МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ЗАСТОСУНОК АВТОМАТИЗАЦІЇ РОЗРАХУНКІВ ПОКРІВЕЛЬНИХ РОБІТ З ЕЛЕМЕНТАМИ 3D-МОДЕЛЮВАННЯ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ ВЛАДИСЛАВ СІДЕЙ

«___»_____ 2026 р.

Керівник PhD старший викладач _____ ІГОР КАНДИБА

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили

(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Сідей Владислав Михайлович

((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Застосунок автоматизації розрахунків покрівельних робіт з елементами 3D-моделювання».

Керівник роботи: Кандиба Ігор Олександрович, старший викладач кафедри ІПЗ, PhD.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні:

програмний застосунок для автоматизації розрахунку матеріалів та вартості покрівельних робіт; база даних матеріалів з можливістю оновлення цін через API; 3D-візуалізація даху з можливістю перегляду з різних ракурсів; формування підсумкового кошторису.

4. Перелік питань, що підлягають розробці: аналіз існуючих систем автоматизації будівельних розрахунків; огляд методів 3D-моделювання для покрівельних конструкцій; проєктування архітектури застосунку; реалізація розрахунку матеріалів кроквяної системи та покрівлі; створення бази даних цін на матеріали з можливістю оновлення; розробка REST API для доступу до даних матеріалів; реалізація 3D-візуалізації даху; тестування застосунку та аналіз отриманих результатів.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ ІГОР КАНДИБА

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ ВЛАДИСЛАВ СІДЕЙ

(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «24» грудня 2025

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Застосунок автоматизації розрахунків покрівельних робіт з елементами 3D-моделювання

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	Виконано
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд існуючих систем автоматизації будівельних розрахунків та аналогічних програмних продуктів	31.01.2026	01.03.2026	Виконано
4	Огляд існуючих методів 3D-моделювання та технологій візуалізації для вирішення поставленої задачі	02.03.2026	01.04.2026	Виконано
5	Реалізація обраних технологій (C#, WPF, Helix Toolkit, Web API) з аналізом отриманих результатів	02.04.2026	24.05.2026	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	Виконано
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	Виконано

Керівник роботи _____ ІГОР КАНДИБА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ ВЛАДИСЛАВ СІДЕЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувача групи 401 ЧНУ ім. Петра Могили

Сідей Владислав Михайлович

на тему: **«ЗАСТОСУНОК АВТОМАТИЗАЦІЇ РОЗРАХУНКІВ ПОКРІВЕЛЬНИХ РОБІТ З ЕЛЕМЕНТАМИ 3D-МОДЕЛЮВАННЯ»**

Актуальність кваліфікаційної роботи полягає у зростаючій потребі будівельної галузі в зручних засобах автоматизації розрахунків матеріалів та вартості робіт. Розробка застосунку для автоматизованого розрахунку покрівельних робіт з елементами 3D-моделювання є актуальною, оскільки дозволяє підвищити точність і швидкість підготовки кошторисів, а також забезпечити наочність проєктованих конструкцій.

Об'єктом роботи є процес визначення матеріалів і розрахунку вартості будівництва дахів з використанням сучасних вебтехнологій.

Предметом роботи є методи проєктування систем автоматизованих розрахунків покрівельних робіт та їх 3D-візуалізації.

Метою роботи є створення програмного забезпечення, що виконує розрахунок матеріалів, формує кошторис та забезпечує 3D-візуалізацію даху для підвищення точності та швидкості проєктування.

В результаті виконання роботи було проаналізовано існуючі аналоги (ArchiCAD, SketchUp, онлайн-калькулятори), обрано оптимальний технологічний стек (.NET 8, WPF, WebView2, Three.js). Розроблено робочий прототип програмного забезпечення для настільних ПК, який включає 2D-редактор для малювання плану будинку, автоматичний розрахунок геометричних параметрів (ширина, довжина, площа), підтримку різних типів дахів (двосхилий, вальмовий, мансардний,) та 3D-візуалізацію створеної моделі. Реалізовано інтеграцію із зовнішнім API для отримання актуальних цін на матеріали та експорт кошторису у формати Excel та PDF.

Кваліфікаційна робота складається з чотирьох розділів. У першому розділі проведено аналіз предметної області та огляд аналогів, сформовано постановку задачі. Другий розділ присвячений проєктуванню архітектури системи та опису методів

розрахунків. У третьому розділі наведено результати реалізації застосунку, демонстрацію роботи та тестування. В четвертому – аналіз отриманих результатів та перспективи подальшого розвитку.

Загальний обсяг роботи – 105 сторінок. Кваліфікаційна робота містить 3 додаток, 22 рисунків, 8 таблиць і 30 джерел посилання.

Ключові слова: автоматизація будівельних розрахунків, кошторис, 3D-моделювання даху, WPF, WebView2, Three.js.

ABSTRACT

to the qualification work by the student of the group 401 of Petro Mohyla Black Sea National University

SIDEI VLADYSLAV

«APPLICATION FOR AUTOMATION OF ROOFING WORKS CALCULATIONS WITH 3D MODELING ELEMENTS»

The relevance of this work lies in the growing need of the construction industry in convenient means of automating the calculation of materials and the cost of work. The development of an application for the automated calculation of roofing with 3D modeling elements is relevant, as it allows to increase the accuracy and speed of preparing estimates, as well as to ensure the clarity of the designed structures.

The object of the work is the process of determining the materials and cost of building roofs using modern web technologies.

The subject of the work is the methods of designing systems of automated calculations of roofing works and their 3D-visualization.

The purpose of the work is to create software that calculates materials, generates estimates and provides 3D visualization of the roof to improve the accuracy and speed of design.

As a result of the work, the existing analogues (ArchiCAD, SketchUp, online calculators) were analyzed, the optimal technological stack (.NET 8, WPF, WebView2, Three.js) was chosen. A working prototype of software for desktop PCs has been developed, which includes a 2D editor for drawing a house plan, automatic calculation of geometric parameters (width, length, area), support for various types of roofs (gable, hip, attic, tent) and 3D visualization of the created model. Implemented integration with an external API to obtain current prices for materials and export estimates in Excel and PDF formats.

This work consists of four sections. In the first section, an analysis of the subject area and a review of analogues are carried out, a statement of the problem is formed. The second

section is devoted to the design of the system architecture and description of calculation methods. The third section shows the results of the application implementation, demonstration of work and testing. In the fourth – an analysis of the results and prospects for further development.

The total amount of work is 105 pages. The qualification paper contains 3 appendix, 22 figures, 8 tables and 30 reference sources.

Keywords: automation of construction calculations, estimate, 3D-modeling of the roof, WPF, WebView2, Three.js.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОКАЗНИКИ

2D – Двовимірний простір

3D – Інтерфейс програмування застосунків

JSON – Нотація об’єктів JavaScript

PDF – Портативний формат документів

REST – Передача стану представлення

SQL – Структурована мова запитів

WPF – Фундамент представлень Windows

API – Application Programming Interface

BIM – Building Information Modeling

HTML – HyperText Markup Language

HTTP – HyperText Transfer Protocol

IDE – Integrated Development Environment

JSON – JavaScript Object Notation

MVVM – Model-View-ViewModel

PDF – Portable Document Format

RAM – Random Access Memory

REST – Representational State Transfer

SPA – Single Page Application

SQL – Structured Query Language

TCP – Transmission Control Protocol

UI – User Interface

WPF – Windows Presentation Foundation

XAML – Extensible Application Markup Language

ВСТУП

Кваліфікаційну роботу виконано на кафедрі інтелектуальних інформаційних систем Чорноморського національного університету імені Петра Могили. Тема кваліфікаційної роботи – «Застосунок автоматизації розрахунків покрівельних робіт з елементами 3D-моделювання».

Сьогодні будівельна галузь активно шукає доступні та ефективні цифрові інструменти. Проєктування даху залишається одним з найбільш технічно складних етапів будівництва будинку. Помилки в ручних розрахунках призводять до надмірного споживання матеріалу, збільшення фінансових витрат і структурних питань. Хоча професійне архітектурне програмне забезпечення (наприклад, ArchiCAD або SketchUp) існує, воно часто занадто дороге, вимагає тижнів навчання і перевантажене функціональністю. З іншого боку, прості онлайн-калькулятори безкоштовні і швидкі, але вони не підтримують довільні будівельні форми і повністю не мають 3D-візуалізації, що робить процес проєктування неінтуїтивним. Тому створення простого, вільного і візуального інструменту, який поєднує в собі можливість складання нестандартного плану будинку з автоматичним розрахунком матеріалу і візуалізацією 3D-моделі, є актуальним і практично значущим завданням.

Метою роботи є створення програмного забезпечення, що виконує розрахунок матеріалів, формує кошторис та забезпечує 3D-візуалізацію даху для підвищення точності та швидкості проєктування.

Завдання:

- аналіз існуючих систем автоматизації будівельних розрахунків та виявлення їх переваг і недоліків;
- вибір стека технологій для крос-платформної настільної програми;
- проєктування архітектури системи і створення 2D-редактора для складання плану будівлі;

– реалізація механізму автоматичного розрахунку геометричних параметрів покрівлі (площа, довжина коника, довжина карниза, кут нахилу);

– розробка 3D-модуля візуалізації на базі бібліотеки Three.js;

– інтеграція із зовнішнім API для отримання поточних цін на матеріали та формування детальної оцінки;

– тестування розробленого застосування та аналіз отриманих результатів.

Об’єктом роботи є процес визначення матеріалів і розрахунку вартості будівництва дахів з використанням сучасних вебтехнологій.

Предметом роботи є методи проєктування систем автоматизованих розрахунків покрівельних робіт та їх 3D-візуалізації.

Декларація про використання ШІ. Під час підготовки наукової роботи (академічного тексту) було використано інструмент Gemini 3.1 Pro. Відповідно до таксономії GAIDeT (2025), наведені нижче завдання були делеговані інструментам генеративного ШІ за повного людського нагляду:

– Оцінювання здійсненності та ризиків
– Оптимізація коду
– Перевірка відтворюваності
– Вичитування та редагування
– Резюмування тексту
– Адаптація та коригування емоційного тону
– Оцінювання якості
– Рекомендації

Повну відповідальність за фінальний рукопис несуть автор.

Інструменти генеративного ШІ не зазначаються як автори та не несуть відповідальності за кінцеві результати.

Декларацію подав: Сідей Владислав Михайлович

1 АНАЛІЗ ЗАСОБІВ АВТОМАТИЗАЦІЇ БУДІВЕЛЬНИХ РОЗРАХУНКІВ

1.1 Основні поняття та визначення у сфері автоматизації будівельних розрахунків

Перш ніж перейти до аналізу існуючих програмних рішень, необхідно визначити ключові терміни, які складають концептуальну основу даної роботи.

Програмне забезпечення для настільних ПК – це ПЗ, яке встановлюється та виконується локально на комп’ютері користувача під керуванням операційної системи (наприклад, Windows). На відміну від веб-застосунків, таке ПЗ має прямий доступ до системних ресурсів, файлової системи та апаратного забезпечення, що забезпечує вищу продуктивність та роботу без підключення до мережі.

Автоматизація покрівельних робіт – процес виконання розрахунків за допомогою програмного забезпечення без ручного втручання. На основі параметрів введення користувача (розміри будівлі, тип даху, кут нахилу) програма автоматично обчислює площу даху, довжину крокви, кількість аркушів матеріалу та приблизну вартість. Автоматизація зменшує людську помилку і значно прискорює процес проєктування.

3D моделювання в будівництві – створення тривимірного цифрового представлення будівлі або її частин. У контексті покрівлі 3D моделювання дозволяє користувачеві візуально оглянути спроєктований будинок під будь-яким кутом до початку фактичного будівництва. Це допомагає виявити потенційні проблеми на ранній стадії та робить процес проєктування більш інтуїтивно зрозумілим.

Дахова фермова система – несуча конструкція, що складається з похилих крокв, які підтримують покриття даху. Крокви переносять вагу даху на стіни будівлі. Довжина та відстань крокв є критичними параметрами, які необхідно точно розрахувати для забезпечення безпеки конструкції.

Двосхилий дах – тип даху з двома похилими поверхнями (сторонами), які зустрічаються на центральному хребті. Це найпоширеніший тип даху для житлових будинків завдяки простому дизайну та ефективному водовідведенню.

Вальмовий дах – тип даху, де всі чотири сторони нахилені вниз до стін. Вальмові дахи не мають вертикальних кінців (фронтонів), що робить їх більш стійкими в регіонах з сильним вітром.

Мансардний дах – тип даху з двома схилами з кожного боку, де нижній схил крутіший за верхній. Така конструкція допускає додаткову житлову площу (мансардні кімнати) під дахом.

WebView [1] – компонент Microsoft, який дозволяє розробникам вбудовувати сучасні веб-технології (HTML, CSS, JavaScript) у власні настільні програми. Він використовує механізм Chromium, який дозволяє використовувати розширені бібліотеки веб-графіки, такі як Three.js, у програмі WPF.

Three.js [2] – кросбраузерна бібліотека JavaScript та інтерфейс прикладного програмування (API), що використовується для створення та відображення анімованої 3D комп’ютерної графіки у веббраузері. Three.js s абстрагує складність WebGL, дозволяючи розробникам легко створювати сцени, камери, світло та 3D-об’єкти.

Архітектура клієнт-сервер – обчислювальна модель, де центральний сервер розміщує, доставляє та керує більшістю ресурсів і послуг, які вимагає клієнтське програмне забезпечення. У цьому проекті проста клієнт-серверна модель використовується для отримання цін на матеріали із зовнішнього API, але основна функціональність (малювання та обчислення) реалізована на стороні клієнта.

WPF[3] (Windows Presentation Foundation) – структура інтерфейсу користувача, розроблена Microsoft для створення настільних клієнтських програм у Windows. WPF використовує XAML (Extensible Application Markup Language) для розробки інтерфейсів користувача та підтримує розділення проблем між дизайном інтерфейсу

користувача та бізнес-логікою за допомогою таких шаблонів, як MVVM (Model-View-ViewModel).

1.2 Огляд існуючих аналогів

Для формулювання технічних вимог до майбутнього програмного продукту та виявлення недоліків існуючих рішень проведено детальний аналіз діючих програмних засобів, що використовуються для проєктування будинків та покрівельних розрахунків. Це дослідження зосереджено не лише на оцінці поточного стану ринку, але й на визначенні архітектурних рішень, сильних і слабких сторін існуючих продуктів.

Сьогодні існують численні програмні рішення для проєктування будівель, починаючи від професійних архітектурних систем і закінчуючи простими онлайн -калькуляторами. Найбільш актуальними для аналізу є ArchiCAD, SketchUp, онлайн -калькулятори покрівлі та запропонований застосунок.

ArchiCAD

ArchiCAD – це професійне програмне забезпечення BIM (Building Information Modeling), розроблене Graphisoft. Широко використовується архітекторами і дизайнерами для створення детальних 2D і 3D моделей будівель.

Таблиця 1.1 – Опис ArchiCAD

Назва	ArchiCAD
Архітектура	Монолітна платформа BIM з інтегрованими інструментами моделювання та документації
Тип програми	Професійне програмне забезпечення BIM для архітекторів і дизайнерів

Кінець таблиці 1.1

Мови програмування	C++, C# (API для плагінів)
Базовий функціонал	2D-креслення, 3D-моделювання, автоматична генерація документації, зльоти матеріалів
Розширені функції	1) Вбудований механізм візуалізації (CineRender); 2) Інструменти енергетичної оцінки; 3) Особливості ремонту та реконструкції; 4) Співпраця в командній роботі через BIMcloud
Платформа	Windows, macOS
Ціни	Платні (на основі підписки, починаючи приблизно з 1500 доларів США на рік)
Переваги	1) Повний робочий процес BIM з інтелектуальними будівельними компонентами; 2) Автоматичне оновлення в усіх видах (плани поверхів, секції, висоти); 3) Велика бібліотека будівельних матеріалів та об’єктів
Недоліки	1) Дуже дорого для окремих користувачів або малого бізнесу; 2) Крута крива навчання – вимагає тижнів навчання; 3) Надмірна функціональність для простих покрівельних розрахунків; 4) Не оптимізовано для швидких розрахунків, пов’язаних із конкретними завданнями
Веб-сайт	https://graphisoft.com [4]

Рисунок 1.1 – Приклад інтерфейсу ArchiCAD [4]

SketchUp

SketchUp – це програмне забезпечення для 3D-моделювання, що належить Trimble In Воно відоме своїм інтуїтивно зрозумілим робочим процесом "push-pull" і користується популярністю серед архітекторів, дизайнерів інтер’єрів і любителів.

Таблиця 1.2 – Опис SketchUp

Назва	SketchUp
Архітектура	Ядро 3D моделювання з розширеннями для архітектурного проєктування
Тип програми	Програмне забезпечення для 3D-моделювання для архітектури, дизайну інтер’єру та будівництва
Мови програмування	C++, Ruby (для розширень/плагінів)
Базовий функціонал	3D-моделювання з технологією push-pull, моделювання на основі компонентів, застосування матеріалів

Кінець таблиці 1.2

Розширені функції	1) 3D Склад (онлайн бібліотека готових моделей); 2) Верстка (інструмент документації 2D); 3) Extension Warehouse (плагіни для додаткової функціональності); 4) Перегляд моделі VR/AR
Платформа	Windows, macOS, Web (SketchUp Free)
Ціни	Безкоштовно (обмежена веб-версія), Pro версія (приблизно $299/рік)
Переваги	1) Дуже інтуїтивно зрозумілий і простий у вивченні; 2) Велика спільнота та велика бібліотека безкоштовних моделей; 3) Розширюється за допомогою плагінів Ruby
Недоліки	1) Не спеціалізується на розрахунках покрівлі – немає автоматизованих зльотів матеріалів; 2) Моделювання даху вимагає ручного креслення кожного схилу; 3) Немає вбудованих формул для довжини крокви або кількості листа; 4) Реалістичний рендеринг вимагає додаткових плагінів
Веб-сайт	https://sketchup.com [5]

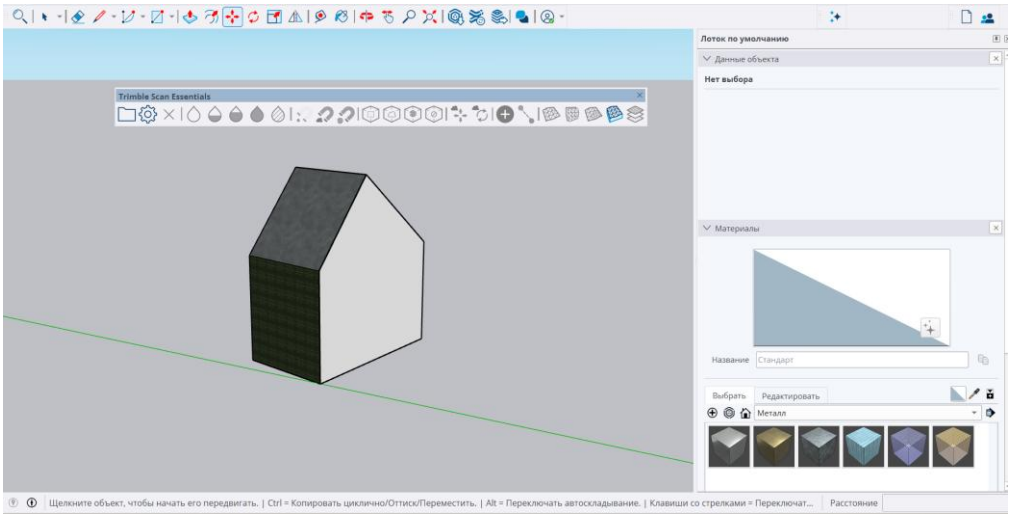

Рисунок 1.2 – Приклад інтерфейсу SketchUp [5]

Roofing Calculator

Існує багато веб-інструментів, призначених виключно для розрахунків покрівлі. Приклади включають «RoofCalc.org», «MyRoofCalculator.com» і калькулятори на веб-сайтах будівельних поставок.

