;
        private readonly ExportService _exportService;

        private CuttingProject _currentProject;
        private CuttingResult _currentResult;
```

```
        public CuttingProject CurrentProject
        {
            get => _currentProject;
            set
            {
                _currentProject = value;
                OnPropertyChanged(nameof(CurrentProject));
            }
        }

        public CuttingResult CurrentResult
        {
            get => _currentResult;
            set
            {
                _currentResult = value;
                OnPropertyChanged(nameof(CurrentResult));
            }
        }

        public ICommand CreateProjectCommand { get; }

        public ICommand OpenProjectCommand { get; }

        public ICommand OptimizeCommand { get; }

        public ICommand ExportCommand { get; }

        public MainViewModel(
            CuttingOptimizer cuttingOptimizer,
            ExportService exportService)
        {
            _cuttingOptimizer = cuttingOptimizer;
            _exportService = exportService;

            CreateProjectCommand = new
RelayCommand(CreateProject);
            OpenProjectCommand = new RelayCommand(OpenProject);
            OptimizeCommand = new RelayCommand(RunOptimization,
CanRunOptimization);
            ExportCommand = new RelayCommand(ExportResult,
CanExportResult);
        }

        private bool CanRunOptimization()
        {
            return CurrentProject != null &&
                   CurrentProject.Parts != null &&
                   CurrentProject.Parts.Count > 0;
        }

        private bool CanExportResult()
```

```
        {
            return CurrentResult != null &&
                   CurrentResult.PlacedParts.Count > 0;
        }

        private void CreateProject()
        {
            CurrentProject = new CuttingProject
            {
                Name = "Новий проєкт розкрою"
            };
        }

        private void OpenProject()
        {
            // Завантаження проєкту виконується через
ProjectRepository.
        }

        private void RunOptimization()
        {
            if (!CanRunOptimization())
                return;

            CurrentResult =
_cuttingOptimizer.BuildCuttingMap(CurrentProject);
        }

        private void ExportResult()
        {
            if (!CanExportResult())
                return;

            _exportService.Export(
                CurrentResult,
                "cutting_map.svg",
                ExportFormat.Svg);
        }
    }
}
```

Файл ViewModels/BaseViewModel.cs

```
using System.ComponentModel;

namespace CutMapDesigner.ViewModels
{
    public class BaseViewModel : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler
PropertyChanged;
```

```
        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(
                this,
                new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Файл ViewModels/RelayCommand.cs

```
using System;
using System.Windows.Input;

namespace CutMapDesigner.ViewModels
{
    public class RelayCommand : ICommand
    {
        private readonly Action _execute;
        private readonly Func<bool> _canExecute;

        public RelayCommand(
            Action execute,
            Func<bool> canExecute = null)
        {
            _execute = execute;
            _canExecute = canExecute;
        }

        public bool CanExecute(object parameter)
        {
            return _canExecute == null || _canExecute();
        }

        public void Execute(object parameter)
        {
            _execute();
        }

        public event EventHandler CanExecuteChanged;
    }
}
```

Файл Services/ExportService.cs

```
using System;
using CutMapDesigner.Models;

namespace CutMapDesigner.Services
{
    public class ExportService
    {
```

```
        private readonly DxfExporter _dxfExporter;
        private readonly SvgExporter _svgExporter;

        public ExportService(
            DxfExporter dxfExporter,
            SvgExporter svgExporter)
        {
            _dxfExporter = dxfExporter;
            _svgExporter = svgExporter;
        }

        public void Export(
            CuttingResult result,
            string filePath,
            ExportFormat format)
        {
            if (result == null)
                throw new ArgumentNullException(nameof(result));

            if (string.IsNullOrWhiteSpace(filePath))
                throw new ArgumentException(
                    "Не задано шлях збереження файлу.");

            switch (format)
            {
                case ExportFormat.Dxf:
                    _dxfExporter.Export(result, filePath);
                    break;

                case ExportFormat.Svg:
                    _svgExporter.Export(result, filePath);
                    break;

                default:
                    throw new NotSupportedException(
                        "Формат експорту не підтримується.");
            }
        }
    }
}
```

