МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

Волков Артем Анатолійович
(прізвище, ім'я, по батькові здобувача)

на тему
«Розробка програмного забезпечення проектування інтер'єрів»
(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД
(повна назва бази дослідження)

науковий керівник к.т.н., доцент _____ Медведєв Д. Г.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 р. № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«11» червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ[1]

1. Тема роботи «Розробка програмного забезпечення проектування інтер'єрів»

Керівник роботи к.т.н., доцент Медведєв Д. Г
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об'єкт, предмет та мета дослідження:

Розділ 1. Аналіз предметної області та постановка задачі

Розділ 2. Проектування архітектури системи

Розділ 3. Розробка бази даних та інформаційного забезпечення

Розділ 4.Реалізація веб- і десктопного застосунку

Об'єкт дослідження: Системи тривимірного моделювання інтер'єрів

Предмет дослідження: Архітектура та алгоритми побудови онлайн-конструкторів інтер'єрів з 3D-візуалізацією

Мета кваліфікаційної роботи: *Розробка програмного комплексу для створення та перегляду 3D-моделей інтер'єрів, який включає веб-застосунок і десктоп 3D-переглядач*

5. Дата видачі завдання «04» квітня 2025 р.

[1] Назва кваліфікаційної роботи відповідно до ОПП

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ Медведєв Д. Г.
(підпис) (прізвище та ініціали)

Завдання одержав _____ Волков А. А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на кваліфікаційну бакалаврську роботу

«Розробка програмного забезпечення проектування інтер’єрів»

Волкова Артема Анатолійовича

У бакалаврській кваліфікаційній роботі розроблено програмний комплекс «Онлайн-конструктор інтер'єрів», що включає веб-застосунок та десктопний застосунок для проектування приміщень з можливістю візуалізації інтер’єру у 3D. Реалізовано інтерфейс для створення кімнат, додавання меблів, зміни параметрів інтер'єру (колір стін, підлоги, освітлення) та збереження проектів у базі даних. Десктопний застосунок дозволяє завантажувати створені інтер’єри та відображати їх за допомогою технології OpenGL у режимі реального часу.

Програмне забезпечення зроблено використовуючи мову програмування C# у середовищі .NET WPF. Для графічного рендерингу використовується OpenGL. Веб-частина реалізована з використанням React (фронтенд) і Node.js + Express (бекенд). Для збереження інформації про користувачів і проекти застосовується база даних MongoDB. 3D-моделі меблів зберігаються у файловому сховищі.

Ключові слова: 3D-ВІЗУАЛІЗАЦІЯ, ІНТЕР’ЄР, ВЕБ-ЗАСТОСУНОК, ДЕСКТОПНИЙ ЗАСТОСУНОК, OpenGL, ПРОЄКТУВАННЯ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, REACT, NODE.JS, MongoDB.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД (база даних)	Організовані дані, які мають логічні зв’язки між собою, що зберігаються та обробляються для забезпечення роботи системи і забезпечення користувачів потрібними даними.
СУБД	Система управління базами даних; програмне забезпечення для створення, редагування, збереження та організації доступу до бази даних (у цій роботі використовується MongoDB).
ПЗ	Програмне забезпечення; сукупність програм, процедур та документації, що забезпечують функціонування комп’ютерної системи.
WPF (.NET)	Windows Presentation Foundation; графічна підсистема для побудови інтерфейсів користувача у C#-додатках на платформі .NET.
OpenGL	Open Graphics Library; кросплатформенний API для рендерингу 2D- та 3D-графіки у реальному часі.
API	Application Programming Interface — інтерфейс прикладного програмування, що забезпечує взаємодію між компонентами системи (наприклад, між веб-застосунком і базою даних).
React	JavaScript-бібліотека для побудови користувацьких інтерфейсів.
Node.js	Програмна платформа, що дозволяє запускати JavaScript на сервері.

ЗМІСТ

ВСТУП

Активний розвиток цифрових технологій та їх широка доступність персональних комп’ютерів і мобільних пристроїв суттєво вплинули на всі сфери людської діяльності, зокрема на галузь архітектури, будівництва та дизайну інтер’єрів. Проектування приміщень, яке донедавна було складним та дороговартісним процесом, сьогодні все частіше здійснюється за допомогою спеціалізованих програмних засобів, що забезпечують зручний інтерфейс, функціональну гнучкість і високий рівень візуалізації. Водночас зростає потреба у програмному забезпеченні, яке було б не тільки інструментом для професіоналів, а й доступним рішенням для широкого кола користувачів, включаючи аматорів і студентів.