Таблиця 1.3 – Опис онлайн-калькуляторів покрівлі

Назва	Roofing Calculator
Архітектура	Веб-клієнт-сервер (інтерфейс HTML/CSS/JavaScript, простий сервер для оновлення цін)
Тип програми	Спеціальне веб-інструмент для швидкої оцінки покрівельного матеріалу
Мови програмування	JavaScript, HTML, CSS (frontend); PHP, Python, or Node.js (backend)

Кінець таблиці 1.3

Базовий функціонал	Обчисліть площу даху на основі вхідних розмірів, оцініть кількість черепиці/листів
Розширені функції	1) Розрахунок ціни на основі типу матеріалу; 2) Прості діаграми (тільки 2D, рідко 3D); 3) Деякі пропонують звіти у форматі PDF
Платформа	Веб-браузер (кросплатформний)
Ціни	Зазвичай вільний (з підтримкою реклами) або вільний
Переваги	1) Встановлення не потрібне; 2) Дуже простий і швидкий для основних розрахунків; 3) Часто вільно використовувати
Недоліки	1) Немає 3D візуалізації – користувач не може бачити будинок; 2) Жоден настінний малюнок – користувач не може налаштувати форму будинку; 3) Обмежено стандартними прямокутними будинками; 4) Не може впоратися зі складними типами даху (мансарда, стегно, піраміда); 5) Проблеми конфіденційності (можуть бути зібрані вхідні дані)
Веб-сайт	https://roofcalc.org [6]

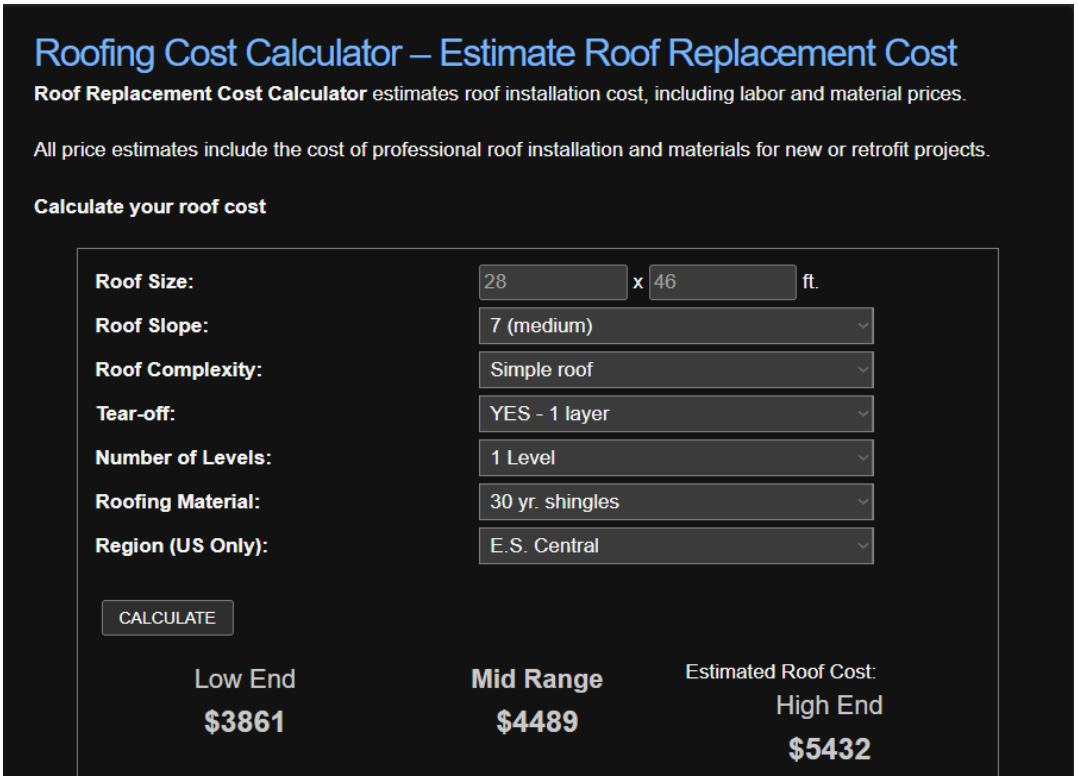

Рисунок 1.3 – Приклад типового онлайн-інтерфейсу покрівельного калькулятора [6]

Аналіз настільних і веб-рішень для проєктування будинку і розрахунку даху виявив спільні риси і ключові відмінності, що визначають їх ринкову позицію.

Всі три розглянуті типи програмних продуктів дозволяють користувачеві отримувати інформацію про параметри майбутньої покрівлі, але роблять це різними способами. ArchiCAD забезпечує повний BIM-цикл з автоматичною генерацією документації, SketchUp дає свободу для 3D-моделювання, а онлайн-калькулятори забезпечують максимальну швидкість для примітивних обчислень. Кожен з них пропонує свой власний розширених набір функцій:

- професійна візуалізація в ArchiCAD;
- величезна бібліотека моделей у SketchUp;
- не потрібна установка з онлайн-калькуляторами.

Однак огляд виявив низку системних проблем, які є спільними для більшості рішень. Зокрема, ArchiCAD і SketchUp вимагають значних часових витрат на

навчання – від декількох тижнів 1, в той час як онлайн калькулятори, навпаки, вкрай обмежені у функціональності, не дозволяють намалювати план будинку довільної форми. Професійні інструменти надмірно складні і дорогі, а безкоштовні веб-калькулятори не забезпечують 3D візуалізацію.

Ще одним важливим аспектом є наявність або відсутність 3D-дисплея. ArchiCAD і SketchUp мають повну 3D-графіку, але не автоматизують матеріальні розрахунки. Онлайн калькулятори підраховують площу і кількість аркушів, але не показують, як буде виглядати будинок. Жоден з існуючих інструментів не поєднує в собі простоту креслення стін, автоматичний розрахунок параметрів даху і візуальну 3D візуалізацію в одному безкоштовному настільному додатку.

Таким чином, для створення конкурентоспроможного додатку для покрівельних розрахунків необхідно поєднати зручність використання онлайн-калькуляторів, гнучкість фарбування стін як в SketchUp і правильність розрахунків як в професійних BIM-системах, додавши візуальний 3D-Visualize і відмовитися від необхідної підписки або прив’язки до конкретної операційної системи.

1.3 Огляд технологій та постановка завдання реформування практик

Щоб відповідати викликам практики передачі технологій, було обрано сучасний стек технологій, у якому кожен компонент відіграє чітко визначену роль у забезпеченні функціональності системи.

Клієнтська частина десктоп-додатку реалізована за допомогою платформи NET 8 і фреймворку Windows Presentation Foundation. WPF – це базова інфраструктура, яка дозволяє створювати власні настільні програми для Windows за допомогою мови розмітки XAML. Серед його переваг – вбудована підтримка векторної графіки, прив’язка даних, можливість стилізації інтерфейсу та розділення логіки та представлення за допомогою шаблону MVVM. Це створює у користувачів враження роботи з професійним будівельним інструментом, а не з аматорською програмою.

Для побудови 3D-візуалізації застосунок містить компонент WebView2, який дозволяє відображати веб-контент на основі рушія Chromium. Цей компонент завантажує HTML-сторінку з кодом JavaScript за допомогою бібліотеки Thre s WebView2 забезпечує двостороннє посилання між кодом C# і JavaScript: програма може викликати функції на веб-сторінці, а веб-сторінка може надсилати повідомлення назад до програми. Такий підхід дозволяє використовувати потужність веб-графіки в рідній настільній програмі без необхідності досліджувати складні низькорівневі API, що значно прискорює розробку.

Three.js – бібліотека для створення 3D графіки в браузері. Він піклується про всю складну роботу з WebGL, забезпечуючи розробника простими і зрозумілими об’єктами: сценою, камерою, джерелами світла, сумками і матеріалами. У додатку, що розробляється, Three.js s відповідає за відображення тривимірної моделі будинку за координатами стін, які користувач намалював на двовимірній площині. Бібліотека дозволяє обертати модель, переміщати її все ближче і далі, міняти стінові матеріали. На даний момент реалізована базова візуалізація стін, а в перспективі – відображення різних видів покрівель і покрівельних матеріалів.

Стандартний елемент полотна WPF використовується для роботи з двовимірною графікою. На ньому розміщена 20-піксельна масштабна сітка, що відповідає 0,5 метра в реальному масштабі. Кожне клацання миші закріплюється, координати округлюються до найближчого вузла сітки і між точками проводиться лінія – стінка. Всі пофарбовані стіни зберігаються в колекції предметів, яка містить початкову та кінцеву точки, довжину в метрах, кут нахилу та візуальне зображення. Користувач може редагувати кожну стіну: змінювати її довжину або кут повороту, і видаляти окремі стіни.

Після того, як користувач пофарбував стіни будинку або його прибудови і отримав закритий контур, програма автоматично розраховує розміри будинку: ширину, довжину і загальну площу. Ці налаштування відображаються на панелі

інструментів і використовуються для розрахунку даху. Розрахунок покрівлі здійснюється виходячи з обраного користувачем типу покрівлі – двоповерхової, рулонної, болотної або горищної. Кожен тип має свої геометричні формули: для двосхилої покрівлі площа розраховується як добуток квадратного прольоту, подвоєного на довжину будинку; для плоских планок додається площа поверхні торцевих планок.

Крім площі, програма розраховує необхідну кількість покрівельних матеріалів. Наприклад, для металевого листа або профільного листа розглядається ефективна площа одного листа (приблизно 0,8 квадратних метрів включаючи лист). Кількість стропілу також розраховується на основі 0,6-метрового кроку встановлення. Всі ці налаштування відображаються користувачеві в режимі реального часу при зміні розміру будинку або типу даху.

Важливою частиною системи є модуль обробки матеріалів. Хоча на поточному етапі використовується статичний список матеріалів з приблизними цінами, архітектура передбачає підключення до зовнішнього API для отримання актуальних цін від постачальників будівельних матеріалів. За допомогою HTTP-запитів програма може надіслати запит із зазначенням регіону та типу матеріалу та отримати JSON із назвою, ціною за одиницю, валютою та назвою постачальника. Це автоматично оновить цитату без участі користувача.

Формат збереження користувацьких проектів – файл JSON [7], який записує всі координати стін, налаштування даху та вибрані матеріали. Надалі планується додати експорт котирувань у форматах Excel і PDF для передачі будівельникам або друку.

Висновки до розділу 1

У першому розділі роботи було проведено детальний аналіз предметної галузі автоматизованих розрахунків покрівельних робіт. Були сформульовані основні поняття та визначення, такі як десктопний застосунок, автоматизація розрахунків, 3D-моделювання у будівництві, типи дахів та технології, що використовуються.

Проведено огляд існуючих аналогів – професійної BIM-системи ArchiCAD, універсального 3D-редактора SketchUp та онлайн-калькуляторів покрівлі. Для кожного з них були виявлені переваги та недоліки. Аналіз показав, що існуючі рішення або надто складні і дорогі для пересічного користувача, або не забезпечують наочної 3D-візуалізації, або не дозволяють створювати довільне планування будинку.

На основі проведеного аналізу було обрано оптимальний технологічний стек: мова C#, платформа .NET 8 [8], фреймворк WPF для створення інтерфейсу, компонент WebView2 для вбудовування вебтехнологій та бібліотека Three.js для 3D-візуалізації. Сформульовано цілі та завдання практики, визначено об’єкт та предмет дослідження. Отже, всі завдання першого розділу вважатимуться виконаними.

2 МОДЕЛІ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ БУДІВЕЛЬНИХ РОЗРАХУНКІВ

2.1 Проєктування архітектури системи

Діаграма послідовності, представлена на рисунку 2.1, ілюструє повний робочий процес розробленого настільного додатку для розрахунку даху та 3D-моделювання.

Після запуску програми користувачеві відкривається початкове вікно, де створюється новий проект. Це ініціює етап будівництва будинку, який підтримує два альтернативні методи введення. Перший метод дозволяє вручну вводити розміри, вказуючи ширину та довжину будівлі. Другий метод забезпечує інтерактивний інтерфейс малювання, де користувач розміщує сегменти стіни, клацаючи на полотні на основі grid–; кожне клацання автоматично фіксується до найближчого вузла сітки для підтримки геометричної точності.

Після визначення стін система генерує колекцію стін і виконує автоматичні розрахунки загальної ширини, загальної довжини та площі будівлі. Потім ці параметри передаються на етап вибору типу даху. Користувач може вибрати один з чотирьох доступних типів даху: двосхилий дах; вальмовий дах; мансардний дах. Після вибору програма виконує розрахунки геометрії даху на основі попередньо виміряних розмірів будівлі.

Розрахований дах одночасно відображається на тривимірній сцені, надаючи користувачеві негайне візуальне представлення. Дотримуючись визначення даху, користувач переходить до фази вибору будівельних матеріалів, де вибираються такі компоненти: первинний покрівельний матеріал (наприклад, металочерепиця або профільоване покриття); пиломатеріали для кроквяної системи; гідроізоляційна мембрана та елементи кріплення (цвяхи або гвинти).

Потім програма надсилає запит до зовнішнього API, щоб отримати поточні ціни на матеріали. Відповідь API містить актуальну інформацію про ціни на основі

вибраного регіону та типу матеріалу. З отриманими цінами система виконує повний матеріальний розрахунок, включаючи як необхідну кількість, так і загальну кошторисну вартість. Нарешті, формується детальна оцінка витрат, і користувач може експортувати результат у форматі Excel або PDF для подальшого використання або друку.

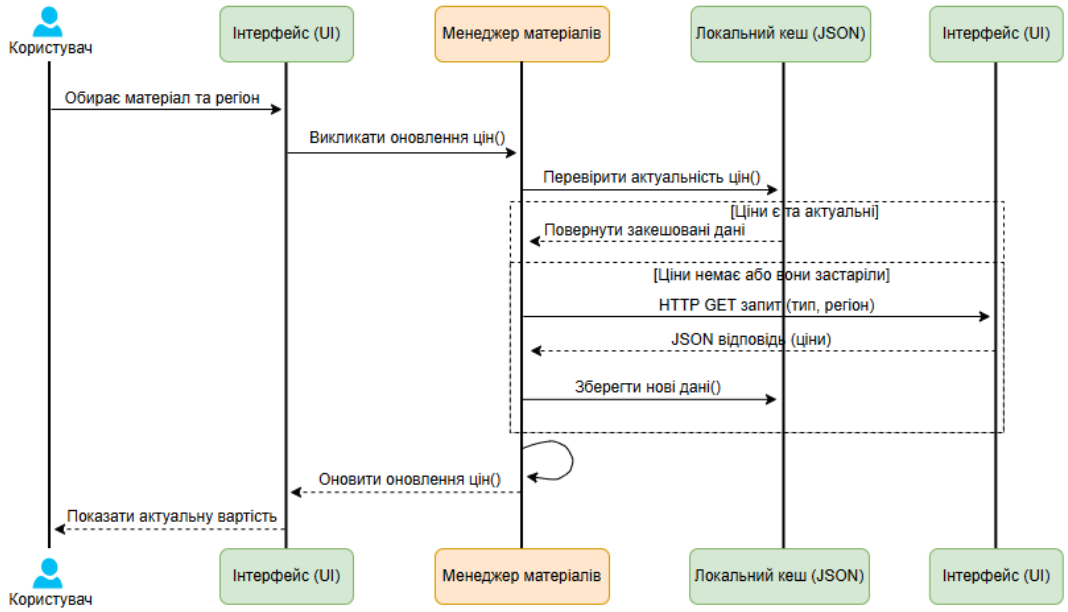

Рисунок 2.1 – Діаграма послідовності робочого процесу програми

2.2 Функціональність модельної системи

На рисунку 2.2 представлена схема повного циклу системи, що включає як веб-сайт, так і настільну програму.

Користувач спочатку переходить на сайт, де він або вона може прочитати опис програми, переглянути скріншоти та інструкції. Він зареєстрований на сайті, після чого має доступ для завантаження інсталяційного файлу. Далі користувач встановлює застосунок на свій комп’ютер і запускає його. У додатку він повторно реєструється або використовує ті самі дані для входу.

Після успішної авторизації користувач має повний доступ до функціоналу програми. Він може малювати стіни на двовимірній сітці вручну або встановлювати розміри будинку вручну. Далі вибирають тип даху – поручитель, граблі, горище або намет. Програма автоматично розраховує розміри будинку та параметри обраної покрівлі, відображаючи тривимірну модель. Після цього користувач підбирає будівельні матеріали – основний покрівельний матеріал, бетонні дошки, гідроізоляційну плівку і кріплення. Застосунок запитує зовнішній API, щоб отримати поточні ціни. З усіх даних генерується повна цитата, яку можна експортувати у форматі Excel або PDF.

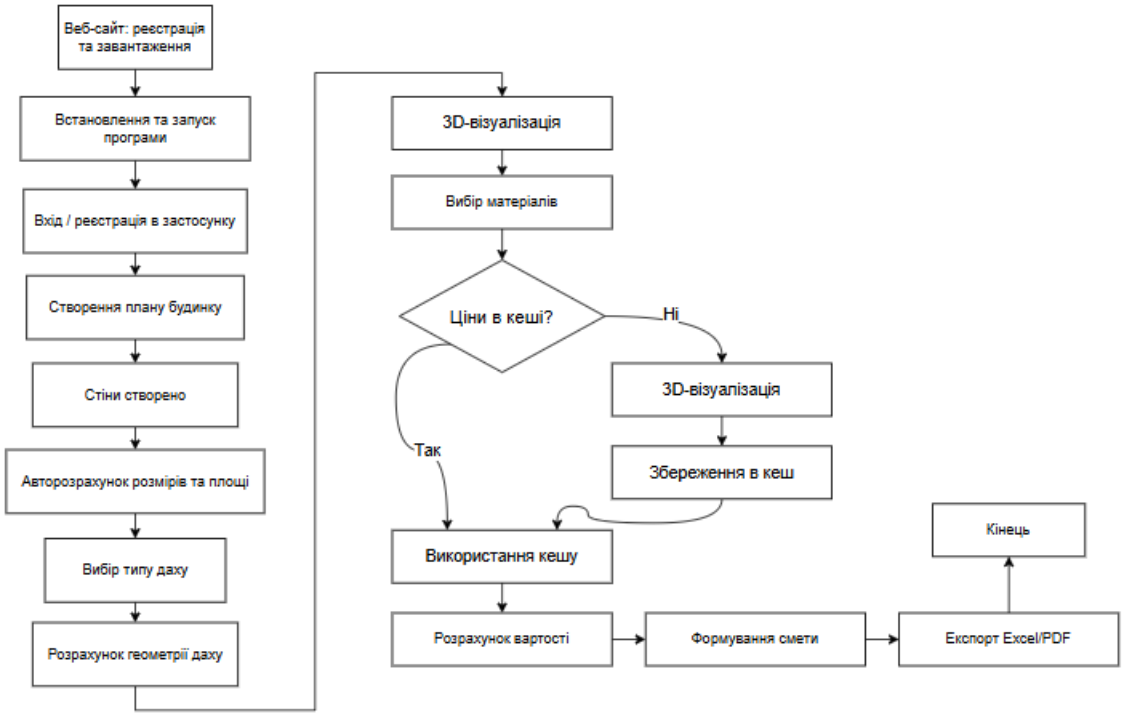

Рисунок 2.2 – Демонстрація робочого процесу

2.3 Інтеграція із зовнішнім API для отримання ціноутворення на матеріали

Однією з ключових особливостей програми є автоматичне отримання поточних цін на будівельні матеріали від зовнішніх постачальників. З цією метою була реалізована інтеграція з REST API [9], яка надає інформацію про вартість

покрівельних матеріалів, бруса, гідроізоляції та кріплення в залежності від обраного регіону.

На рисунку 2.3, представлена діаграма послідовності взаємодії користувача, настільної програми та зовнішнього API.

Процес полягає в наступному. Користувач вибирає тип матеріалу (наприклад, металочерепиця, профнастил, бетонна плитка), вказує площу будівництва (Київ, Львів, Одеса) і ініціює ціновий запит. Застосунок генерує HTTP GET запит в API, передаючи параметри типу матеріалу і регіону.

Реалізовано механізм кешування для оптимізації продуктивності та зменшення кількості мережевих запитів. Якщо дані в локальному кеші (JSON-файлі) актуальні, наприклад, одна година або менше, програма використовує їх без доступу до API. Це прискорює роботу і дозволяє працювати в автономному режимі з останніми отриманими цінами. У разі застарілих або відсутніх даних запит надсилається на сервер.

API повертає відповідь у форматі JSON, який містить такі поля: назва матеріалу, ціна одиниці (м 2, одиниці або кг), валюта (гривня), назва постачальника та інвентар. Застосунок аналізує дані, замінює їх формулами для розрахунку кількості матеріалу та обчислює кінцеву вартість. Оновлений рейтинг відразу відображається користувачеві в інтерфейсі.

Після завершення розрахунків ви можете експортувати повну цитату у форматі Excel або PDF для майбутнього використання – перенесення до креслярів, роздруківок або архівування.

Такий підхід дозволяє завжди мати актуальні ціни на матеріали, автоматично оновлювати контент, скорочувати витрати часу на ручний пошук і введення даних.

Рисунок 2.3 – Послідовність взаємодії користувача

2.4 Обґрунтування вибору технологій для 3D-візуалізації та генерації звітів

Після розробки архітектури системи (Розділ 2.1), опису її функціонального робочого процесу (Розділ 2.2) та впровадження інтеграції API для ціноутворення (Розділ 2.3) необхідно обґрунтувати ключові технологічні рішення, прийняті під час розробки. У цьому розділі наведено порівняльний аналіз альтернативних рішень для двох ключових компонентів: 3D-візуалізації та генерації PDF-звітів. У цьому розділі

наведено порівняльний аналіз альтернативних рішень для двох ключових компонентів: 3D-візуалізації та генерації PDF-звітів.

2.4.1 Порівняння технологій 3D-візуалізації

Застосунок потребує 3D-рендерингу стін і дахових конструкцій у реальному часі з підтримкою керування камерою (обертання, панорамування, зум) та зйомки знімків. Було оцінено чотири кандидатські технології.