Файл Services/SvgExporter.cs

```
using System.Globalization;
using System.IO;
using System.Text;
using CutMapDesigner.Models;

namespace CutMapDesigner.Services
{
    public class SvgExporter
    {
```

```
        private readonly GeometryService _geometryService;

        public SvgExporter(GeometryService geometryService)
        {
            _geometryService = geometryService;
        }

        public void Export(CuttingResult result, string
filePath)
        {
            var builder = new StringBuilder();

            builder.AppendLine("<?xml version=\"1.0\"
encoding=\"UTF-8\"?>");
            builder.AppendLine("<svg
xmlns=\"http://www.w3.org/2000/svg\" " +
                               "version=\"1.1\">");

            foreach (var placedPart in result.PlacedParts)
            {
                var contour =
_geometryService.BuildContour(placedPart.Part);
                var rotated = _geometryService.Rotate(
                    contour,
                    placedPart.RotationAngle);

                var moved = _geometryService.Move(
                    rotated,
                    placedPart.X,
                    placedPart.Y);

                builder.Append("<polygon points=\"");

                foreach (var point in moved)
                {
                    builder.Append(

point.X.ToString(CultureInfo.InvariantCulture));
                    builder.Append(",");
                    builder.Append(

point.Y.ToString(CultureInfo.InvariantCulture));
                    builder.Append(" ");
                }

                builder.AppendLine("\" " +
                    "fill=\"none\" stroke=\"black\" stroke-
width=\"1\" />");
            }

            builder.AppendLine("</svg>");

            File.WriteAllText(filePath, builder.ToString());
```

```
        }
    }
}
```

Файл Services/DxfExporter.cs

```
using System.Globalization;
using System.IO;
using System.Text;
using CutMapDesigner.Models;

namespace CutMapDesigner.Services
{
    public class DxfExporter
    {
        private readonly GeometryService _geometryService;

        public DxfExporter(GeometryService geometryService)
        {
            _geometryService = geometryService;
        }

        public void Export(CuttingResult result, string
filePath)
        {
            var builder = new StringBuilder();

            builder.AppendLine("0");
            builder.AppendLine("SECTION");
            builder.AppendLine("2");
            builder.AppendLine("ENTITIES");

            foreach (var placedPart in result.PlacedParts)
            {
                var contour =
_geometryService.BuildContour(placedPart.Part);
                var rotated = _geometryService.Rotate(
                    contour,
                    placedPart.RotationAngle);

                var moved = _geometryService.Move(
                    rotated,
                    placedPart.X,
                    placedPart.Y);

                for (int i = 0; i < moved.Count; i++)
                {
                    var current = moved[i];
                    var next = moved[(i + 1) % moved.Count];
```

```
                    AddLine(builder, current.X, current.Y,
next.X, next.Y);
                }
            }

            builder.AppendLine("0");
            builder.AppendLine("ENDSEC");
            builder.AppendLine("0");
            builder.AppendLine("EOF");

            File.WriteAllText(filePath, builder.ToString());
        }

        private void AddLine(
            StringBuilder builder,
            double x1,
            double y1,
            double x2,
            double y2)
        {
            builder.AppendLine("0");
            builder.AppendLine("LINE");
            builder.AppendLine("8");
            builder.AppendLine("CUT_MAP");

            builder.AppendLine("10");
            builder.AppendLine(ToDxfValue(x1));
            builder.AppendLine("20");
            builder.AppendLine(ToDxfValue(y1));

            builder.AppendLine("11");
            builder.AppendLine(ToDxfValue(x2));
            builder.AppendLine("21");
            builder.AppendLine(ToDxfValue(y2));
        }

        private string ToDxfValue(double value)
        {
            return value.ToString(CultureInfo.InvariantCulture);
        }
    }
}
```