Особливого значення набувають програмні комплекси, що поєднують у собі переваги вебтехнологій та десктопних застосунків. Такі гібридні рішення дозволяють розробляти, зберігати, поширювати та переглядати 3D-моделі інтер’єрів з будь-якого пристрою. Вони також забезпечують можливість збереження проектів у базі даних, подальшої модифікації, обміну між користувачами тощо.

Кваліфікаційна робота присвячена розробці програмного комплексу «Онлайн-конструктор інтер’єрів», який складається з двох основних частин: веб-застосунку для створення й редагування інтер’єрів та десктопного застосунку для їх перегляду в 3D-режимі. У межах цієї системи реалізується доступ до бази даних, побудова інтерфейсів користувача, графічне відображення тривимірних об’єктів, а також інтеграція з серверною логікою на основі сучасних технологій.

Розробка даного програмного комплексу вимагає ґрунтовного підходу до аналізу предметної області, вивчення наявних рішень, вибору архітектури системи та інструментів реалізації. У роботі детально розглянуто основні етапи створення програмного забезпечення: від постановки задачі та формування вимог до функціонування системи — до реалізації інтерфейсів,

опрацювання структури бази даних, написання коду й тестування готового продукту.

Актуальність теми обумовлена потребою ринку у зручних, гнучких і водночас простих засобах візуалізації інтер’єрів, які не потребують глибоких знань у сфері програмування чи тривимірного моделювання. Запропоноване рішення дозволяє задовольнити запити як побутових користувачів, так і професіоналів, завдяки зручному інтерфейсу, простому обміну проектами через хмарну базу даних та можливості перегляду готового інтер’єру в режимі справжнього часу.

Отже, дана кваліфікаційна робота має на меті розробити ефективний, доступний і технологічно актуальний інструмент для створення й перегляду інтер’єрів, що враховує сучасні вимоги до зручності, продуктивності та взаємодії з базою даних.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Сфера застосування програмного комплексу

Сьогодні, коли технології мають вирішальне значення для щоденного життя людини, архітектурне проектування та інтер’єрний дизайн стають усе більш тісно пов’язаними з цифровими інструментами. Сфера інтер’єрного дизайну давно вийшла за межі виключно творчого процесу: сьогодні вона вимагає високої точності, ефективної комунікації між виконавцем та замовником, а також зручних засобів для візуалізації і внесення змін до проекту у реальному часі.

У такому контексті програмні комплекси, що забезпечують створення, збереження та перегляд інтер’єрів, відіграють особливо важливу роль. Вони стають невід’ємним інструментом як для професійних дизайнерів, архітекторів, так і для звичайних користувачів, які прагнуть самостійно проектувати простір свого помешкання або комерційного об'єкта. Ці рішення дозволяють моделювати простір, експериментувати з меблями, кольорами, текстурами та світлом, отримуючи при цьому наочну візуалізацію майбутнього інтер’єру.

Особливої актуальності набувають веб- та десктопні застосунки, які об’єднують зручність доступу, кросплатформеність, підтримку хмарних технологій і 3D-візуалізації. У сфері житлового та комерційного будівництва такі інструменти використовуються для:

- попереднього узгодження макетів приміщень із клієнтом;
- роботи з каталогами меблів та матеріалів;
- створення інтерактивних презентацій для замовника;
- збереження та демонстрації віртуального простору для подальшої реалізації проекту.

Застосування таких програмних комплексів є доцільним у діяльності:

- дизайнерських студій;
- архітектурних бюро;
- будівельних компаній;
- рієлторських агенцій;
- компаній, що займаються виробництвом або продажем меблів;
- навчальних закладів (для підготовки майбутніх спеціалістів у галузі дизайну).

Такі інструменти можуть бути корисними не лише професіоналам, а й звичайним користувачам, які бажають самостійно спроектувати інтер’єр власного помешкання без залучення дизайнерів. Наприклад, користувач може змоделювати свою кімнату, вказавши необхідні розміри, обрати меблі з наявної бібліотеки або завантажити власні 3D-моделі, після чого зберегти готовий проект у базі даних для подальшого перегляду в десктопному застосунку.

Крім суто утилітарних функцій, програмні комплекси такого типу можуть відігравати й роль маркетингових інструментів. Так, компанії-виробники меблів або декору можуть вбудовувати свою продукцію в базу об'єктів конструктора, надаючи потенційним клієнтам можливість побачити, як та чи інша модель виглядатиме у реальному просторі.