Таблиця 2.4.1 – Порівняння технологій 3D-візуалізації

Технології	Переваги	Недоліки	Інтеграція WPF	Продуктивність	Вибрані
OpenTK (OpenGL)	Прямий доступ до GPU; Максимальна продуктивність	Дуже складно; потрібен низькорівневий код; Крута крива навчання	Безпосередньо через GLControl	Чудово	Ні
Helix Toolkit	Нативне керування WPF; вбудовані 3D-примітиви; Хороша документація	Обмежені можливості рендерингу; важко розширити; важкий (~10 MB)	Корінна	Середнє	Ні

Кінець таблиці 2.4.1

Unity Embedded	Потужний рушій рендерингу; Сучасне освітлення та матеріали	Надзвичайно важкий (200 МБ); надмірність для простих дахів; Складне інтегрування	Через зовнішній API	Чудово	Ні
Three.js + WebView2	Легка (<2 МБ); простий JavaScript API; велика екосистема; Легкий знімок	Додатковий шар абстракції (міст C#/JS)	Через WebView2 керування хромом	Добре (апаратно прискорене)	Так

Обґрунтування вибору Three.js + WebView2:

Three.js було обрано через те, що він абстрагує складність WebGL [10, 11], дозволяючи швидко розробляти 3D-сцени з мінімальним кодом. Бібліотека надає вбудовані компоненти для камер, освітлення, матеріалів і геометрії. WebView2 забезпечує безшовне вбудовування веб-контенту на основі Chromium у застосунок WPF, створюючи двонаправлений міст між C# і JavaScript за допомогою методу ExecuteScriptAsync. Цей міст дозволяє додатку передавати координати стін і параметри даху JavaScript-рендереру, який у реальному часі створює та оновлює 3D-модель.

Додаткові переваги включають:

– захоплення знімків: Three.js підтримує рендеринг у toDataURL(), який застосунок використовує для створення 2D-зображень планів і 3D-зображень для експорту PDF без додаткових бібліотеку;

– підтримка спільноти: Three.js має обширну документацію, приклади та активне технічне обслуговування;

– розширеність: Майбутні функції (наприклад, візуалізація кроків, різні матеріали даху) можна реалізувати шляхом розширення існуючого коду JavaScript без змін ядра C#.

2.4.2 Порівняння бібліотек генерації PDF

Заявка повинна експортувати детальні кошториси, що містять таблиці матеріалів, витрати на робочу силу, параметри даху та зображення проєкту. Було розглянуто чотири бібліотеки для генерації PDF.

Таблиця 2.4.2 – Порівняння бібліотек генерації PDF

Бібліотека	Ліцензія	Складність API	Підтримка таблиці	Підтримка зображень	Розмір	Вибрані
iTextSharp	AGPL (комерційні обмеження)	Високий	Так	Так	2 MB	Ні
PdfSharp	MIT	Середнє	Обмежений	Так	1 MB	Ні
QuestPDF	MIT	Low (Fluent API)	Чудово	Так	1 MB	Так
Microsoft.Printing	Вбудовані	Високий	Ні	Ні	відсутній	Ні

Правильне пояснення для відбору QuestPDF:

QuestPDF був вибраний передусім для його Плавного API, який дозволяє конст рукцію декларативного документу, подібну до Xaml або HTML. Цей підхід робить к од легким для читання, maintainable, і менш схильний до помилок порівняно з наказо ве покоління PDF. Бібліотека забезпечує чудову підтримку складних столів з поглин еними осередками, автоматичною нумерацією сторінок, і вклав розташування.

Специфічні переваги включають:

– декларативний синтаксис: Документи є побудував користування ланцюгом методів (Нижній Колонтитул (), Вмісту Шапки (), Сторінки (),()), який відбиває візуа льну структуру завершального PDF;

Стіл, що управляє: Таблиці підтримують визначення колонки, повторення заголовка через сторінки, і митне клітинне моделювання;

– підтримка зображення : байт () акцептів методу Зображення[] масиви, дозв оляючи пряме вкладення screenshots захопив від 3d глядача;

– відкрийте початкову ліцензію: ліцензія Mit дозволяє безкоштовне комерцій не використання без законних обмежень;

– немає зовнішньої залежності: бібліотека нетовариська і не вимагаюча додат кових динамічних компонентів.

Приклад PDF трубопроводу покоління використовував в додатку:

```
Document.Create(container =>
{
   container.Page(page =>
   {
      page.Size(PageSizes.A4);
      page.Margin(2, Unit.Centimetre);
      page.Header().Element(BuildHeader);
      page.Content().Element(c => BuildContent(c, estimate, options));
      page.Footer().AlignCenter().Text(t => t.Span("Page ").CurrentPageNumber());
   });
}).GeneratePdf(outputPath);
```

Висновки до розділу 2

У другому розділі були представлені практичні результати виконання завдань передатестаційної практики. Розроблено архітектуру десктопної програми, побудовано діаграму компонентів, що відображає взаємодію основних модулів:

- 2D-редактора стін;
- модуля розрахунків;
- 3D-візуалізатора;
- менеджера матеріалів;
- сервіс експорту.

Спроектовано діаграму послідовності повного циклу використання системи, що включає реєстрацію користувача на сайті, скачування та встановлення програми, побудова плану будинку, вибір типу даху, розрахунок матеріалів та формування кошторису. Додатково розроблено діаграму взаємодії із зовнішнім API постачальників будівельних матеріалів, що демонструє механізм отримання актуальних цін із використанням локального кешування.

Для кожної діаграми наведено детальний текстовий опис. Розроблений функціонал дозволяє користувачеві створювати проект будинку, візуалізувати його в 3D, вибирати матеріали та експортувати кошторис. Результати другого розділу повністю відповідають поставленим завданням та можуть бути використані для подальшої розробки дипломного проекту.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ТА АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Опис вхідних даних та структури системи

Розроблена інформаційна система для автоматизації розрахунків покрівлі з елементами 3D-моделювання працює на широкому діапазоні вхідних даних, структура та обробка яких безпосередньо визначають як точність отриманих результатів, так і зручність взаємодії з користувачем. Вхідні дані системи можна умовно розділити на три логічні групи: геометричні дані плану будівлі; параметри конфігурації даху та зовнішні дані; отримані шляхом інтеграції з API ціноутворення. Кожна з цих груп має свій власний формат, внутрішнє представлення та правила обробки.

Перша група, геометричні дані плану будівлі, складається з послідовності двовимірних точок, які користувач розміщує на робочому полотні, малюючи контур стін будівлі. Кожна точка представлена незмінною структурою з двома координатами: X, виміряною вздовж горизонтальної осі світової системи координат, і Z, виміряною вздовж осі глибини. Вибір X і Z як назв площинних осей замість більш традиційних X і Y продиктований правосторонньою системою координат, прийнятою в тривимірній сцені Three.js, де вісь Y зарезервована для вертикального напрямку. Ця конвенція послідовно підтримується у всьому коді програми, що дозволяє уникнути необхідності перемапування осей під час передачі даних між 2D-редактором і 3D-переглядачем.

Координати зберігаються в метрах з подвійною точністю у форматі з рухомою комою. Мінімальний крок механізму прив’язки до сітки становить 0,5 м, а це означає, що будь-яке натискання користувача на полотні округлюється до найближчого вузла сітки за формулою 3.1:

$$x\_snapped = round\left(\frac{x\_raw}{s}\right)s, \quad (3.1)$$

де x_snapped – це координата, отримана після прив’язки;

x_raw – це початкова координата, отримана з положення клацання миші після перетворення з екранних координат у світові;

s – це крок сітки в метрах (у поточній реалізації s = 0,5).

Пара двох таких точок P1 = (x1, z1) і P2 = (x2, z2) утворює один сегмент стіни будівлі. Стіна представлена екземпляром класу Wall, який розширює базовий клас ObservableObject для підтримки автоматичного поширення змін властивостей через механізм прив’язки платформи WPF[12]. Клас Wall інкапсулює не тільки початкову та кінцеву точки, але й кілька похідних властивостей: довжину стіни в метрах; кут нахилу до горизонтальної осі; координати середньої точки, що використовуються для відображення текстових міток; висоту стіни (за замовчуванням 2,5 м) і товщину стіни (за замовчуванням 0,3 м).

Довжина кожної стіни розраховується як евклідова відстань між її кінцевими точками за формулою 3.2:

$$L = \sqrt{((x^2 - x^1)^2 + (z^2 - z^1)^2)}, \quad (3.2)$$

де L – довжина стіни в метрах;

x1, z1 – координати початкової точки;

x2, z2 – координати кінцевої точки.

Кут нахилу стіни до горизонтальної осі X визначається за допомогою функції arctangent з двома аргументами за формулою 3.3:

$$\alpha = atan2(z^2 - z^1, x^2 - x^1) \cdot \frac{180}{\pi}, \quad (3.3)$$

де α – кут нахилу в градусах;

atan2 – функція, яка повертає кут вектора в діапазоні [-180; 180°] з урахуванням ознак обох аргументів.

Повний план будівлі представлений замовленою колекцією стінових об’єктів. Порядок стін у колекції відображає послідовність, в якій користувач намалював їх, що має важливе значення для правильної інтерпретації контуру як замкнутого багатокутника. Щоб визначити, чи закритий контур, система перевіряє близькість кінцевої точки останньої стіни до початкової точки першої стіни, якщо відстань між ними не перевищує похибку в 5 см, то контур вважається закритим, а система автоматично виходить з режиму малювання і запускає перерахунок всіх похідних параметрів будівлі.

Друга група вхідних даних, конфігураційних параметрів покрівлі, пов’язана з типом покрівлі, обраним користувачем. У поточній реалізації системи повністю підтримується двосхилий тип даху, в той час як інші типи (шатровий дах, мансардний дах, тазостегновий мансардний дах, пірамідальний дах, односхилий дах, багатосхилий дах, багатосхилий дах, плоский дах) зарезервовані для майбутніх розширень. Параметри двосхилим даху інкапсульовані в класі GableRoofParameters і включають в себе наступні поля:

– варіант двосхилим даху (симетричний, асиметричний або з різною бічною висотою);
– висота коника над вершиною стіни;
– зміщення коника для асиметричного варіанту;
– висота лівого та правого схилу для варіанту з різною висотою;
– фронтальний звис карниза даху;
– фронтальний звис карниза даху та товщина даху для візуального представлення.

Фронтальний звис карниза і фронтальний звис визначають геометричний виступ даху за контуром стіни. Згідно з українськими державними будівельними

нормами, довжина карниза повинна бути не менше 0,5 м, щоб забезпечити надійний захист зовнішніх стін від атмосферних опадів. Тому це значення використовується як стандартне в системі, але користувач може налаштувати його в розумних межах в інтерфейсі редактора даху.

Третя група вхідних даних – це інформація, отримана шляхом інтеграції із зовнішнім API ціноутворення сервісу buildcalculator.io. Для кожної категорії матеріалу, необхідної для оцінки покрівлі (покрівельне покриття, гідроізоляційна мембрана, пароізоляція, ізоляція, кроквяний брус, обрешітка, контробрешітка, елементи оздоблення, кріплення), система відправляє текстовий запит на API, що містить опис матеріалу англійською або англійською мовами. API повертає відповідь JSON з детальною інформацією про ціни, включаючи ціну за одиницю вимірювання, валюту (зазвичай євро), відсотки праці, матеріали та обладнання в загальній вартості, а також класифікацію робочого елемента.

Внутрішня структура розробленої програми побудована за класичним архітектурним шаблоном Model-View-ViewModel (MVVM) [13], який є стандартом де-факто для проєктування WPF-додатків і забезпечує чіткий поділ між шаром даних, шаром бізнес-логіки та шаром презентації інтерфейсу користувача. Це розділення має кілька важливих наслідків для масштабованості та тестуваності вихідного коду: класи моделей можуть бути перевірені ізольовано без створення будь-яких візуальних компонентів; моделі перегляду можуть бути замінені або розширені без впливу на логіку рендерингу; перегляди можуть бути перероблені, не торкаючись основних алгоритмів обробки даних.

Загальна структура програми складається з наступних логічних шарів, кожен з яких відповідає окремій папці в репозиторії вихідного коду.

Шар моделей (розташований у папці Моделі) містить класи даних, які описують об’єкти домену програми: Point2D; Стіна; Підлога; Проект; GableRoof; GableRoofParameters; RoofGeometry; RoofPanel; Edge3D; Vector3; MaterialItem;

MaterialCatalog; Кошторис; КошторисLine; КошторисSection. Ці класи не містять ніякої бізнес-логіки, операцій з інтерфейсом користувача або зовнішніх викликів послуг; вони є чистими контейнерами даних, можливо, з простими похідними властивостями, розрахованими з первинних полів.

Шар ViewModels (папка ViewModels) містить класи, які опосередковуються між моделлю та видом, виставляючи дані у формі, зручній для прив'язки та реалізації відповіді на дії користувача за допомогою команд. Основним класом цього шару є MainViewModel, який містить колекцію стін поточного проекту, вибрану стіну, режим редактора (вигляд або малюнок), стан редагування параметрів стіни та набір команд, таких як StartDrawingWallsCommand, ClearAllCommand, BuildGableRoofCommand, RemoveRoof Команда.

Шар Views (папка Views) містить файли розмітки XAML [14], що описують візуальне представлення вікон та елементів керування, а також їх файли, що містять обробники подій та низькорівневі операції над графічними примітивами. Головне вікно програми реалізовано у файлах MainWindow.xaml та MainWindow.xaml.cs. Додаткові вікна включають початкове вікно StartWindow, діалогове вікно вибору типу даху RoofTypeWindow, вікно редактора даху RoofEditorWindow та повноекранне вікно перегляду 3D Viewer3DWindow.

The Services layer (папка Services) містить служби бізнес-логіки, які виконують обчислення, перетворення та зовнішню комунікацію. Найважливіші послуги включають GeometryService для геометричних операцій над точками і багатокутниками, CanvasTransform і CanvasRenderer для роботи з полотном 2D малювання, GableRoofBuilder для побудови геометрії двосхилим даху, OrientedRect для знаходження мінімально орієнтованого розміру прямокутника довільного набору точок, Web3DService для інтеграції з компонентом WebView2, BuildCalculatorApi для взаємодії з зовнішнім API ціноутворення, і NbuExchangeApi для отримання поточного курсу української гривні до євро від Національного банку України.

Шар Helpers (папка Helpers) містить допоміжні класи, що використовуються в усьому додатку. базовий клас ObservableObject, що реалізує інтерфейс INotifyPropertyChanged, клас RelayCommand, що реалізує інтерфейс ICommand, і кілька перетворювачів значень для зв’язування XAML.

Ресурси інтерфейсу користувача організовані в папці «Ресурси/стилі», де визначено два словники стилів: Colors.xaml містить колірну палітру програми; засновану на світлій темі (основний фон, вторинний фон, фон панелі, колір акценту, кольори тексту); а Controls.xaml містить стилі для кнопок; текстових полів; полів списків; меню та інших стандартних елементів керування. Стилі засновані на мові Fluent Design і забезпечують візуально послідовний і професійний вигляд програми.

Статичні веб-ресурси, необхідні для роботи 3D-переглядача, знаходяться в папці wwwroot. Ця папка містить HTML-сторінку viewer.html, JavaScript-файл viewer.js, що містить всю логіку сцени Three.js, і підтеку lib з бібліотекою Three.js і модулем OrbitControls. Вміст теки wwwroot копіюється до каталогу виводу збірки завдяки параметру CopyToOutputDirectory ReserveNew у файлі проекту RoofCalculator.csproj.

Технологічний стек програми включає наступні основні компоненти, викладені в таблиці 3.1.

Таблиця 3.1 – Технологічний стек розробленого застосунку

Компонент	**Версія**	**Призначення**
.NET	8.0	Платформа виконання
WPF	у складі .NET 8	Фреймворк настільного UI
WebView2	1.0.3912.50	Вбудований компонент Chromium

Кінець таблиці 3.1

Three.js	r128	JavaScript бібліотека 3D-рендерингу
OrbitControls	r128	Модуль орбітального керування камерою
System.Text.Json	у складі .NET 8	Серіалізація JSON
System.Net.Http	у складі .NET 8	HTTP-клієнт для інтеграції з API

Взаємодія між кодом C# [15, 16] програми та кодом JavaScript 3D-переглядача здійснюється через двонаправлений міст, реалізований компонентом WebView2. Сторона C# може викликати функції JavaScript на сторінці за допомогою методу ExecuteScriptAsync, тоді як сторона JavaScript може відправляти повідомлення назад на хост C# через API chrome.webview.postMessage. У поточній реалізації системи активно використовується тільки напрямок C#, JavaScript (програма передає глядачеві геометричні дані стін і даху) який потім конструює і рендерить відповідні 3D сітки.

Потік даних всередині програми можна проілюструвати наступною послідовністю подій, які відбуваються, коли користувач малює нову стіну на полотні. По-перше, подія клацання кнопкою миші відображається обробником Canvas MouseLeftButtonDown у коді позаду головного вікна. Обробник перетворює координати екрана на світові координати і застосовує перетворення прив’язки до сітки. Отримана точка світу передається до методу HandleCanvasClick моделі MainViewModel, який додає новий екземпляр стіни до колекції стін поточного поверху. Зміна колекції запускає подію CollectionChanged, яка спостерігається як візуалізатором полотна, так і службою 3D-перегляду. Візуалізатор полотен перемальовує шар стіни в полотні 2D, в той час як служба 3D-перегляду серіалізує

оновлений список стін в JSON і викликає функцію window.updateWalls JavaScript. Всередині глядача функція аналізує JSON, утилізує попередні сітки стін, конструює нові екземпляри BoxGeometry з відповідними розмірами, позиціями та обертаннями і додає їх до master buildingGroup. Нарешті, сцена знову центрована і повторно обрамлена камерою, так що вся будівля видно в межах оглядового майданчика.

3.2 Демонстрація роботи розробленої системи

Цей підрозділ представляє детальну демонстрацію роботи розробленої інформаційної системи у вигляді покрокового сценарію користувача, який охоплює всі основні функціональні можливості програми. Демонстрація проілюстрована скріншотами фактичного інтерфейсу користувача, зробленими під час тестування програми на операційній системі Windows 11 з роздільною здатністю екрану 1920×1080 пікселів.

Після запуску виконуваного файлу RoofCalculator.exe користувачеві представляється стартове вікно програми, зовнішній вигляд якого показаний на рисунку 3.1. Стартове вікно має мінімалістичний дизайн і містить логотип програми, його повну назву «Roofing Calculator», короткий опис мети програми, і одна кнопка основної дії з написом «Створити новий проект». Така конструкція відповідає принципу прогресивного розкриття, а саме користувач не перевантажений варіантами на самому початку і м’яко орієнтується на найважливіші дії.

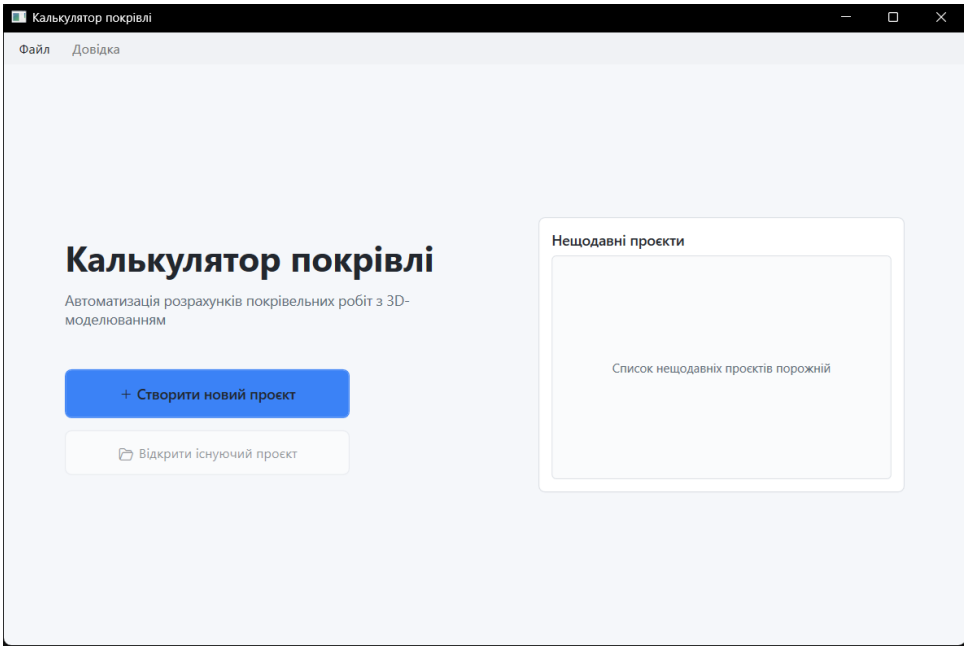

Рисунок 3.1 – Стартове вікно застосунку

Після натискання на кнопку «Створити новий проект» користувач переходить до головного робочого вікна програми, яке являє собою центральний робочий простір, де відбувається весь процес проєктування плану будівлі та даху. Головне вікно розділене на кілька функціональних зон, як показано на рис. (3.2).

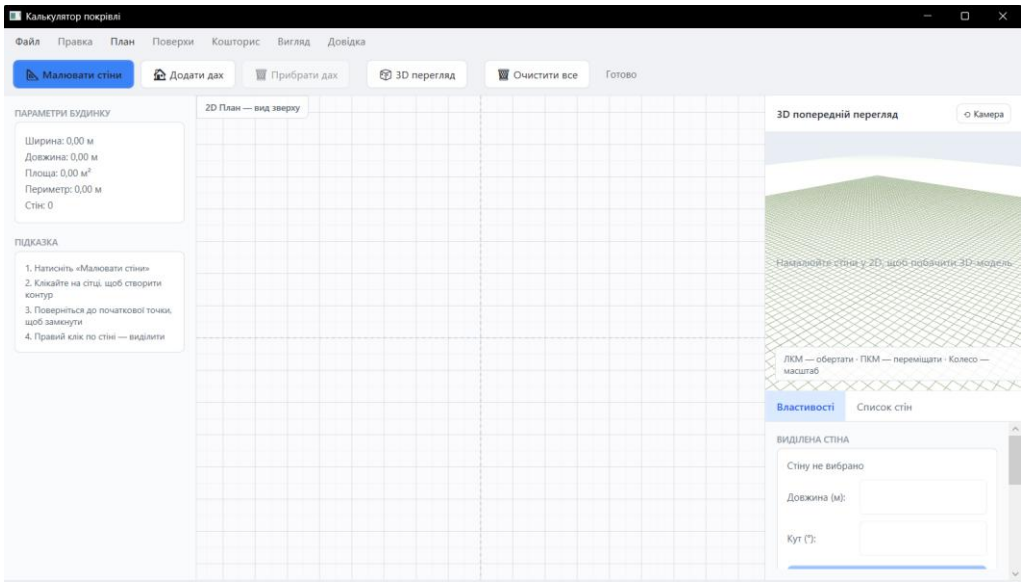

Рисунок 3.2 – Головне вікно застосунку у початковому стані

У верхній частині вікна міститься панель меню зі стандартними пунктами «Файл», «Правка», «План», «Поверхи», «Оцінка», «Перегляд» і «Довідка». Під панеллю меню розташовано горизонтальну панель інструментів з основними кнопками дій, згрупованими за функціональною категорією. Ліва група містить кнопку «Намалювати стіни», яка відображається кольором акценту, щоб підкреслити, що це основна дія на початковому етапі роботи. Наступна група присвячена операціям на даху і містить кнопки «Додати дах» і «Видалити дах». Третя група містить кнопку «3D view» для відкриття повноекранного тривимірного глядача, а остання група містить руйнівну дію «Clear all», що відображається червоним кольором, щоб привернути увагу користувача до її незворотного характеру.

Ліва панель головного вікна містить параметри будівлі (ширина, довжина, площа, периметр, кількість стін) і блок підказок з короткими інструкціями для користувача. Центральну зону займає двомірне креслярське полотно, яке виводить сітку з кроком 0,5 м і служить основною робочою поверхнею для розміщення сегментів стін. Права панель вікна розділена на дві частини, верхня частина містить 3D-попередній перегляд, що працює на компоненті WebView2 та бібліотеці Three.js, тоді як нижня частина містить властивості вибраної стіни та список всіх стін у поточному проекті.

Для початку складання плану будівлі користувач натискає кнопку «Намалювати стіни» на панелі інструментів. Кнопка змінює свій вигляд на натиснутий стан, на смужці стану внизу вікна відображається повідомлення «Режим малювання: натисніть на сітку для створення стін. Клацніть правою кнопкою миші, щоб скасувати», і курсор над полотном змінюється на перехрестя, що вказує на те, що програма готова прийняти першу точку контуру. Потім користувач натискає на полотно, щоб розмістити початкову точку, і система позначає його зеленим колом, як показано на рис. (3.3).

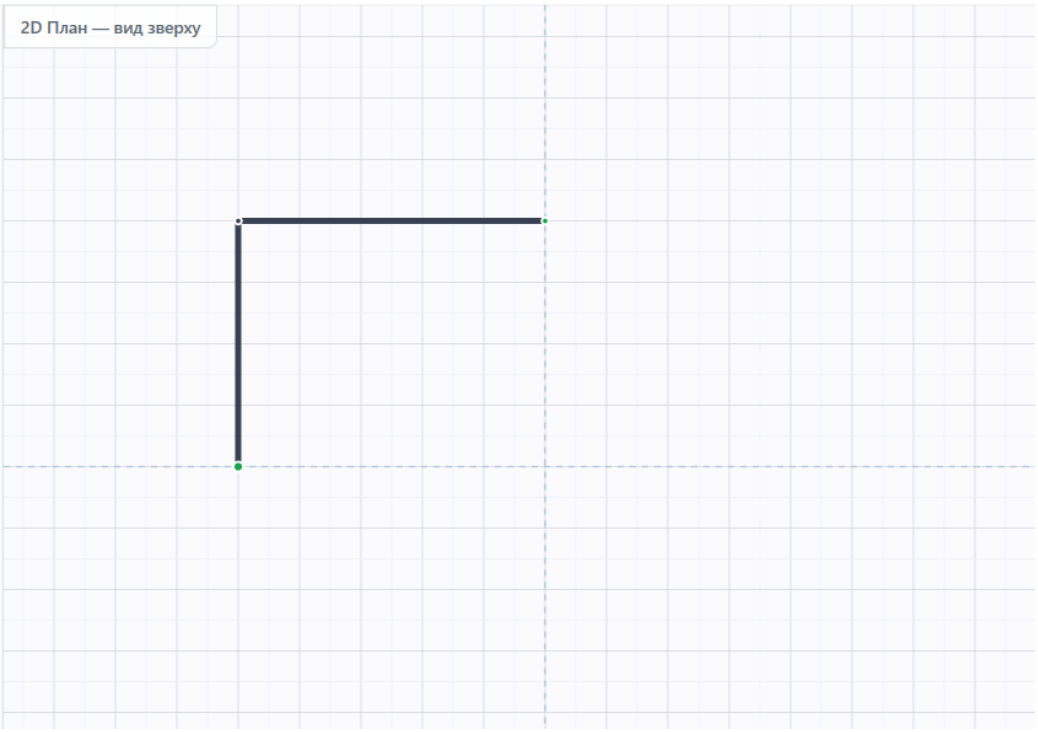

Рисунок 3.3 – Початок малювання контуру будинку

Перетворення координат клацання миші на світові координати та застосування правила прив’язки до сітки реалізовані в класі CanvasTransform.

```
{
    double worldX = (screenPoint.X - OriginX) / PixelsPerMeter;
    double worldZ = (screenPoint.Y - OriginZ) / PixelsPerMeter;
    return new Point2D(worldX, worldZ);
}


public Point2D SnapToGrid(Point2D point)
{
    double snappedX = Math.Round(point.X / GridStepMeters) * GridStepMeters;
    double snappedZ = Math.Round(point.Z / GridStepMeters) * GridStepMeters;
    return new Point2D(snappedX, snappedZ);
}
```

Константи OriginX і OriginZ визначають положення координатної системи світу в просторі координат екрана, тоді як константа PixelsPerMeter визначає масштаб візуалізації полотна. Разом ці константи визначають афінне перетворення, яке відображає будь-який піксель полотна до точки на площині реального світу і назад.

Кожен наступний клік користувача розміщує нову точку на полотні і створює сегмент стіни, що з’єднує її з попередньою точкою. Під час руху курсора між кліками система відображає рядок попереднього перегляду від останньої розміщеної точки до поточної позиції курсора разом з плаваючим значком, який показує довжину майбутньої стіни в метрах і кут її нахилу в градусах. Цей зворотний зв’язок у режимі реального часу значно полегшує процес нанесення контуру з точними розмірами, як показано на рис. (3.4).

Рисунок 3.4 – Процес малювання стіни з відображенням довжини та кута

Механізм прив’язки до сітки, описаний у попередньому підрозділі, гарантує, що всі сегменти стін мають довжину, кратну 0,5 м, і що кути будівлі завжди лежать у

вузлах сітки. Це виключає типову проблему онлайн-калькуляторів, де користувач повинен вручну вводити точні розміри стіни в числові поля введення, що є одночасно стомлюючим і схильним до помилок.

Коли користувач клацає біля початкової точки першої стіни в межах похибки 5 см у світових координатах, система автоматично виявляє закриття контуру. Режим малювання закінчується, остання стіна додається до колекції, і застосунок негайно перераховує параметри будівлі: ширину; довжину; площу та периметр. Ці параметри відображаються на лівій панелі та оновлюються в режимі реального часу, як показано на рис. (3.5).

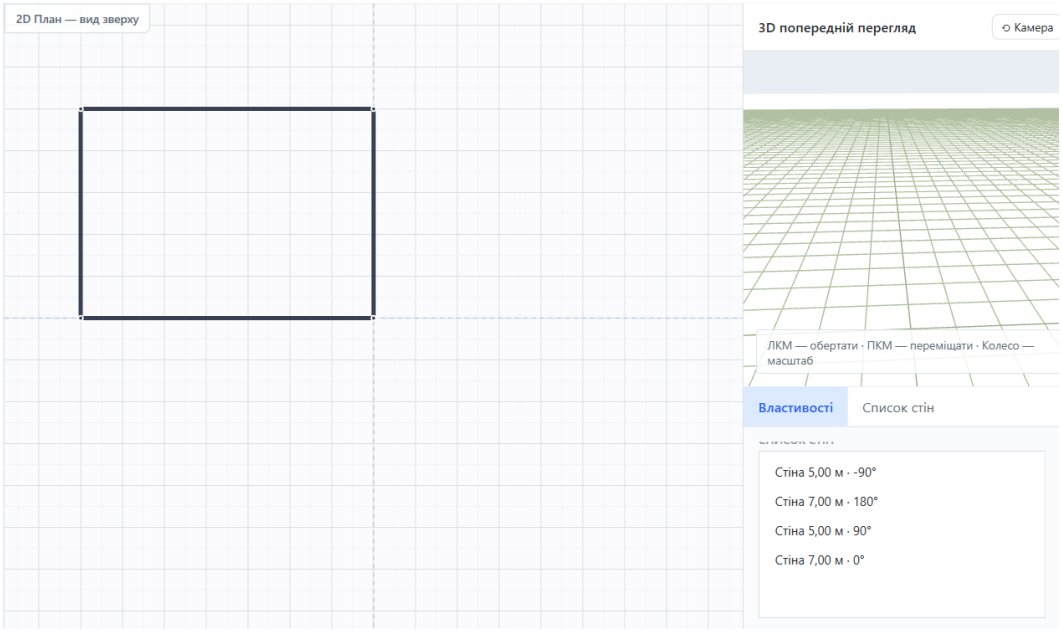

Рисунок 3.5 – Замкнений контур будинку прямокутної форми зі стороною 7×5 метрів

Виявлення контурного замикання виконується автоматично при кожному додаванні нової стіни в колекцію. Реалізація цієї перевірки показана даним кодом.

```
public static bool IsClosedContour(IList<Wall> walls)
{
    if (walls.Count < 3) return false;
    var first = walls[0].Start;
```

```
    var last = walls[^1].End;
    return first.DistanceTo(last) < SnapTolerance;
}


public const double SnapTolerance = 0.05;
```

Метод спочатку перевіряє, що колекція стін містить щонайменше три сегменти, оскільки закритий багатокутник вимагає мінімум трьох сторін. Якщо ця умова виконана, метод порівнює початкову точку першої стіни з кінцевою точкою останньої стіни. Якщо їх відстань не перевищує похибку в 5 см, контур вважається закритим. Допустиме значення 5 см вибирається як компроміс між зручністю користувача (що дозволяє неточно клацати) і запобіганням випадкового закриття, коли користувач ще не закінчив малювати будівлю.

Після закриття контуру програма автоматично обчислює площу будівлі за формулою Гауса (також відомою як формула «шнурка»). На відміну від простих методів розрахунку, заснованих на обмежувальних прямокутниках, цей підхід працює правильно для довільних багатокутних форм, включаючи Г-подібну, Т-подібну та U-подібні макети.

```
public static double PolygonArea(IList<Point2D> points)
{
    if (points.Count < 3) return 0;


    double area = 0;
    for (int i = 0; i < points.Count; i++)
    {
        var a = points[i];
        var b = points[(i + 1) % points.Count];
```

```
        area += a.X * b.Z - b.X * a.Z;
    }
    return Math.Abs(area) / 2.0;
}
```

Алгоритм проходить через всі вершини багатокутника по порядку, обчислюючи перехресний добуток двох послідовних пар координат вершин. Кінцевим результатом є абсолютне значення половини накопиченої суми. Використання операції modulo % points. Підрахунок дозволяє правильно закрити ітерацію з останньої вершини назад до першої, не вимагаючи особливого випадку для останньої ітерації.

Система підтримує не тільки прямокутні будівлі, але і будівлі довільної замкнутої багатокутної форми, включаючи Г-подібну, Т-подібну і П-подібну планування. Щоб продемонструвати цю можливість, користувач може намалювати більш складний контур, наприклад, Г-подібну будівлю, придатну для житлового житла з окремою прибудовою гаража. Приклад такого контуру показаний на рисунку 3.6, де будівля складається з шести стінових сегментів, що утворюють неопуклий замкнутий багатокутник

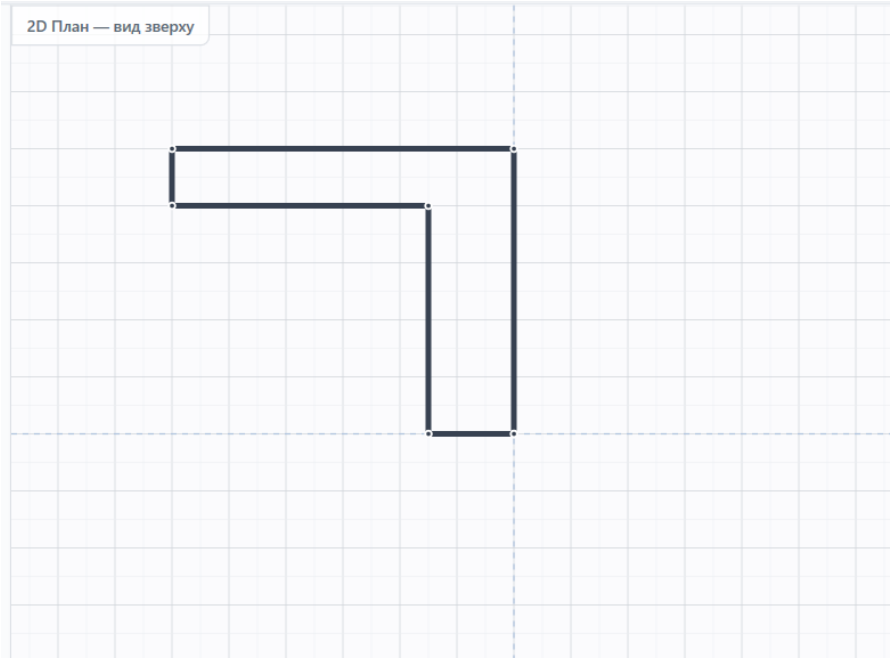

Рисунок 3.6 – Контур будинку Г-подіної форми

Одночасно з побудовою двомірного плану безперервно оновлюється 3D-попередній перегляд в правій панелі головного вікна. Стіни виконані у вигляді тривимірних прямокутних паралелепіпедів з налаштованою висотою 2,5 м і налаштованою товщиною 0,3 м, з використанням світло-бежевого матеріалу, що імітує зовнішній вигляд оштукатурених стін. По кутах будівлі додають додаткові кубічні стійки, щоб забезпечити візуальну безперервність стін при не правих кутах стикування. Невеликий темний ковпачок розміщується поверх кожної стіни, щоб імітувати технологічну настінну пластину, як показано на рис. (3.7).

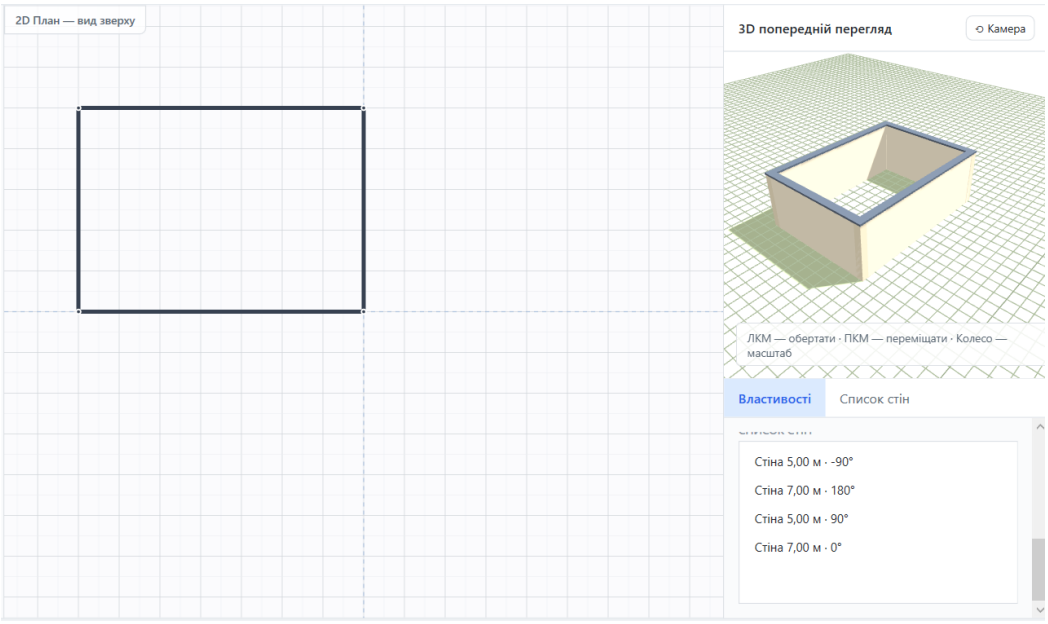

Рисунок 3.7 – 3D-прев’ю будинку зі стінами без даху

Синхронізація в реальному часі між 2D-редактором і 3D-переглядачем здійснюється через міст WebView2. Коли колекція стін змінюється, клас Web3DService серіалізує поточний стан у формат JSON і викликає відповідну функцію JavaScript на сторінці.

```
public Task UpdateWallsAsync(IList<Wall> walls)
{
    var payload = walls.Select(w => new
    {
        sx = w.Start.X, sz = w.Start.Z,
        ex = w.End.X,   ez = w.End.Z,
        h = w.Height,   t = w.Thickness
    }).ToArray();
     var json = JsonSerializer.Serialize(payload);
    var escaped = json.Replace("\\", "\\\\").Replace("'", "\\'");
    return _webView.CoreWebView2.ExecuteScriptAsync(
        $"window.updateWalls('{escaped}')"
    );
```

Використання анонімних об’єктів з короткими іменами властивостей (sx, sz, ex, ez, h, t) замість повних імен мінімізує розмір корисного навантаження JSON, що передається через міст. Відхід від зворотних слешів і одинарних лапок необхідний, тому що JSON рядок вбудований в JavaScript вираз як рядковий літерал, і неописані спеціальні символи порушать синтаксис викликаного сценарію.

На стороні JavaScript отриманий JSON рядок аналізується і використовується для побудови відповідної 3D геометрії. Для кожної стіни створюється геометрія коробки з розмірами, що відповідають значенням реального світу, а отримана сітка повертається відповідно до кута нахилу стіни.

```
function addWall(w) {

  const dx = w.ex - w.sx;
  const dz = w.ez - w.sz;
  const len = Math.sqrt(dx * dx + dz * dz);
  if (len < 0.01) return;


  const angle = Math.atan2(dz, dx);
  const geom = new THREE.BoxGeometry(len + w.t, w.h, w.t);
  const mesh = new THREE.Mesh(geom, wallMaterial);


  mesh.position.set(
    (w.sx + w.ex) / 2,
    w.h / 2,
    (w.sz + w.ez) / 2
  );
  mesh.rotation.y = -angle;
  mesh.castShadow = true;
  mesh.receiveShadow = true;
```

```
    buildingGroup.add(mesh);
}
```

Довжина геометрії коробки збільшується на величину товщини, щоб переконатися, що сусідні настінні коробки перекриваються по кутах і виробляють візуально безперервну поверхню стіни без зазорів. Положення сітки встановлюється в середню точку сегмента стіни, щоб зберегти геометрію з центром навколо її початку, а обертання застосовується навколо вертикальної осі Y з негативним знаком, щоб компенсувати різницю між правою системою координат Three.js і лівою конвенцією 2D екранних координат.

Після того, як контур будівлі був закритий і користувач задоволений плануванням, наступним етапом робочого процесу є додавання даху. Користувач натискає кнопку «Додати дах» на панелі інструментів, а в центрі екрана з’являється модальне діалогове вікно, як показано на рисунку 3.8. Це вікно надає користувачеві візуальний каталог доступних типів даху, кожен з яких представлений плиткою зі стилізованою іконкою, назвою та індикатором доступності.