Таким чином, сфера застосування розроблюваного програмного комплексу охоплює широкий спектр задач: від проектування інтер’єру до візуального маркетингу, від індивідуального використання до корпоративного, від освітніх завдань до комерційних презентацій. Це робить програмний продукт універсальним інструментом, що має високий попит на ринку цифрових рішень для просторового планування.

1.2. Аналіз існуючих рішень

Із розвитком ІТ-сфери з’явилося багато програм, що дозволяють можливість створення, візуалізації та редагування інтер’єрів. До найпоширеніших рішень у цій галузі належать як професійні системи, орієнтовані на дизайнерів та архітекторів, так і простіші інструменти, призначені для масового користувача. У цьому підрозділі буде розглянуто найвідоміші представники кожної з цих категорій.

Одним із найпотужніших рішень є Autodesk 3ds Max — це професійний програмний інструмент для створення 3D-моделей, анімації та візуалізації. Програма підтримує складні сцени, має широкі можливості для налаштування матеріалів, освітлення та рендерингу. Однак через складність інтерфейсу, високу вартість і потребу в потужному обладнанні вона не є зручною для масового користувача.

Ще одним прикладом професійного ПЗ є SketchUp Pro, що дозволяє швидко створювати тривимірні моделі інтер’єрів і будівель. Програма має велику базу 3D-об’єктів, простий у засвоєнні інтерфейс і можливість підключення сторонніх плагінів. Проте повна функціональність доступна лише у платній версії.

Для широкого кола користувачів популярними є онлайн-інструменти на кшталт Planner 5D, Roomstyler 3D Home Planner та HomeByMe. Ці сервіси мають інтуїтивно зрозумілий інтерфейс, дозволяють обирати меблі з готової бібліотеки, створювати планування приміщень та переглядати результати у 2D і 3D режимах. Основними перевагами є простота використання та доступність через браузер без потреби встановлення.

Проте ці сервіси мають низку обмежень. Частина функцій зазвичай доступна лише після реєстрації або придбання преміум-доступу. Крім того, можливості збереження проектів, експорту моделей у сторонні формати та подальшого рендерингу обмежені. Також спостерігається недостатня

інтеграція з локальними десктопними рішеннями, що ускладнює подальше використання створених проектів у професійному середовищі.

Таким чином, існуючі рішення або мають занадто складний функціонал для пересічного користувача, або, навпаки, є надто обмеженими для виконання складних дизайнерських завдань. Це створює потребу в розробці програмного комплексу, який би поєднував зручність використання, доступність, функціональність і підтримку як веб-, так і десктопної взаємодії.

1.3. Основні вимоги до функціональності

Під час розробки програмного комплексу «Онлайн-конструктор інтер’єрів» важливо чітко визначити основні функціональні вимоги, які забезпечать його ефективну роботу, зручність використання для кінцевого користувача, а також дозволять гнучко масштабувати систему в майбутньому. Ці вимоги формуються на основі аналізу предметної області, існуючих аналогів, а також технічного завдання, сформованого на початковому етапі проекту.

До функціональних вимог відносяться ті можливості та дії, які система має забезпечувати у процесі своєї роботи. Їх реалізація дозволить досягти головної мети — створення зручного й доступного інструменту для створення, перегляду та збереження 3D-інтер’єрів у веб- та десктоп-середовищі.

Основні функціональні вимоги до програмного комплексу включають наступні підсистеми:

- Реєстрація та авторизація користувачів: Користувачі можуть створювати облікові записи, авторизуватись, редагувати свої дані, а також здійснювати вхід через сторонні сервіси (Google, Facebook — у перспективі).
- Керування проектами: Кожен зареєстрований користувач повинен мати змогу створювати нові проекти, переглядати список

збережених, редагувати та видаляти їх. Для зручності мають бути передбачені фільтрація й сортування.

- Редактор інтер’єру (веб-застосунок): Цей модуль повинен надавати інтерфейс drag-and-drop для розміщення меблів та елементів інтер’єру в кімнаті, зміну розмірів приміщення, вибір кольору стін, підлоги, встановлення дверей, вікон тощо.
- Збереження 3D-проєкту в базі даних: Кожен проєкт має зберігатися у вигляді опису приміщення, меблів, їх розташування, орієнтації, а також текстур і додаткових параметрів. Збереження виконується через API-сервер.
- Перегляд збережених сцен у десктопному застосунку: У десктопному переглядачі користувач може завантажити збережений проект із серверу, відобразити його у 3D-сцені з можливістю навігації (огляд з першої або третьої особи), але без редагування.
- Обробка 3D-графіки: Програма повинна підтримувати завантаження 3D-моделей меблів у популярному форматі (наприклад, .obj, .glb), їх візуалізацію, масштабування та обертання.
- Підтримка текстур і освітлення: Для забезпечення реалістичності сцени повинна бути реалізована підтримка матеріалів і текстур, а також елементарна система освітлення (ambient, directional light).
- Відповідність вимогам безпеки та цілісності даних: Усі дії користувача мають бути захищеними, з використанням сучасних методів шифрування (JWT, HTTPS). Сервер має фіксувати всі звернення до API та контролювати права доступу.
- Кросплатформеність: Веб-застосунок повинен бути доступним на різних пристроях і браузерах, а десктопна версія має коректно функціонувати на Windows 10/11. У перспективі — адаптація під macOS.