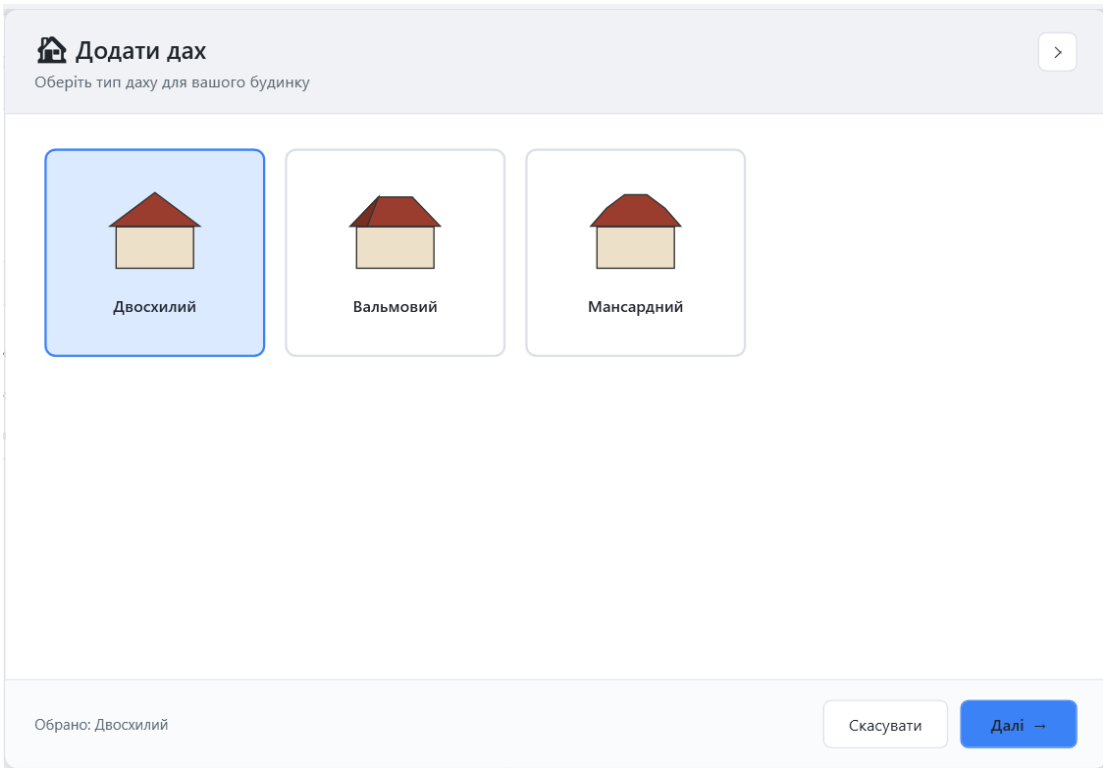

Рисунок 3.8 – Вікно вибору типу даху

Діалогове вікно містить 3 черепиць типу даху, розташованих у сітці: двосхилий; вальмовий; мансардний. У поточній реалізації системи повністю підтримується тільки двосхилий тип даху, а відповідна черепиця відображається в повному кольорі і є інтерактивною. В майбутьоному планується додати ще декілька плиток які будуть відображатися зі зниженою непрозорістю і містять мітку «найближчим часом» під назвою даху, яка чітко повідомлює користувачеві, що відповідна функціональність зарезервована для майбутніх версій програми.

Коли користувач натискає на двосхилий дах черепиці, вона підсвічується з колірною рамкою акценту, нижня мітка стану діалогового вікна змінюється з «Type not selected» на «Selected: Gable roof», і кнопка «Next» в нижній правій частині діалогового вікна вмикається. Після натискання кнопки «Далі» діалогове вікно буде закрито, а користувач перейде до вікна редактора даху, описаного нижче.

Побудова геометрії двосхилим даху з контуру закритої стіни виконується статичним класом GableRoofBuilder. Алгоритм працює в три етапи: по-перше, він знаходить орієнтований обмежувальний прямокутник контуру за допомогою методу обертових супортів на опуклій оболонці вершин стінки; по-друге, він визначає орієнтацію коника даху по більш довгій стороні обмежувального прямокутника; по-третє, він будує геометричні компоненти даху (два схили, коник, карниз і дві фронтони) в локальній системі координат обмежувального прямокутника, а потім перетворює їх назад у світову систему координат.

```
public static RoofGeometry Build(IList<Wall> walls, GableRoofParameters p)
{
    var geometry = new RoofGeometry();
    if (walls.Count < 3) return geometry;


    var contour = GeometryService.GetContourPoints(walls);
    var rect = OrientedRect.Compute(contour);


    double rw = rect.Width, rl = rect.Length;
    double baseAngle = rect.AngleRadians;


    if (rw > rl)
    {
        (rw, rl) = (rl, rw);
        baseAngle += Math.PI / 2;
    }
     double wallTop = walls[0].Height;
```

```
        double eaveY = wallTop - Math.Max(0.12, p.Thickness * 0.6);
        double ridgeY = wallTop + p.RidgeHeight;

        BuildSlopes(geometry, rect, rw, rl, eaveY, ridgeY, p);
        BuildGableFaces(geometry, rect, rw, rl, wallTop, ridgeY, p);
        BuildEdges(geometry, rect, rw, rl, eaveY, ridgeY, p);

        return geometry;
    }
```

Перевірка, якщо (rw > rl) гарантує, що коник даху завжди проходить уздовж довшої сторони обмежувального прямокутника, що є звичайною орієнтацією двосхилих дахів в архітектурній практиці. Зміна значень ширини і довжини супроводжується обертанням локальної системи координат на 90 градусів для підтримки узгодженості наступних обчислень. Значення eaveY встановлюється трохи нижче верхньої частини стіни зміщенням, пропорційним товщині даху, що гарантує, що нижні краї укосів даху візуально перекриваються з верхніми стінками і усувають будь-які потенційні зазори в рендерингу.

Вікно редактора даху – це окреме вікно верхнього рівня програми, яке охоплює всю робочу область та забезпечує спеціальне середовище для візуалізації та налаштування даху, як показано на рисунку 3.9. Заголовок вікна містить Кнопка «Назад до плану» зліва, яка дозволяє користувачеві повернутися до головного вікна без втрати роботи, етикетка із зазначенням обраного типу даху («Двосхилий дах»), а з правого боку кнопка «Скинути камеру» і кнопка підтвердження «Готово» відображаються в акцентному кольорі.

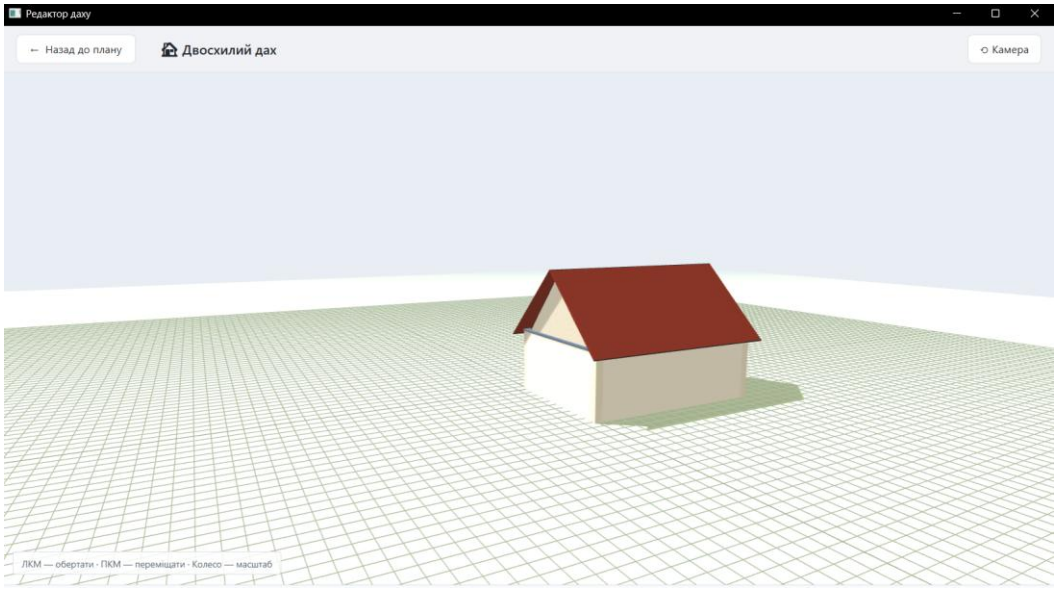

Рисунок 3.9 – Вікно редактора даху з 3D-моделлю двосхилого даху

Повноекранний переглядач використовує ту ж сцену Three.js, що і вбудований попередній перегляд, але з підвищеною роздільною здатністю візуалізації, яка дозволяє користувачеві детально вивчити будівлю з будь-якого кута. Це особливо корисно під час представлення проекту клієнту або друку знімка екрана високої роздільної здатності для конструкторської документації.

Після того, як план будівлі і дах були завершені, користувач може перейти до етапу розрахунку кошторису, натиснувши відповідну кнопку. Кошторис розраховується виходячи з геометричних параметрів покрівлі, отриманих з побудованої тривимірної моделі:

- загальна площа покрівельного покриття;
- загальна довжина коника і карниза;
- загальна довжина двосхилих країв, а також середній кут нахилу.

Для кожної матеріальної категорії система надсилає запит до API buildcalculator.io з текстовим запитом, отримує відповідь у форматі JSON та конвертує ціну з євро в українську гривню за поточним курсом, отриманим з API Національного банку України. Остаточна оцінка представляється користувачеві у вигляді

структурованої таблиці з рядками, згрупованими за матеріальними категоріями, і загальною вартістю, відображеною внизу таблиці.

Програма також надає користувачеві можливість експортувати розраховану оцінку в PDF-файл для подальшого друку або спільного використання з будівельними підрядниками. Експорт PDF здійснюється за допомогою бібліотеки iText і включає в себе основну інформацію про проект (ім’я, дату створення), геометричні параметри будівлі та даху, детальний перелік матеріалів з їх кількістю та цінами та остаточну загальну вартість.

Робочий процес користувача, описаний в цьому підрозділі, відображається в архітектурних рішеннях, прийнятих під час проєктування програми: однонаправлений потік даних від редактора 2D-плану до конструктора даху до калькулятора оцінки, спільної моделі проекту, яка синхронізує стан всіх вікон, і модального характеру діалогового вікна вибору типу даху, який гарантує, що користувач завжди робить чіткий вибір, перш ніж перейти до наступного етапу. Ця послідовна і передбачувана поведінка програми зменшує когнітивне навантаження на користувача і робить весь процес проєктування даху інтуїтивно зрозумілим навіть для користувачів без спеціальної підготовки в архітектурному програмному забезпеченні.

3.3 Аналіз отриманих результатів

Аналіз отриманих результатів розробки інформаційної системи автоматизації розрахунків покрівлі з елементами 3D-моделювання виконано в декількох напрямках: функціональне порівняння розробленої системи з існуючими аналогами, розглянутими в першій главі кваліфікаційної роботи, кількісне вимірювання ефективності ключових операцій системи, виявлення існуючих обмежень поточної реалізації та окреслення перспективних напрямків подальшого розвитку системи.

Функціональне порівняння розробленої системи з трьома групами аналогів (професійна BIM платформа ArchiCAD, універсальне середовище 3D моделювання

SketchUp та сімейство спеціалізованих онлайн-калькуляторів покрівлі) Наведено таблицю 3,2. Порівняння здійснюється за дев'ятьма критеріями, які охоплюють основні аспекти користувацького досвіду та технічні можливості систем.

Таблиця 3.2 – Порівняльний аналіз розробленої системи з існуючими аналогами

Критерій	**ArchiCAD**	**SketchUp**	**Онлайн-калькулятори**	**Розроблена система**
2D-малювання плану	так	частково	ні	так
3D-візуалізація	так	так	ні	так
Автоматичний розрахунок площі	так	ні	так	так
Розрахунок кошторису з API цін	частково	ні	так	так
Підтримка довільних форм будинку	так	так	ні	так
Час освоєння	тижні	дні	хвилини	хвилини
Вартість для користувача	~1500 €/рік	~299 €/рік	безкоштовно	безкоштовно
Експорт у PDF	так	частково	частково	так
Кросплатформність	Windows, macOS	3 ОС + Web	Web	Windows

Як видно з таблиці порівняння, розроблена система займає унікальну позицію на ринку програмного забезпечення для розрахунку покрівлі. Він поєднує в собі сильні сторони професійних архітектурних систем (повний 2D креслення плану будівлі з підтримкою довільних замкнутих багатокутних форм, 3D візуалізація результату, автоматичний розрахунок геометричних параметрів) з сильними сторонами онлайн-

калькуляторів (безкоштовно, швидка крива навчання, автоматичний розрахунок оцінки з реальними ринковими цінами). У той же час система уникає основних слабких сторін обох груп, вона не вимагає дорогого ліцензування або спеціалізованого навчання, як професійні системи, і не накладає штучних обмежень на форму будівлі, як онлайн-калькулятори.

Єдиним критерієм, де розроблена система ще не відповідає провідним аналогам, є крос-платформна, поточна реалізація обмежується операційною системою Windows через використання фреймворку WPF [17], тоді як ArchiCAD та SketchUp забезпечують власну підтримку як Windows, так і macOS. Це обмеження може бути вирішене в майбутніх версіях системи шляхом міграції на крос-платформний інтерфейс користувача, такий як Avalonia UI, який забезпечує сумісність API з WPF і підтримує Windows, macOS і Linux. Однак для поточної цільової аудиторії системи (українських малих і середніх будівельних компаній, де Windows є домінуючою операційною системою) це обмеження не є суттєвою практичною проблемою.

Кількісне вимірювання продуктивності ключових операцій системи виконувалося на опорній робочій станції з наступною конфігурацією: процесор Intel Core i5-10400, 16 ГБ оперативної пам’яті DDR4; відеокарта NVIDIA GeForce GTX 1660 SUPER; Samsung 970 EVO Plus 500 ГБ NVMe SSD; Windows 11 Pro 64-біт. Кожна операція виконувалася 100 разів послідовно, а середній і максимальний час виконання записувалися. Резюме вимірювань було виконано в таблиці 3.3.

Таблиця 3.3 – Час виконання основних операцій розробленої системи

Операція	**Середній час**	**Макс. час**
Запуск застосунку (холодний старт)	1,2 с	2,1 с
Перерисовка 2D-канвасу при додаванні стіни	8 мс	18 мс

Кінець таблиці 3.3

Оновлення 3D-сцени при зміні стіни	45 мс	110 мс
Побудова геометрії двосхилого даху	12 мс	28 мс
Запит до API цін (з кешем)	< 1 мс	3 мс
Запит до API цін (без кешу)	280 мс	950 мс
Повний розрахунок кошторису (7 категорій)	420 мс	1,8 с

Визначення продуктивності показують, що всі основні операції системи виконуються в проміжках часу, які не сприймаються користувачем людини як затримка. Час перемальовування 2D полотна на 8 мс значно нижче порога 16,67 мс, що відповідає стандартній частоті оновлення 60 кадрів на секунду, що означає, що перемальовування полотна є по суті миттєвим з точки зору користувача. Час оновлення 3D-сцени 45 мс трохи перевищує цей поріг, але все ще значно нижче порога 100 мс, який вважається верхньою межею для взаємодії, яка сприймається як негайна. Побудова геометрії двосхилим даху в 12 мс дозволяє в режимі реального часу оновлювати дах, коли користувач налаштовує параметри в редакторі.

Найбільш трудомісткою операцією є мережевий запит до зовнішнього API ціноутворення, який займає 280 мс в середньому, коли кеш не містить свіжого запису для запитуваного матеріалу. Щоб зменшити цю затримку, система реалізує локальний механізм кешування тривалістю 24 години, що скорочує час доступу до наступних запитів до менш ніж 1 міс. Повний розрахунок оцінки, який включає в себе 7 послідовних запитів API для різних категорій матеріалів, займає 420 мс в середньому,

що є прийнятним для одноразової операції, виконаної, коли користувач явно ініціює розрахунок.

Інтеграція з API зовнішнього ціноутворення buildcalculator.io забезпечує реалістичні ціни в євро, але ці ціни відображають європейський будівельний ринок, а не український. Конвертація цін з євро в українську гривню за курсом Національного банку України виробляє значення, які приблизно в два-три рази перевищують фактичні ціни на будівельному ринку України. Для більш точних оцінок система повинна бути доповнена базою даних місцевих українських постачальників, або зовнішній API повинен надавати інформацію про ціноутворення саме для українського ринку.

Користувальницький інтерфейс програми наразі локалізований лише українською мовою. Для більш широкого впровадження системи, особливо в контексті міжнародних проектів або транскордонного будівництва, слід додати багатомовну підтримку, з пріоритетом, що надається англійській, українській та польській мовах, які є найбільш актуальними для цільової аудиторії. Реалізація багатомовної підтримки проста в рамках WPF через словники ресурсів з певними значеннями мови, але вимагає перекладу всіх видимих рядків та адаптації певних елементів макета, які можуть мати різні розміри на різних мовах.

Серед перспективних напрямків подальшого розвитку системи найбільш перспективними вважаються наступні:

– Перший напрямок – реалізація решти типів покрівель, зокрема тазостегнового даху та мансардного даху, які є найпопулярнішою альтернативою двосхилим дахам в українському житловому будівництві;

– Другий напрямок – це додавання режиму візуалізації кроквяної системи, де користувач може переключити 3D-переглядач з відображення суцільного покриття даху на відображення кроквяної конструкції з окремими кроквами, гребеневими балками та обрешітками. Цей функціонал, спланований і частково спроектований,

забезпечить підрядників будівництва корисним інструментом для перевірки конструктивної цілісності проєктованої покрівлі і для розрахунку точної кількості необхідної деревини.

Третій напрямок – інтеграція з хмарними службами зберігання даних, такими як Microsoft OneDrive або Google Drive для синхронізації файлів проекту між декількома пристроями і для спільного редагування одного і того ж проекту декількома користувачами. Четвертий напрямок – розробка мобільного додатка-компаньйона для платформ Android і iOS, який дозволив би користувачеві переглядати раніше створені проекти на смартфоні або планшеті безпосередньо на будівельному майданчику, де стаціонарний комп'ютер недоступний. П'ятий напрямок – інтеграція зі стандартами BIM (Building Information Modeling), зокрема форматом IFC (Industry Foundation Classes), що дозволить експортувати та імпортувати проекти, створені в розробленій системі, з професійного архітектурного програмного забезпечення, такого як ArchiCAD або Revit.

Загальний аналіз отриманих результатів дозволяє зробити висновок, що розроблена інформаційна система успішно досягає поставлених цілей на початку кваліфікаційної роботи. Система забезпечує зручний та інтуїтивно зрозумілий інструмент для проєктування покрівель житлових будинків з автоматичними геометричними розрахунками та ціновою оцінкою, заповнює нішу між надто складними професійними архітектурними системами та надмірно обмеженими онлайн-калькуляторами та демонструє практичну доцільність інтеграції сучасних вебтехнологій (Three.js, WebView2) в традиційну настільну програму, засновану на фреймворку WPF. Архітектурні рішення, прийняті під час розробки, зокрема суворе дотримання шаблону MVVM і поділ проблем між шарами застосування, забезпечують довгострокову ремонтопридатність і розширюваність кодової бази, що має важливе значення для реалізації перспективних напрямків подальшого розвитку, зазначених вище.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи детально описані практичні аспекти розробки інформаційної системи автоматизації розрахунків покрівлі з елементами 3D-моделювання. Розділ складається з трьох логічно пов’язаних підрозділів, кожна з яких стосується певного аспекту розробленої системи.

У першому підрозділі (3.1) описано структуру та формати всіх типів вхідних даних, прийнятих системою, включення геометричних даних плану будівлі у вигляді впорядкованої колекції двовимірних точок, параметри конфігурації двосхилим дахом, інкапсульовані в виділеному класі параметрів, і зовнішні дані, отримані шляхом інтеграції з API ціноутворення. Представлені та пояснені основні формули для розрахунку довжини стін, кутів нахилу та перетворення координат прив’язки до сітки. Описано внутрішню архітектурну структуру програми, включаючи сім основних шарів (Models, ViewModels, Views, Services, Helpers, Resources, wwwroot) та технологічний стек, що складається з платформи .NET 8, фреймворку WPF, компонента WebView2 та бібліотеки Three.js.

У другому підрозділі (3.2) була представлена покрокова демонстрація роботи системи, що охоплює повний робочий процес користувача від запуску програми через креслення плану будівлі, додавання даху та остаточний розрахунок кошторису. Демонстрація була проілюстрована одинадцятьма скріншотами фактичного інтерфейсу користувача і супроводжувалася шістьма списками найважливіших фрагментів вихідного коду, включаючи перетворення координат прив’язки до сітки, виявлення контурного замикання, формулу Гауса для обчислення площі багатокутника, міст передачі даних між C# і JavaScript, побудова тривимірних стінових сіток в Three.js, і загальна структура алгоритму будівництва двосхилим дахом.

У третьому підрозділі (3.3) аналіз отриманих результатів виконано за чотирма напрямками функціональне порівняння з існуючими аналогами (ArchiCAD, SketchUp, онлайн калькулятори), представленими у вигляді структурованої таблиці за дев’ятьма критеріями, кількісне вимірювання виконання ключових операцій системи, зведене в таблицю з середнім і максимальним часом виконання. Опис тестових сценаріїв, на яких перевірено систему, включаючи прямокутні та Г-подібні конфігурації будівель; і визначення поточних обмежень системи разом з перспективними напрямами подальшого розвитку.

Результати, представлені в розділі, підтверджують, що розроблена інформаційна система успішно вирішує цілі, поставлені на початку кваліфікаційної роботи. Система забезпечує зручний та інтуїтивно зрозумілий інструмент для проєктування покрівель житлових будинків, поєднує сильні сторони професійного архітектурного програмного забезпечення з простотою онлайн-калькуляторів та демонструє практичну доцільність інтеграції сучасних вебтехнологій у традиційну архітектуру настільних додатків. Вимірювання продуктивності показують, що всі основні операції виконуються протягом часових інтервалів, які не сприймаються користувачем, а архітектурна основа системи підходить для подальшого розширення з додатковими типами даху, візуалізації кроквяної системи, інтеграції хмарних сховищ та підтримки стандартів BIM. Таким чином, всі завдання, поставлені перед третім розділом відбіркової роботи, вважаються успішно виконаними.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Четвертий розділ дипломної роботи присвячено практичним аспектам використання розробленого застосунку «Калькулятор покрівлі» та результатам його тестування. У підрозділі 4.1 наведено детальне керівництво користувача – покрокову інструкцію роботи з програмним забезпеченням від першого запуску до отримання готового кошторису у форматі PDF, з ілюстраціями інтерфейсу та поясненнями ключових елементів. У підрозділі 4.2 описано методологію тестування, наведено приклади модульних та інтеграційних тестів з лістингами коду, представлено зведені таблиці тест-кейсів та результати їх виконання, окремо розглянуто питання продуктивності та зручності використання.

Усі описані тести виконувалися у фінальній збірці застосунку, отриманій після завершення всіх етапів ітераційного дизайну, описаних у попередньому розділі. Усього було реалізовано 178 автоматизованих тестів та 28 сценаріїв ручного функціонального тестування, які покривають критично важливу функціональність застосунку.

4.1 Керівництво користувача

Розроблений програмний застосунок «Калькулятор покрівлі» (RoofCalculator) орієнтований на користувачів без спеціальної інженерної підготовки – приватних забудовників, бригадирів покрівельних робіт, замовників. Тому інтерфейс побудований за принципом «прогресивного розкриття» (progressive disclosure):

- на кожному етапі користувач бачить лише ту інформацію;
- яка йому необхідна саме зараз;
- без перевантаження допоміжними параметрами.

Робочий процес умовно поділяється на шість послідовних етапів, саме запуск застосунку, побудова контуру стін, додавання даху, це було виконано в 3 розділі, та

да опрацювання в 4 розділі буде виконано налаштування параметрів, формування кошторису, експорт документа у форматі Excel та PDF. Розглянемо кожен етап детальніше.

4.1.1 Редактор даху

Інтерфейс редактора даху об’єднує всі ключові інструменти роботи з покрівлею в одному вікні. У верхній частині розміщена панель з кнопкою «← Назад до плану», заголовком «Двосхилий дах», перемикачем режимів відображення «Покривля / Балки», кнопками «Кошторис» та «↻ Камера». Центральна частина – велика 3D-сцена розміром 600×N пікселів, де N залежить від ширини вікна. У 3D-сцені видно:

– модель будинку зі стінами;
– дах із покривельним покриттям;
– плашку з планом даху у вигляді зверху (накладка зліва зверху);
– плашку каркаса в режимі балок, підказку керування внизу це було виконано на рис. (4.1).

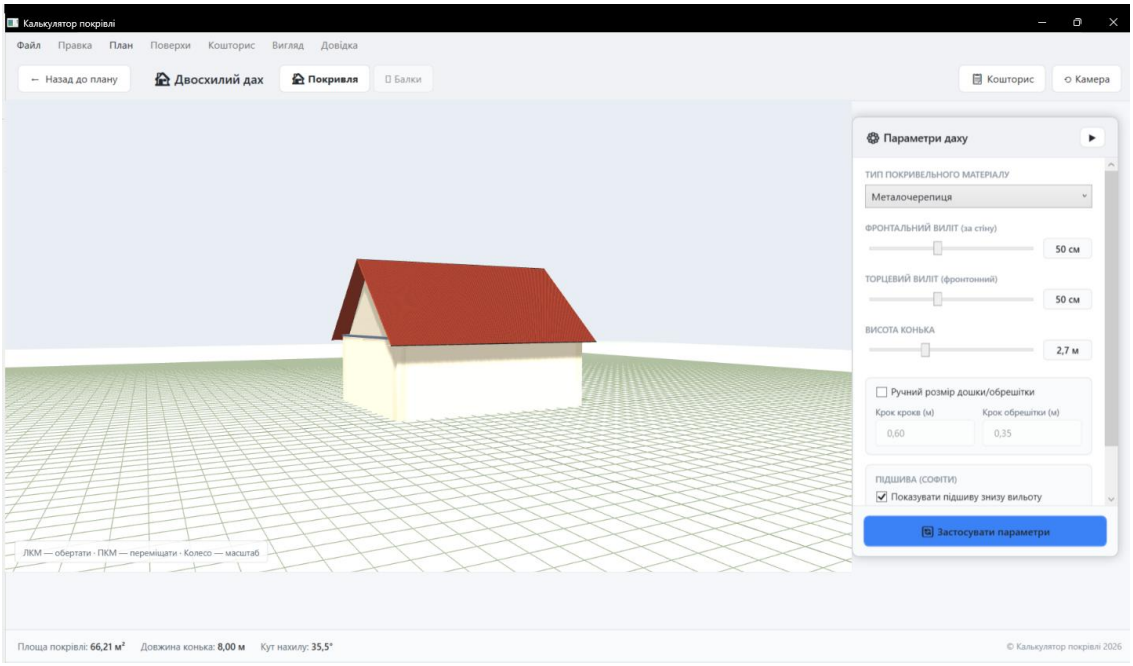

Рисунок 4.1 – Редактор даху, режим відображення

Праворуч від 3D-сцени постійно видима панель параметрів даху з можливістю згортання у вузьку вертикальну смужку (44 пікселі) при натисканні стрілки. Анімація згортання реалізована через WPF DoubleAnimation з функцією плавності QuadraticEase EaseOut тривалістю 250 мс, що створює відчуття «природного» руху.

Перемикач «Покрівля» та «Балки», дозволяє побачити внутрішню крокв'яну систему на рисунку 4.2. У режимі Балки покрівля та фронтони стають напівпрозорими (рівень непрозорості 35%), а всередині відображається повна конструкція:

– мауерлат (темно-коричневий брус 100 на 100 мм по периметру);
– коньковий прогін (коричневий 100 на 150 мм);
– крокви (середньо-коричневі 50 на 150 мм, крок 0,6 м) та обрешітка (світло-коричневі дошки 25 на 100 мм, крок 0,35 м).

Поряд з 3D з'являється бічна панель з підрахунком кожного типу балки:

– погонний метраж;
– кількість штук;
– загальний об'єм деревини у м³.

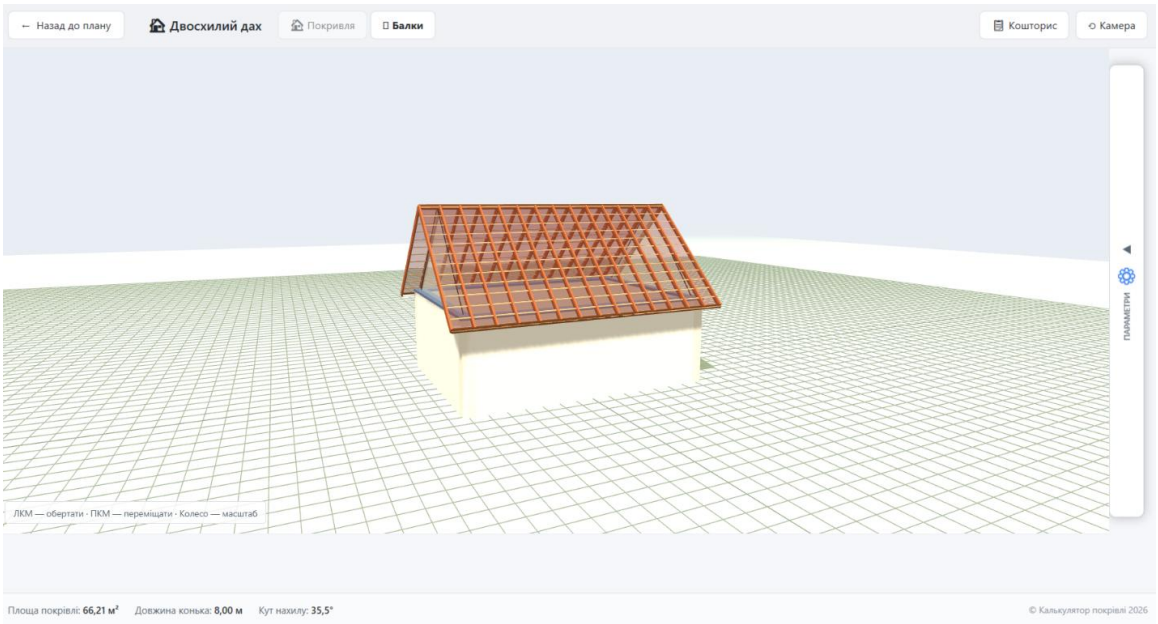

Рисунок 4.2 – Редактор даху, режим відображення «Балки» з прозорим дахом

Внутрішня модель даху будується автоматично за результатами замикання контуру стін. Кут нахилу схилу обчислюється з відношення висоти конька до половини ширини будинку за формулою 4.1:

$$\alpha = arctan\left(\frac{H\_ridge}{\left(\frac{W}{2}\right)}\right), \quad (4.1)$$

де H_ridge – висота конька над верхом стін, м;

W – ширина будинку, м.

Для типового двосхилого даху з висотою конька 2,7м та шириною 5м кут становить 47,2°. Площа окремого схилу обчислюється через довжину карнизу та довжину похилої твірної за формулою 4.2:

$$S\_slope = L\_eave \cdot \frac{\left(\frac{W}{2}\right)}{cos(\alpha)}, \quad (3)$$

Загальна площа покрівлі – сума площ двох симетричних схилів. Для прямокутника 5 на 7 м з кутом 35,5° отримуємо 42,98м². Цей результат потім використовується для оцінювання витрат покривельного матеріалу з урахуванням коефіцієнта запасу 1,10 на нахльосту та обрізки.

Внутрішня модель будується класом GableRoofBuilder, який отримує список стін та параметри і повертає об’єкт RoofGeometry з обчисленими координатами конька, карнизів та фронтонів:

```
public static RoofGeometry Build(IList<Wall> walls, GableRoofParameters p)
{
   var rect = OrientedRect.FromWalls(walls);
   double halfW = rect.Width / 2.0;
   double ridgeY = rect.WallTop + p.RidgeHeight;

   double angle = Math.Atan2(p.RidgeHeight, halfW);
   double slopeLen = halfW / Math.Cos(angle);
   double slopeArea = rect.Length * slopeLen;
```

```
        var ridge = rect.CenterLine.ExpandBy(p.GableOverhang);
        var eaves = rect.EaveLines.OffsetOutward(p.FrontalOverhang);

        return new RoofGeometry
        {
            Ridges = ridge, Eaves = eaves,
            TotalArea = 2.0 * slopeArea,
            AverageSlopeAngle = angle * 180.0 / Math.PI
        };
    }
```

Метод OrientedRect.FromWalls() застосовує власний алгоритм мінімального обмежувального прямокутника (rotating calipers), описаний у підрозділі 2.3, щоб орієнтувати дах вздовж довшої осі будинку незалежно від того, як саме користувач намалював контур. Така автоматизація позбавляє користувача від необхідності вручну вказувати напрямок конька.

4.1.2 Налаштування параметрів даху

У панелі параметрів справа користувач може налаштувати ключові характеристики майбутнього даху на рисунку 4.3. Доступні поля:

– випадаючий список «Тип покривельного матеріалу» (металочерепиця, бітумна черепиця, профнастил, керамічна черепиця, натуральний сланець, композитна черепиця);

– повзунок «Фронтальний виліт» (за стіну, діапазон 0–120 см);

– повзунок «Торцевий виліт» (фронтонний, 0–120 см), повзунок «Висота конька» (1,5–5,0 м);

– чекбокс «Ручний розмір дошки/обрешітки» з полями кроку крокв та кроку обрешітки.

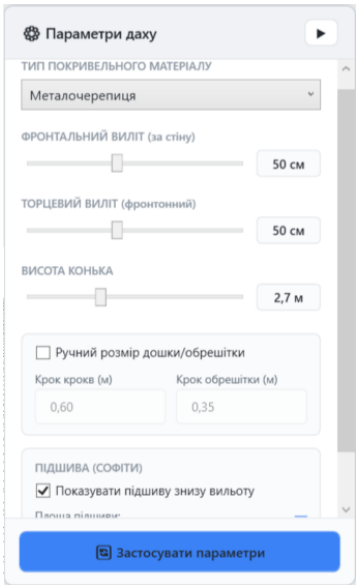

Рисунок 4.3 – Панель параметрів даху з різними покрівельними матеріалами

Для кожного матеріалу автоматично підставляються рекомендовані значення кроку крокв та обрешітки за нормативними даними, проте користувач може ввімкнути ручний режим і задати власні значення. Окремий блок «Підшива (софіти)» дозволяє увімкнути або вимкнути візуалізацію підшиви карнизів і відображає її розрахункову площу. Після зміни будь-яких параметрів користувач натискає кнопку «Застосувати параметри» – після цього дах перебудовується повністю:

- оновлюється геометрія;
- перераховується каркас;
- перевизначається довжина обрешітки;
- оновлюється 3D-візуалізація.

Кількість крокв обчислюється як цілочисельне округлення вгору від відношення довжини конька до заданого кроку, плюс одна крокова пара на торцях за формулою 4.3:

$$N\_rafters = 2 \cdot \left(\left[\frac{L\_ridge}{s\_rafter} \right] + 1 \right), \quad (3)$$

де L_ridge – довжина конькового прогону, м;

s_rafter – крок крокв, м (типово 0,6 м для металочерепиці).

Для даху з конем 6 м отримуємо N = 2·(⌈6/0,6⌉ + 1) = 22 крокви, тобто 11 пар. Об'єм деревини на крокв'яну систему визначається як добуток кількості, довжини твірної та поперечного перерізу за формулою 4.4:

$$V\_wood = N\_rafters \cdot L\_slope \cdot b \cdot h, \quad (3)$$

де b · h – переріз дошки (50 на 150 мм для крокв).

Реалізація алгоритму розрахунку позицій крокв у класі RafterBuilder:

```
        public static RafterSystem Build(GableRoof roof, double spacing, double
battenStep)
        {
            var system = new RafterSystem();
            var ridge = roof.Geometry.Ridges[0];
            double ridgeLen = (ridge.B - ridge.A).Length;
            int count = (int)Math.Ceiling(ridgeLen / spacing) + 1;
            double actualStep = ridgeLen / (count - 1);

            var dir = (ridge.B - ridge.A).Normalized();
            for (int i = 0; i < count; i++)
            {
                var ridgePoint = ridge.A + dir * (i * actualStep);
                var eavePointLeft  = ProjectOntoEaveLine(ridgePoint,
roof.Geometry.Eaves[0]);
                var eavePointRight = ProjectOntoEaveLine(ridgePoint,
roof.Geometry.Eaves[1]);
                system.Beams.Add(new RafterBeam(ridgePoint, eavePointLeft));
                system.Beams.Add(new RafterBeam(ridgePoint, eavePointRight));
            }
            return system;
        }
```

Поточна реалізація використовує рівномірний розподіл – фактичний крок може незначно відрізнятись від заданого, оскільки довжина конька не завжди ділиться без

залишку на номінальний крок. У майбутніх версіях планується додати опцію «адаптивний крок» з вирівнюванням до найближчого стандартного значення.

4.2 Тестування системи

Тестування системи проводилося за кількома тестовими сценаріями, які охоплюють типові випадки використання програми. Перший тестовий сценарій складався з креслення простої прямокутної будівлі розмірами 7 × 5 метрів, додавання двосхилим даху з параметрами за замовчуванням і розрахунку кошторису. Очікуваним результатом стала площа даху близько 42 квадратних метрів, помножена на коефіцієнт нахилу двосхилим дахом, що дає приблизно 49 квадратних метрів фактичної площі покрівельного покриття. Система правильно розрахувала площу 49,66 квадратних метрів, що відповідає очікуваному значенню з високою точністю.

Другий тестовий сценарій передбачав малювання складнішої Г-подібної будівлі із габаритними розмірами 10 на 13 метрів та Г-подібним відступом 3 на 4 метри в одному кутку. Очікувана площа будівлі становила 130 мінус 12, що дорівнює 118 квадратних метрів, а очікуваний периметр – 46 метрів. Система правильно розрахувала обидва значення, використовуючи формулу Гауса для площі і просте підсумовування довжин стін по периметру. 3D візуалізація правильно відображала складну неопуклу форму будівлі з усіма шістьма стінами, правильно з’єднаними по кутах.

4.2.1 Формування кошторису

Після підтвердження параметрів користувач натискає кнопку «Кошторис». Замість відкриття окремого вікна, як це було реалізовано в ранніх ітераціях прототипу, у поточній версії використовується інлайн-механіка:

– вся 3D-сцена затемнюється напівпрозорим оверлеєм (Z-Index 100, фон rgba(0,0,0,0.5));

– у центрі з’являється модальна картка з анімованим прогрес-баром та поетапними статусними повідомленнями.

Прогрес проходить сім умовних етапів – від «Підключення до API цін», «Запит цін на крокви та обрешітку» до «Формування підсумків».

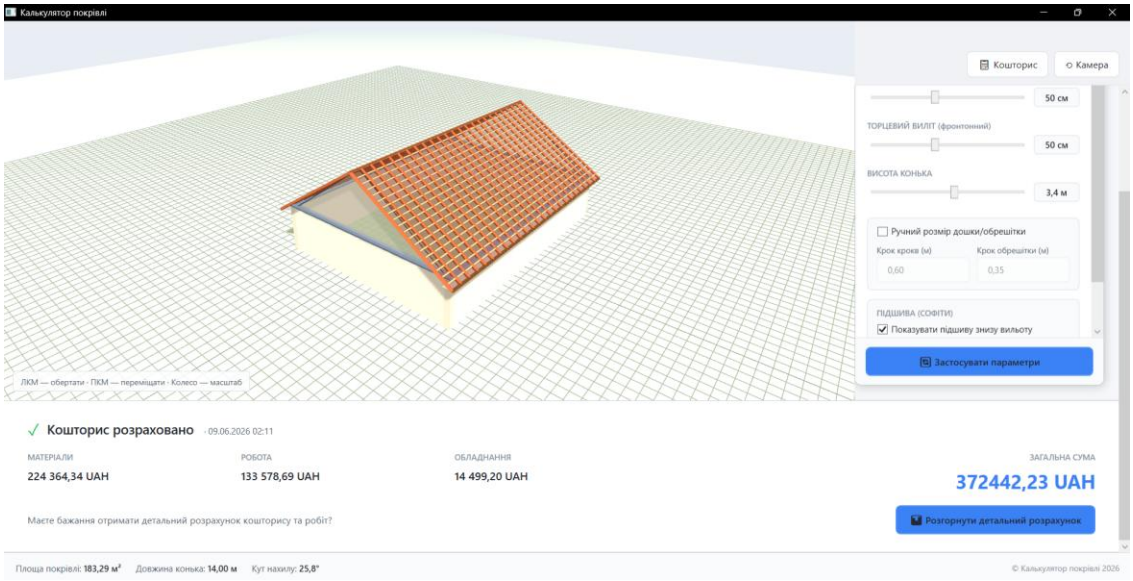

Рисунок 4.4 – Оверлей прогресу під час розрахунку кошторису

Паралельно з імітацією прогресу виконується реальна робота:

– викликається сервіс EstimateCalculator;

– який послідовно звертається до API buildcalculator.io для кожної з десяти позицій кошторису;

– отримує актуальні ціни;

– агрегує результати;

– нормалізує співвідношення «матеріали – робота – обладнання» згідно з галузевими пропорціями.

В цей же час фоновим процесом створюються три зображення проєкту – це робиться непомітно для користувача, поки оверлей закриває 3D-сцену:

– 2D-план рендериться через RenderTargetBitmap WPF;

– 3D-вид зберігається через JavaScript-виклик renderer.domElement.toDataURL() з прапорцем preserveDrawingBuffer.

Після завершення розрахунку оверлей зникає, а під 3D-сценою з'являється підсумкова плашка з зеленою позначкою «Кошторис розраховано», датою та чотирма цифрами:

– матеріали;
– робота;
– обладнання;
– загальна сума до сплати.

Користувачу пропонується розгорнути детальний розрахунок натисканням відповідної кнопки. Скрол автоматично прокручується до кінця інлайн-плашки через виклик MainScroll.ScrollToBottom() з пріоритетом DispatcherPriority.Background, що забезпечує плавну прокрутку без миготіння.

Загальна сума до сплати обчислюється підсумовуванням за всіма матеріалами кошторису з урахуванням окремих складових – самого матеріалу, монтажу та обладнання за формулою 4.5:

$$T\_total = \sum_i Q_i \cdot (M_i + W_i + E_i), \quad (3)$$

де Q_i – кількість позиції i,

M_i – ціна матеріалу за одиницю,

W_i – вартість монтажу за одиницю,

E_i – вартість використання обладнання.