- Можливість розширення функціоналу: Архітектура має передбачати гнучке додавання нових функцій (наприклад, модуль освітлення, збереження знімків сцени, генерація PDF-звітів із плануванням приміщення тощо)(див. табл. 1.1).

Таблиця 1.1

Характеристика функціональних модулів програмного забезпечення

№	Назва модуля	Опис функціоналу
1	Реєстрація та авторизація	Створення аккаунту та вхід у систему
2	Управління проектами	Створення, перегляд, редагування, видалення проектів
3	Веб-редактор інтер’єру	Розміщення об’єктів, налаштування приміщення
4	Збереження проекту	Збереження конфігурації сцени у базі даних
5	Переглядач проектів (десктоп)	Візуалізація сцени у 3D, навігація
6	Підтримка 3D-моделей	Завантаження, обробка, відображення моделей
7	Текстури та освітлення	Реалістичність відображення сцени
8	Захист і безпека	Авторизація, контроль доступу, шифрування
9	Кросплатформеність	Доступність на різних ОС та пристроях
10	Розширюваність	Система дозволяє розширювати функціональність без внесення змін до її ядра.

1.4. Обґрунтування вибору технологій

Однією з ключових передумов успішної реалізації програмного комплексу «Онлайн-конструктор інтер’єрів» є правильний вибір технологій. Сучасна індустрія розробки програмного забезпечення пропонує широкий спектр інструментів, мов програмування, бібліотек і фреймворків, кожен із яких має свої сильні й слабкі сторони. При цьому важливо забезпечити баланс між продуктивністю, масштабованістю, простотою супроводу, зручністю для розробників, а також відповідністю технічному завданню.

Проект передбачає реалізацію веб-застосунку, десктопного застосунку, серверної частини та бази даних, що вимагає міжплатформеної інтеграції, надійної взаємодії між компонентами й ефективної обробки 3D-графіки.

Веб-застосунок (редактор інтер'єру):

Для розробки веб-редактора було обрано React — сучасну JavaScript-бібліотеку з відкритим кодом, створену компанією Meta. Основними перевагами React є компонентна структура, віртуальний DOM для швидкого оновлення інтерфейсу та широка екосистема додаткових бібліотек. Він дозволяє створити інтерактивний drag-and-drop інтерфейс, що реагує в режимі справжнього часу на дії користувача.

Для роботи з 3D-графікою в браузері планується використання бібліотеки Three.js. Вона дозволяє виводити складні 3D-сцени, імпортувати моделі у форматах glTF, OBJ, FBX тощо, а також забезпечує візуалізацію з текстурами, освітленням і тінями.

У поєднанні React і Three.js дають можливість реалізувати зручний редактор інтер'єру, що працює у звичайному браузері без потреби встановлення стороннього програмного забезпечення.

Десктопний застосунок (переглядач):

Для створення десктопного застосунку обрано платформу Windows Presentation Foundation (WPF) з бібліотекою OpenTK.

WPF дозволяє створювати багаті інтерфейси з підтримкою XAML-розмітки, а OpenTK (Open Toolkit Library) — це кросплатформенний .NET-обгортка для OpenGL, що надає можливості для високопродуктивної 3D-графіки. Такий тандем дозволяє завантажувати збережені проекти з бази даних, виводити їх у тривимірній формі та реалізувати навігацію всередині сцени.

Переваги такого підходу:

- Висока продуктивність рендерингу сцени за рахунок апаратного прискорення (GPU).

- Можливість точного контролю над графічним контентом (позиція, освітлення, масштаб тощо).
- Зручна розробка у середовищі .NET із доступом до потужної екосистеми Microsoft.

Серверна частина:

У якості серверної платформи було обрано Node.js. Це серверне середовище виконання JavaScript-коду, яке ідеально підходить для обробки великої кількості одночасних з’єднань завдяки неблокуючій моделі введення/виведення.