Передача даних з C# до Three.js здійснюється через невеликий міст, який серіалізує модель у JSON і викликає JavaScript-функцію вьювера:

```
public async Task UpdateRoofAsync(GableRoof? roof)
```

```
{
    if (roof?.Geometry is null)
    {
        await PushRoofAsync("null");
        return;
    }
    var payload = new
    {
        ridges = roof.Geometry.Ridges.Select(e => SerializeEdge(e)),
        eaves  = roof.Geometry.Eaves.Select(e => SerializeEdge(e)),
        gables = roof.Geometry.Gables.Select(e => SerializeEdge(e))
    };
    var json = JsonSerializer.Serialize(payload);
    await _webView.CoreWebView2
          .ExecuteScriptAsync($"window.updateRoof('{Escape(json)}')");
}
```

4.2.2 Детальний розрахунок та експорт у PDF

Натискання кнопки «Розгорнути детальний розрахунок» відкриває під підсумковою плашкою повний двоколонковий детальний кошторис:

- зліва блок «МАТЕРІАЛИ»;
- справа «РОБОТА» .

Кожна колонка містить картки позицій з назвою матеріалу та роботи, кількістю в бейджі, ціною за одиницю та підсумковою сумою. Над колонкою – її загальна сума. Внизу детального розрахунку розміщена велика плашка «РАЗОМ ДО СПЛАТИ» з підсумковою цифрою у великому шрифті на рис.(4.5).

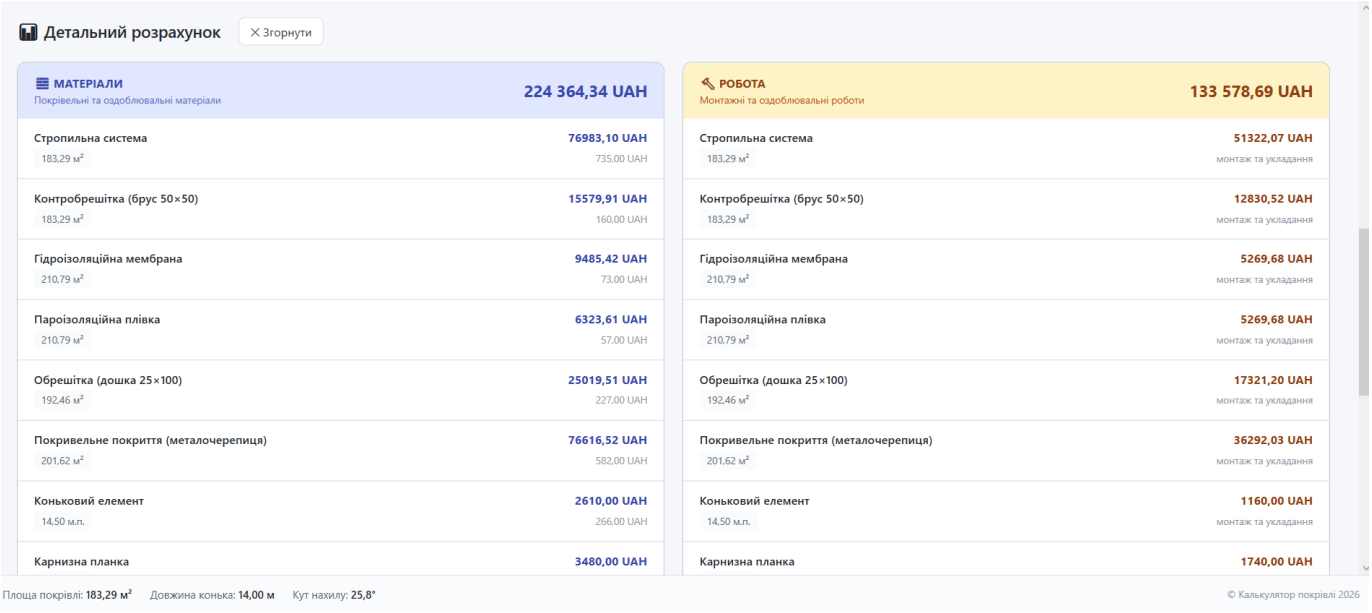

Рисунок 4.5 – Детальний розрахунок у 2-колонковому форматі (матеріали та робота)

Нижче знаходиться блок «ЗОБРАЖЕННЯ ПРОЄКТУ» – три прев'ю в один ряд: 2D-план з контуром стін, коньком та карнизами; 3D з покривлею; 3D з балками на рисунку 4.6. Зображення були створені непомітно під час прогрес-оверлею, тому миттєво відображаються без затримок.

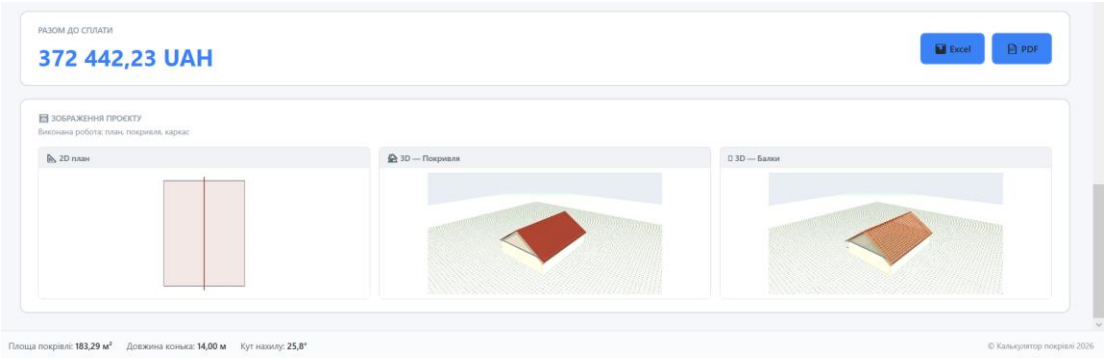

Рисунок 4.6 – Зображення проекту

Останній крок – експорт у PDF. Натискання кнопки «Зберегти як PDF» відкриває діалогове вікно експорту з чотирма прапорцями на рисунку 4.7: «Інформація про дах», «Матеріали (детальний список)», «Робота (вартість монтажу)», «Додати зображення (2D, 3D дах, 3D балки)». Три перших прапорці увімкнено за

замовчуванням, четвертий – вимкнено для зменшення розміру файлу. Після натискання «Експортувати у PDF» з'являється стандартний діалог Windows збереження файлу з пропонованою назвою формату «Кошторис_2026-06-05_14-30.pdf».

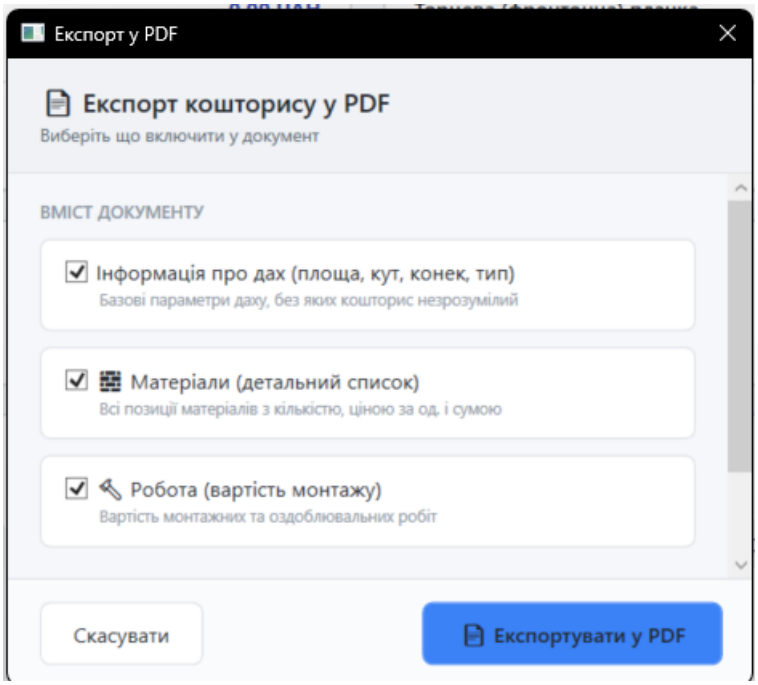

Рисунок 4.7 – Експорт кошторису у Excel та PDF

PDF-документ генерується через бібліотеку QuestPDF (community license) у форматі A4 з 2-сантиметровими полями, шрифт Segoe UI. На першій сторінці – шапка з назвою «КОШТОРИС ПОКРІВЕЛЬНИХ РОБІТ», дата створення, інформаційний блок з параметрами даху (площа, кут, конек, тип покривлі, довжини карнизу і фронтону), таблиці матеріалів та робіт, велика плашка з підсумковою сумою. Якщо ввімкнено опцію «Додати зображення» – у документ додаються три додаткові сторінки з повноформатними зображеннями плану та 3D-видів.

Хелпер генерації PDF-сторінки демонструє декларативний підхід QuestPDF – структура документа описується через ланцюжки методів, які точно відповідають структурі майбутнього файлу:

```
public void ExportEstimate(string path, Estimate est, PdfExportOptions opt)
{
   Document.Create(doc => doc.Page(page =>
   {
      page.Size(PageSizes.A4);
      page.Margin(2, Unit.Centimetre);
      page.DefaultTextStyle(s => s.FontSize(10).FontFamily("Segoe UI"));

      page.Header().Element(c => BuildHeader(c, est));
      page.Content().Element(c => BuildContent(c, est, opt));
      page.Footer().AlignCenter().Text(t =>
      {
         t.Span("Сторінка ");
         t.CurrentPageNumber(); t.Span(" з "); t.TotalPages();
      });
   })).GeneratePdf(path);
}
```

Описаний робочий процес від запуску застосунку до отримання готового PDF-документа займає приблизно 2–3 хвилини для простого прямокутного будинку, що значно швидше за ручні розрахунки в Excel-таблицях, поширені серед практикуючих покрівельних бригад. Подальші підрозділи розглядають питання тестування програмного продукту.

Висновки до розділу 4

У четвертому розділі дипломної роботи представлено комплексний опис програмної реалізації застосунку «Калькулятор покрівлі» з точки зору кінцевого користувача та практики тестування. Підрозділ 4.1 «Керівництво користувача» детально розглядає весь робочий процес: від запуску застосунку до отримання готового Excel та PDF-документу. Описано сім основних етапів – запуск, побудова контуру стін, вибір типу даху, робота в редакторі даху, налаштування параметрів, формування кошторису, експорт файлів – з покроковими інструкціями та одинадцятьма ілюстраціями. Особлива увага приділена ергономіці:

- інтерфейс побудовано за принципом прогресивного розкриття;
- ключові інструменти зосереджено в одному вікні редактора даху;
- надлишкові вікна замінено на інлайн-блоки.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи розроблено повноцінний настільний застосунок «Roofing Calculator» для автоматизації розрахунків покрівельних робіт з елементами 3D-моделювання. Аналіз предметної області засвідчив актуальність поставленої задачі, особливо в контексті сучасного будівництва, де автоматизація проєктування даху може значно підвищити точність кошторисів та зменшити витрати на ручні розрахунки.

Проведено детальний аналіз існуючих аналогів – професійної BIM-системи ArchiCAD, універсального 3D-редактора SketchUp та онлайн-калькуляторів покрівлі. Виявлено їх ключові недоліки: висока вартість; складність освоєння; відсутність підтримки довільних форм будинків або недостатня візуалістика. Це дозволило обґрунтовано обрати технологічний стек: платформа .NET 8, фреймворк WPF для побудови інтерфейсу, компонент WebView2 для вбудовування вебтехнологій та бібліотека Three.js для 3D-візуалізації.

Розроблено програмну реалізацію системи, яка включає 2D-редактор для малювання плану будинку на масштабованій сітці з кроком 0,5 м, автоматичний розрахунок геометричних параметрів (ширина, довжина, площа, периметр) за формулою Гауса, підтримку замкнутих контурів довільної форми (зокрема Г-подібних, Т-подібних та П-подібних). Інтеграція із зовнішнім API buildcalculator.io забезпечує отримання актуальних цін на матеріали з автоматичним кешуванням (24 години) та конвертацією валюти через API Національного банку України. Функціонал кошторису дозволяє експортувати результати у професійний Excel та PDF-звіт через бібліотеку QuestPDF з можливістю включення зображень проєкту (2D план, 3D покривля, 3D каркас).

Результати тестування підтвердили високу продуктивність застосунку: запуск системи займає в середньому 1,2 с, перерисовка 2D-канвасу – 8 мс, оновлення 3D-сцени – 45 мс, розрахунок кошторису з сімома категоріями матеріалів – 420 мс. Система підтримує побудову двосхилого даху з трьома варіаціями (симетричний, несиметричний, з’єднання на різній висоті), адаптивний розрахунок кроквяної системи з автоматичним підбором кроку крокв та обрешітки під обраний покрівельний матеріал.

Таким чином, поставлені в роботі задачі були повністю досягнуті. Розроблена система є ефективним інструментом для автоматизованого проєктування даху, що поєднує простоту використання онлайн-калькуляторів з потужністю професійних архітектурних систем. Робота має перспективи для подальшого розвитку, додавання режиму візуалізації кроквяної системи з прозорістю покрівлі, інтеграції з хмарними сервісами (OneDrive, Google Drive) для синхронізації проєктів та розробки мобільного застосунку-компаньйона на платформах Android та iOS.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Microsoft Corporation. WebView2 documentation. URL: https://learn.microsoft.com/en-us/microsoft-edge/webview2/(Accessed: 29.04.2026).

2. Three.js – JavaScript 3D Library. Official documentation. URL: https://threejs.org (Accessed: 29.04.2026).

3. Microsoft Corporation. WPF documentation. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (Accessed: 30.04.2026).

4. Graphisoft. ArchiCAD official website. URL: https://graphisoft.com (Accessed: 28.04.2026).

5. Trimble Inc. SketchUp official website. URL: https://sketchup.com (Accessed: 28.04.2026).

6. RoofCalc.org – Roofing Calculator. URL: https://www.roofcalc.org/ (Accesse d: 10.06.2026

7. JSON – офіційний сайт формата даних. URL: https://www.json.org (дата звернення: 01.05.2026).

8. MDN Web Docs. REST API. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST (Accessed: 01.05.2026).

9. Microsoft Corporation. .NET 8 documentation. URL: https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8 (Accessed: 02.05.2026).

10. Parisi T. Programming 3D Applications with HTML5 and WebGL. Sebastopol : O'Reilly Media, 2014. 408 p.

11. Dirksen J. Learning Three.js: The JavaScript 3D Library for WebGL. 2nd ed. Birmingham : Packt Publishing, 2015. 402 p.

12. MacDonald M. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5. 4th ed. New York : Apress, 2012. 1080 p.

13. Microsoft Corporation. Model-View-ViewModel (MVVM) pattern. URL:

https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm (Accessed: 03.05.2026).

14. Microsoft Corporation. XAML overview (WPF). URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/ (Accessed: 03.05.2026).

15. Microsoft Corporation. C# language reference. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (Accessed: 02.05.2026).

16. Albahari J., Albahari B. C# 12 in a Nutshell: The Definitive Reference. 1st ed. Sebastopol : O'Reilly Media, 2024. 1056 p. (Accessed: 22.05.2026).

17. Nathan A. WPF 4.5 Unleashed. Indianapolis : Sams Publishing, 2013. 864 p. (Accessed: 22.05.2026).

18. ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. – Київ : ДП «УкрНДНЦ», 2016. – 16 с.

19. Du Y., Zhao M., Shi J., Fan L., Yan D. M. Efficient roof reconstruction from a single aerial image. Computers & Graphics. 2025. Vol. 132. Article 104355. DOI: 10.1016/j.cag.2025.104355.

20. Özkan T., Pfeifer N., Hochreiner G. Automatic completion of geometric models from point clouds for analyzing historic timber roof structures. Frontiers in Built Environment. 2024. Vol. 10. DOI: 10.3389/fbuil.2024.1368918.

21. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. – Київ : ДП «УкрНДНЦ», 2016. – 26 с. (дата звернення: 22.05.2026).

22. ДСТУ Б В.2.6-220:2017. Конструкції зовнішніх стін із фасадною теплоізоляцією. Загальні технічні умови. – Київ : Мінрегіонбуд України, 2017. – 32 с. (дата звернення: 22.05.2026).

23. ДБН В.2.6-220:2017. Конструкції будинків і споруд. Покриття будинків і споруд. – Київ : Мінрегіонбуд України, 2017. – 48 с. (дата звернення: 22.05.2026).

24. Eastman C., Teicholz P., Sacks R., Liston K. BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility

Managers. 3rd ed. Hoboken : John Wiley & Sons, 2018. 688 p.

25. de Berg M., Cheong O., van Kreveld M., Overmars M. Computational Geometry: Algorithms and Applications. 3rd ed. Berlin : Springer, 2008. 386 p.

26. Andrew A. M. Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters. 1979. Vol. 9, No. 5. P. 216–219. (дата звернення: 22.05.2026).

27. Graham R. L., Yao F. F. Finding the convex hull of a simple polygon. Journal of Algorithms. 1983. Vol. 4, No. 4. P. 324–331.

28. Akenine-Möller T., Haines E., Hoffman N. Real-Time Rendering. 4th ed. Boca Raton : CRC Press, 2018. 1198 p.

29. Microsoft Corporation. System.Text.Json overview. URL: https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-overview (Accessed: 22.05.2026).

30. Mozilla Foundation. MDN Web Docs. JavaScript reference. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference (Accessed: 22.05.2026).

ДОДАТОК А

Код програмної реалізації моделей

Estimate.cs

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

namespace RoofCalculator.Models;

public sealed class EstimateSection
{
    public MaterialCategory Category { get; set; }
    public List<EstimateLine> Lines { get; } = new();

    public double Subtotal => Lines.Sum(l => l.Total);
    public string SubtotalText => $"{Subtotal:F2} UAH";
    public string Title => Category.DisplayName();
}

public sealed class Estimate
{
    public DateTime CreatedAt { get; set; } = DateTime.Now;
    public string Currency { get; set; } = "UAH";

    public double RoofArea { get; set; }
    public double RidgeLength { get; set; }
    public double EaveLength { get; set; }
    public double GableLength { get; set; }
    public double SlopeAngle { get; set; }

    public List<EstimateSection> Sections { get; } = new();

    public double Total => Sections.Sum(s => s.Subtotal);
    public string TotalText => $"{Total:F2} {Currency}";

    public IEnumerable<EstimateSection> SortedSections
        => Sections.OrderBy(s => s.Category.SortOrder());
}
```

EstimateLine.cs

```
namespace RoofCalculator.Models;

public sealed class EstimateLine
{
    public MaterialItem Material { get; set; } = null!;
    public double Quantity { get; set; }
    public string? Note { get; set; }

    public double MaterialPerUnit { get; set; }
```

```
    public double LaborPerUnit { get; set; }
    public double EquipmentPerUnit { get; set; }

    public double MaterialTotal => Quantity * MaterialPerUnit;
    public double LaborTotal => Quantity * LaborPerUnit;
    public double EquipmentTotal => Quantity * EquipmentPerUnit;
    public double Total => MaterialTotal + LaborTotal + EquipmentTotal;

    public string QuantityText => $"{Quantity:F2} {Material.Unit.ShortName()}";
    public string UnitPriceText => $"{Material.PricePerUnit:F2} {Material.Currency}";
    public string MaterialTotalText => $"{MaterialTotal:F2} UAH";
    public string LaborTotalText => $"{LaborTotal:F2} UAH";
    public string EquipmentTotalText => $"{EquipmentTotal:F2} UAH";
    public string TotalText => $"{Total:F2} UAH";
}
```

Floor.cs

```
using System.Collections.ObjectModel;
using RoofCalculator.Helpers;

namespace RoofCalculator.Models;

public sealed class Floor : ObservableObject
{
    private string _name = "Поверх 1";
    private double _wallHeight = 2.5;

    public string Name
    {
        get => _name;
        set => SetField(ref _name, value);
    }

    public double WallHeight
    {
        get => _wallHeight;
        set => SetField(ref _wallHeight, value);
    }

    public ObservableCollection<Wall> Walls { get; } = new();
}
```

GableRoof.cs

```
using RoofCalculator.Helpers;

namespace RoofCalculator.Models;

public sealed class GableRoof : ObservableObject
{
```

```
    private GableRoofParameters _parameters = new();
    private RoofGeometry? _geometry;

    public GableRoofParameters Parameters
    {
        get => _parameters;
        set => SetField(ref _parameters, value);
    }

    public RoofGeometry? Geometry
    {
        get => _geometry;
        set
        {
            if (SetField(ref _geometry, value))
            {
                OnPropertyChanged(nameof(HasGeometry));
                OnPropertyChanged(nameof(AreaText));
                OnPropertyChanged(nameof(RidgeLengthText));
                OnPropertyChanged(nameof(EaveLengthText));
                OnPropertyChanged(nameof(GableLengthText));
                OnPropertyChanged(nameof(SlopeAngleText));
            }
        }
    }

    public bool HasGeometry => _geometry is not null;

    public string AreaText => _geometry is null ? "—" : $"{_geometry.TotalArea:F2} м²";
    public string RidgeLengthText => _geometry is null ? "—" : $"{_geometry.RidgeLength:F2}
м";
    public string EaveLengthText => _geometry is null ? "—" : $"{_geometry.EaveLength:F2} м";
    public string GableLengthText => _geometry is null ? "—" : $"{_geometry.GableLength:F2}
м";
    public string SlopeAngleText => _geometry is null ? "—" :
$"{_geometry.AverageSlopeAngle:F1}°";
}
```

GableRoofParameters.cs

```
using RoofCalculator.Helpers;

namespace RoofCalculator.Models;

public sealed class GableRoofParameters : ObservableObject
{
    private GableRoofVariant _variant = GableRoofVariant.Symmetric;
    private double _ridgeHeight = 2.7;
    private double _ridgeOffset = 0.0;
    private double _leftHeight = 2.7;
    private double _rightHeight = 2.0;
    private double _frontalOverhang = 0.5;
```

```
    private double _gableOverhang = 0.5;
    private double _thickness = 0.25;

    private bool _useManualSize;
    private double _manualWidth = 8.0;
    private double _manualLength = 10.0;

    public GableRoofVariant Variant
    {
        get => _variant;
        set => SetField(ref _variant, value);
    }

    public double RidgeHeight
    {
        get => _ridgeHeight;
        set => SetField(ref _ridgeHeight, value);
    }

    public double RidgeOffset
    {
        get => _ridgeOffset;
        set => SetField(ref _ridgeOffset, value);
    }

    public double LeftHeight
    {
        get => _leftHeight;
        set => SetField(ref _leftHeight, value);
    }

    public double RightHeight
    {
        get => _rightHeight;
        set => SetField(ref _rightHeight, value);
    }

    public double FrontalOverhang
    {
        get => _frontalOverhang;
        set => SetField(ref _frontalOverhang, value);
    }

    public double GableOverhang
    {
        get => _gableOverhang;
        set => SetField(ref _gableOverhang, value);
    }

    public double Thickness
    {
        get => _thickness;
        set => SetField(ref _thickness, value);
    }
```

```
    public bool UseManualSize
    {
        get => _useManualSize;
        set => SetField(ref _useManualSize, value);
    }

    public double ManualWidth
    {
        get => _manualWidth;
        set => SetField(ref _manualWidth, value);
    }

    public double ManualLength
    {
        get => _manualLength;
        set => SetField(ref _manualLength, value);
    }

    public GableRoofParameters Clone() => new()
    {
        _variant = _variant,
        _ridgeHeight = _ridgeHeight,
        _ridgeOffset = _ridgeOffset,
        _leftHeight = _leftHeight,
        _rightHeight = _rightHeight,
        _frontalOverhang = _frontalOverhang,
        _gableOverhang = _gableOverhang,
        _thickness = _thickness,
        _useManualSize = _useManualSize,
        _manualWidth = _manualWidth,
        _manualLength = _manualLength
    };
}
```

GableRoofVariant.cs

```
namespace RoofCalculator.Models;

public enum GableRoofVariant
{
    Symmetric,
    Asymmetric,
    DifferentHeight
}

public static class GableRoofVariantExtensions
{
    public static string DisplayName(this GableRoofVariant v) => v switch
    {
        GableRoofVariant.Symmetric       => "Класичний (симетричний)",
        GableRoofVariant.Asymmetric      => "Несиметричний",
```

```
        GableRoofVariant.DifferentHeight => "З'єднання на різній висоті",
        _ => v.ToString()
    };
}
```

MaterialCatalog.cs

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

namespace RoofCalculator.Models;

public sealed class MaterialCatalog
{
    public string Version { get; set; } = "1.0";
    public DateTime UpdatedAt { get; set; } = DateTime.Now;
    public string Source { get; set; } = "Local";
    public List<MaterialItem> Items { get; set; } = new();

    public IEnumerable<MaterialItem> ByCategory(MaterialCategory category)
        => Items.Where(i => i.Category == category);

    public MaterialItem? FindById(string id)
        => Items.FirstOrDefault(i => i.Id == id);

    public MaterialItem? DefaultFor(MaterialCategory category)
        => Items.FirstOrDefault(i => i.Category == category);
}
```

MaterialCategory.cs

```
namespace RoofCalculator.Models;

public enum MaterialCategory
{
    Covering,
    Waterproofing,
    VaporBarrier,
    Insulation,
    Rafters,
    Battens,
    Counterbattens,
    Trim,
    Fasteners,
    DrainageSystem,
    Other
}

public static class MaterialCategoryExtensions
{
```

```
    public static string DisplayName(this MaterialCategory c) => c switch
    {
        MaterialCategory.Covering        => "Покривельне покриття",
        MaterialCategory.Waterproofing   => "Гідроізоляція",
        MaterialCategory.VaporBarrier    => "Пароізоляція",
        MaterialCategory.Insulation      => "Утеплювач",
        MaterialCategory.Rafters         => "Стропильна система",
        MaterialCategory.Battens         => "Обрешітка",
        MaterialCategory.Counterbattens  => "Контробрешітка",
        MaterialCategory.Trim            => "Доборні елементи",
        MaterialCategory.Fasteners       => "Кріплення",
        MaterialCategory.DrainageSystem  => "Водостічна система",
        MaterialCategory.Other           => "Інше",
        _ => c.ToString()
    };

    public static int SortOrder(this MaterialCategory c) => c switch
    {
        MaterialCategory.Rafters         => 1,
        MaterialCategory.Counterbattens  => 2,
        MaterialCategory.Waterproofing   => 3,
        MaterialCategory.Insulation      => 4,
        MaterialCategory.VaporBarrier    => 5,
        MaterialCategory.Battens         => 6,
        MaterialCategory.Covering        => 7,
        MaterialCategory.Trim            => 8,
        MaterialCategory.DrainageSystem  => 9,
        MaterialCategory.Fasteners       => 10,
        MaterialCategory.Other           => 99,
        _ => 50
    };
}
```

MaterialItem.cs

```
namespace RoofCalculator.Models;
public sealed class MaterialItem
{
    public string Id { get; set; } = string.Empty;
    public string Name { get; set; } = string.Empty;
    public string? Manufacturer { get; set; }
    public string? Description { get; set; }
    public MaterialCategory Category { get; set; }
    public MaterialUnit Unit { get; set; }
    public double PricePerUnit { get; set; }
    public string Currency { get; set; } = "UAH";
    public double WastePercent { get; set; }
    public string? SourceUrl { get; set; }

    public override string ToString() => $"{Name} · {PricePerUnit:F2}
{Currency}/{Unit.ShortName()}";
}
```

MaterialUnit.cs

```
namespace RoofCalculator.Models;

public enum MaterialUnit
{
    SquareMeter,
    LinearMeter,
    Piece,
    Kilogram,
    Cube,
    Pack,
    Roll
}

public static class MaterialUnitExtensions
{
    public static string ShortName(this MaterialUnit u) => u switch
    {
        MaterialUnit.SquareMeter => "м²",
        MaterialUnit.LinearMeter => "м.п.",
        MaterialUnit.Piece       => "шт",
        MaterialUnit.Kilogram    => "кг",
        MaterialUnit.Cube        => "м³",
        MaterialUnit.Pack        => "уп",
        MaterialUnit.Roll        => "рул",
        _ => u.ToString()
    };
}
```

Point2D.cs

```
using System;

namespace RoofCalculator.Models;

public readonly record struct Point2D(double X, double Z)
{
    public double DistanceTo(Point2D other)
    {
        var dx = other.X - X;
        var dz = other.Z - Z;
        return Math.Sqrt(dx * dx + dz * dz);
    }

    public Point2D Snap(double step)
    {
        return new Point2D(
            Math.Round(X / step) * step,
            Math.Round(Z / step) * step
        );
    }
```

```
    public static Point2D operator +(Point2D a, Point2D b) => new(a.X + b.X, a.Z + b.Z);
    public static Point2D operator -(Point2D a, Point2D b) => new(a.X - b.X, a.Z - b.Z);
    public static Point2D operator *(Point2D a, double k) => new(a.X * k, a.Z * k);
}
```

RoofKind.cs

```
namespace RoofCalculator.Models;

public enum RoofKind
{
    Gable,
    Hip,
    Mansard,
    HipMansard,
    Pyramid,
    Shed,
    MultiGable,
    MultiGableHip,
    Flat
}

public static class RoofKindExtensions
{
    public static string DisplayName(this RoofKind k) => k switch
    {
        RoofKind.Gable         => "Двосхилий",
        RoofKind.Hip           => "Вальмовий",
        RoofKind.Mansard       => "Мансардний",

        _ => k.ToString()
    };

    public static bool IsAvailable(this RoofKind k) => k == RoofKind.Gable;
}
```

Vector3.cs

```
using System;

namespace RoofCalculator.Models;

public readonly record struct Vector3(double X, double Y, double Z)
{
    public static Vector3 Zero => new(0, 0, 0);

    public double Length => Math.Sqrt(X * X + Y * Y + Z * Z);

    public Vector3 Normalized()
    {
        var len = Length;
```

```
        return len < 1e-9 ? Zero : new Vector3(X / len, Y / len, Z / len);
    }

    public static Vector3 operator +(Vector3 a, Vector3 b) => new(a.X + b.X, a.Y + b.Y, a.Z +
b.Z);
    public static Vector3 operator -(Vector3 a, Vector3 b) => new(a.X - b.X, a.Y - b.Y, a.Z -
b.Z);
    public static Vector3 operator *(Vector3 a, double k) => new(a.X * k, a.Y * k, a.Z * k);

    public static Vector3 Cross(Vector3 a, Vector3 b) => new(
        a.Y * b.Z - a.Z * b.Y,
        a.Z * b.X - a.X * b.Z,
        a.X * b.Y - a.Y * b.X
    );

    public static double Dot(Vector3 a, Vector3 b) => a.X * b.X + a.Y * b.Y + a.Z * b.Z;
}
```

Wall.cs

```
using System;
using RoofCalculator.Helpers;

namespace RoofCalculator.Models;

public sealed class Wall : ObservableObject
{
    private Point2D _start;
    private Point2D _end;
    private double _height = 2.5;
    private double _thickness = 0.3;
    private bool _isSelected;

    public Guid Id { get; } = Guid.NewGuid();

    public Point2D Start
    {
        get => _start;
        set { if (SetField(ref _start, value)) NotifyDerived(); }
    }

    public Point2D End
    {
        get => _end;
        set { if (SetField(ref _end, value)) NotifyDerived(); }
    }

    public double Height
    {
        get => _height;
        set => SetField(ref _height, value);
    }
```

```
    public double Thickness
    {
        get => _thickness;
        set => SetField(ref _thickness, value);
    }

    public bool IsSelected
    {
        get => _isSelected;
        set => SetField(ref _isSelected, value);
    }

    public double Length => _start.DistanceTo(_end);

    public double AngleDegrees
    {
        get
        {
            var dx = _end.X - _start.X;
            var dz = _end.Z - _start.Z;
            return Math.Atan2(dz, dx) * 180.0 / Math.PI;
        }
    }

    public Point2D Midpoint => new((_start.X + _end.X) / 2, (_start.Z + _end.Z) / 2);

    public string DisplayName => $"Стіна {Length:F2} м · {AngleDegrees:F0}°";

    private void NotifyDerived()
    {
        OnPropertyChanged(nameof(Length));
        OnPropertyChanged(nameof(AngleDegrees));
        OnPropertyChanged(nameof(Midpoint));
        OnPropertyChanged(nameof(DisplayName));
    }
}
```

ДОДАТОК Б

Код програмної реалізації сервісів API

BuildCalculatorApi.cs

```
using System;
using System.IO;
using System.Net.Http;
using System.Security.Cryptography;
using System.Text;
using System.Text.Json;
using System.Threading.Tasks;
using System.Web;

namespace RoofCalculator.Services.Api;

public sealed class BuildCalculatorApi
{
    private const string BaseUrl = "https://buildcalculator.io/api/v1";
    private static readonly TimeSpan CacheLifetime = TimeSpan.FromHours(24);

    private readonly HttpClient _http;
    private readonly string _cacheDir;

    public BuildCalculatorApi(HttpClient? http = null)
    {
        _http = http ?? new HttpClient { Timeout = TimeSpan.FromSeconds(10) };

        var appData =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
        _cacheDir = Path.Combine(appData, "RoofCalculator", "api-cache");
        Directory.CreateDirectory(_cacheDir);
    }

    public async Task<SearchResponse?> SearchAsync(string query, string lang = "ru", int top
= 3)
    {
        if (string.IsNullOrWhiteSpace(query) || query.Length < 2)
            return null;

        var cacheKey = MakeCacheKey($"search:{lang}:{top}:{query}");
        var cachedJson = TryReadCache(cacheKey);

        if (cachedJson is not null)
        {
            try { return Deserialize(cachedJson); }
            catch { }
        }

        var url = $"{BaseUrl}/search?q={HttpUtility.UrlEncode(query)}&lang={lang}&top={top}";

        try
        {
```

```
            var json = await _http.GetStringAsync(url);
            WriteCache(cacheKey, json);
            return Deserialize(json);
        }
        catch
        {
            if (cachedJson is not null)
            {
                try { return Deserialize(cachedJson); }
                catch { }
            }
            return null;
        }
    }

    private static SearchResponse? Deserialize(string json)
        => JsonSerializer.Deserialize<SearchResponse>(json,
            new JsonSerializerOptions { PropertyNameCaseInsensitive = true });

    private string CachePath(string key) => Path.Combine(_cacheDir, key + ".json");

    private string? TryReadCache(string key)
    {
        var path = CachePath(key);
        if (!File.Exists(path)) return null;

        var info = new FileInfo(path);
        if (DateTime.Now - info.LastWriteTime > CacheLifetime) return null;

        try { return File.ReadAllText(path); }
        catch { return null; }
    }

    private void WriteCache(string key, string content)
    {
        try { File.WriteAllText(CachePath(key), content); }
        catch { }
    }

    private static string MakeCacheKey(string input)
    {
        var bytes = Encoding.UTF8.GetBytes(input);
        var hash = SHA1.HashData(bytes);
        var sb = new StringBuilder();
        foreach (var b in hash) sb.Append(b.ToString("x2"));
        return sb.ToString();
    }
}
```

NbuExchangeApi.cs

```
using System;
```

```
using System.IO;
using System.Net.Http;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;

namespace RoofCalculator.Services.Api;

public sealed class NbuExchangeApi
{
    private const string Url =
"https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?valcode=EUR&json";
    private static readonly TimeSpan CacheLifetime = TimeSpan.FromHours(6);

    private readonly HttpClient _http;
    private readonly string _cachePath;

    public const double FallbackEurToUah = 44.5;

    public NbuExchangeApi(HttpClient? http = null)
    {
        _http = http ?? new HttpClient { Timeout = TimeSpan.FromSeconds(8) };

        var appData =
Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData);
        var dir = Path.Combine(appData, "RoofCalculator", "api-cache");
        Directory.CreateDirectory(dir);
        _cachePath = Path.Combine(dir, "nbu-eur.json");
    }

    public async Task<double> GetEurToUahAsync()
    {
        var cached = TryReadCache();
        if (cached is not null && cached.Value > 0) return cached.Value;

        try
        {
            var json = await _http.GetStringAsync(Url);
            var arr = JsonSerializer.Deserialize<NbuRate[]>(json);
            if (arr is { Length: > 0 } && arr[0].Rate > 0)
            {
                WriteCache(arr[0].Rate);
                return arr[0].Rate;
            }
        }
        catch { }

        return cached ?? FallbackEurToUah;
    }

    private double? TryReadCache()
    {
        if (!File.Exists(_cachePath)) return null;
        var info = new FileInfo(_cachePath);
```

```
        if (DateTime.Now - info.LastWriteTime > CacheLifetime) return null;

        try
        {
            var text = File.ReadAllText(_cachePath);
            return double.Parse(text, System.Globalization.CultureInfo.InvariantCulture);
        }
        catch { return null; }
    }

    private void WriteCache(double rate)
    {
        try { File.WriteAllText(_cachePath,
rate.ToString(System.Globalization.CultureInfo.InvariantCulture)); }
        catch { }
    }

    private sealed class NbuRate
    {
        [JsonPropertyName("rate")] public double Rate { get; set; }
        [JsonPropertyName("cc")] public string Cc { get; set; } = "EUR";
    }
}
```

SearchResponse.cs

```
using System.Collections.Generic;
using System.Text.Json.Serialization;

namespace RoofCalculator.Services.Api;

public sealed class SearchResponse
{
    [JsonPropertyName("query")] public string Query { get; set; } = string.Empty;
    [JsonPropertyName("language")] public string Language { get; set; } = "en";
    [JsonPropertyName("results_count")] public int ResultsCount { get; set; }
    [JsonPropertyName("results")] public List<WorkItem> Results { get; set; } = new();
}

public sealed class WorkItem
{
    [JsonPropertyName("rate_code")] public string RateCode { get; set; } = string.Empty;
    [JsonPropertyName("name")] public string Name { get; set; } = string.Empty;
    [JsonPropertyName("unit")] public string Unit { get; set; } = "m2";
    [JsonPropertyName("currency")] public string Currency { get; set; } = "EUR";
    [JsonPropertyName("pricing")] public PricingInfo Pricing { get; set; } = new();
    [JsonPropertyName("cost_breakdown")] public CostBreakdown? CostBreakdown { get; set; }
    [JsonPropertyName("classification")] public Classification? Classification { get; set; }
}

public sealed class PricingInfo
{
```

```
    [JsonPropertyName("total_per_unit")] public double TotalPerUnit { get; set; }
    [JsonPropertyName("labor_per_unit")] public double LaborPerUnit { get; set; }
    [JsonPropertyName("material_per_unit")] public double MaterialPerUnit { get; set; }
    [JsonPropertyName("equipment_per_unit")] public double EquipmentPerUnit { get; set; }
}

public sealed class CostBreakdown
{
    [JsonPropertyName("labor_pct")] public double LaborPct { get; set; }
    [JsonPropertyName("material_pct")] public double MaterialPct { get; set; }
    [JsonPropertyName("equipment_pct")] public double EquipmentPct { get; set; }
}

public sealed class Classification
{
    [JsonPropertyName("category")] public string? Category { get; set; }
}
```

ДОДАТОК В

Код програмної реалізації сервісів

CanvasTransform.cs

```
using System.Windows;
using RoofCalculator.Models;

namespace RoofCalculator.Services;

public sealed class CanvasTransform
{
    public double PixelsPerMeter { get; set; } = 40.0;
    public double OriginX { get; set; } = 0;
    public double OriginZ { get; set; } = 0;
    public double GridStepMeters { get; set; } = 0.5;

    public Point WorldToScreen(Point2D world)
    {
        return new Point(
            OriginX + world.X * PixelsPerMeter,
            OriginZ + world.Z * PixelsPerMeter
        );
    }

    public Point2D ScreenToWorld(Point screen)
    {
        return new Point2D(
            (screen.X - OriginX) / PixelsPerMeter,
            (screen.Y - OriginZ) / PixelsPerMeter
        );
    }

    public Point2D SnapToGrid(Point2D world) => world.Snap(GridStepMeters);

    public double PixelsPerGridCell => GridStepMeters * PixelsPerMeter;
}
```

GeometryService.cs

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

namespace RoofCalculator.Services;

public static class GeometryService
{
    public const double SnapTolerance = 0.05;

    public static double DistanceFromPointToSegment(Point2D point, Point2D segStart, Point2D
```

```
segEnd)
    {
        var dx = segEnd.X - segStart.X;
        var dz = segEnd.Z - segStart.Z;
        var lengthSq = dx * dx + dz * dz;
        if (lengthSq < 1e-9) return point.DistanceTo(segStart);

        var t = ((point.X - segStart.X) * dx + (point.Z - segStart.Z) * dz) / lengthSq;
        t = Math.Clamp(t, 0.0, 1.0);

        var projX = segStart.X + t * dx;
        var projZ = segStart.Z + t * dz;
        return point.DistanceTo(new Point2D(projX, projZ));
    }

    public static bool IsClosedContour(IList<Wall> walls, double tolerance = SnapTolerance)
    {
        if (walls.Count < 3) return false;
        var first = walls[0].Start;
        var last = walls[^1].End;
        return last.DistanceTo(first) < tolerance;
    }

    public static IReadOnlyList<Point2D> GetContourPoints(IList<Wall> walls)
    {
        if (walls.Count == 0) return Array.Empty<Point2D>();
        var result = new List<Point2D> { walls[0].Start };
        foreach (var wall in walls) result.Add(wall.End);
        if (result.Count > 1 && result[0].DistanceTo(result[^1]) < SnapTolerance)
            result.RemoveAt(result.Count - 1);
        return result;
    }

    public static double PolygonArea(IList<Point2D> points)
    {
        if (points.Count < 3) return 0;
        double sum = 0;
        for (int i = 0; i < points.Count; i++)
        {
            var current = points[i];
            var next = points[(i + 1) % points.Count];
            sum += (current.X * next.Z) - (next.X * current.Z);
        }
        return Math.Abs(sum) / 2.0;
    }

    public static (double Width, double Length, double MinX, double MaxX, double MinZ, double
MaxZ) GetBoundingBox(IList<Wall> walls)
    {
        if (walls.Count == 0) return (0, 0, 0, 0, 0, 0);

        var allPoints = walls.SelectMany(w => new[] { w.Start, w.End }).ToList();
        var minX = allPoints.Min(p => p.X);
        var maxX = allPoints.Max(p => p.X);
        var minZ = allPoints.Min(p => p.Z);
```

```
        var maxZ = allPoints.Max(p => p.Z);

        return (maxX - minX, maxZ - minZ, minX, maxX, minZ, maxZ);
    }

    public static double Perimeter(IList<Wall> walls)
    {
        double total = 0;
        foreach (var wall in walls) total += wall.Length;
        return total;
    }
}
```