Фреймворк Express.js, що працює поверх Node.js, дозволяє швидко створювати REST API, які будуть використовуватись для обміну даними між веб-застосунком, десктопною програмою та базою даних.

Node.js з Express.js забезпечують:

- Високу масштабованість і продуктивність;
- Швидку розробку серверної логіки;
- Зручність при розгортанні на хмарних сервісах.

База даних:

Для збереження даних про користувачів, проекти та 3D-сцени було обрано MongoDB — документоорієнтовану базу даних NoSQL. Вона ідеально підходить для збереження неструктурованих або напівструктурованих даних, наприклад, JSON-об’єктів, які описують конфігурацію інтер’єру.

MongoDB має наступні переваги:

- MongoDB має наступні переваги:
- Гнучкість у структурі даних;
- Висока швидкість читання та запису;
- Можливість масштабування в майбутньому;
- Широка підтримка на стороні Node.js (див. Табл. 1.2).

Таблиця 1.2

Обранні технології для реалізації компонентів програмного забезпечення

Компонент	**Обрана технологія**	**Причина вибору**
Веб-інтерфейс	React + Three.js	Компонентність, швидкість, гнучкість, 3D-візуалізація
Десктопна програма	WPF + OpenTK	Апаратне прискорення, точна візуалізація, зручна інтеграція в .NET-середовище
Серверна частина	Node.js + Express.js	Висока продуктивність, простота створення REST API
База даних	MongoDB	Гнучка структура, легке масштабування, інтеграція з JavaScript

Висновки до розділу 1

У першому розділі розглянуто ключові аспекти предметної області в межах якого визначено актуальність створення програмного комплексу для проектування інтер’єрів у 3D-середовищі. Сфера застосування охоплює широкий спектр користувачів — від дизайнерів-початківців до професійних студій, які потребують зручного інструменту для візуалізації простору.

Проведено огляд наявних рішень на ринку, зокрема таких як Planner 5D, Roomstyler та Homestyler, що дозволило виділити ключові функції та недоліки аналогів. Це, у свою чергу, стало основою для формування власних функціональних вимог до системи.

Було чітко сформульовано перелік основних підсистем, які мають бути реалізовані: модулі реєстрації нових користувачів, а також можливості створення та редагування інтер’єрів у веб-браузері, збереження 3D-проєктів на сервері та їх подальший перегляд у десктопному переглядачі. Також визначено вимоги до графічної підсистеми, підтримки текстур, безпеки даних і кросплатформенності.

Особливу увагу приділено обґрунтуванню вибору технологій, що використовуватимуться в процесі розробки, на різних її стадіях: React і Three.js

для веб-редактора, WPF з OpenTK для десктопного перегляду, Node.js з Express для серверної частини та MongoDB для збереження структурованих даних. Обраний стек технологій дозволяє створити гнучкий, масштабований і зручний у розробці програмний комплекс, здатний забезпечити всі поставлені функціональні цілі.

Таким чином, завершення першого розділу створює міцну теоретичну й методологічну основу для подальшого проектування архітектури системи, що буде розглянуто в наступному розділі.

РОЗДІЛ 2
ПРОЕКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

2.1. Загальна структура програмного комплексу

Розроблений програмний комплекс «Онлайн-конструктор інтер’єрів» складається з кількох логічно пов’язаних компонентів, кожна з яких виконує окрему роль у процесі взаємодії користувача з системою. Архітектура комплексу побудована за принципами розділення обов’язків та орієнтована на масштабованість і модульність.

Загальна структура включає наступні основні частини:

- Веб-застосунок (фронтенд), що реалізований з використанням бібліотеки React. Він забезпечує інтерфейс для створення та редагування 3D-сцен, керування обліковим записом і проектами користувача. Працює у веб-браузері та спілкується з серверною частиною через HTTP-запити до API.
- Десктопний переглядач, створений із використанням технологій WPF та OpenTK. Він дозволяє користувачу отримувати з бази даних завантажуванні проекти і переглядати їх у режимі 3D-візуалізації на ПК з операційною системою Windows.
- Сервер реалізовано на платформі Node.js із застосуванням фреймворку Express, що забезпечує обробку запитів, авторизацію, збереження даних, обмін інформацією між клієнтськими застосунками та базою даних.
- База даних, реалізована у вигляді документо-орієнтованої СУБД MongoDB, зберігає інформацію про користувачів, проекти, 3D-моделі, їх властивості та параметри сцен.

Загальну схему архітектури системи наведено на рисунку 2.1.

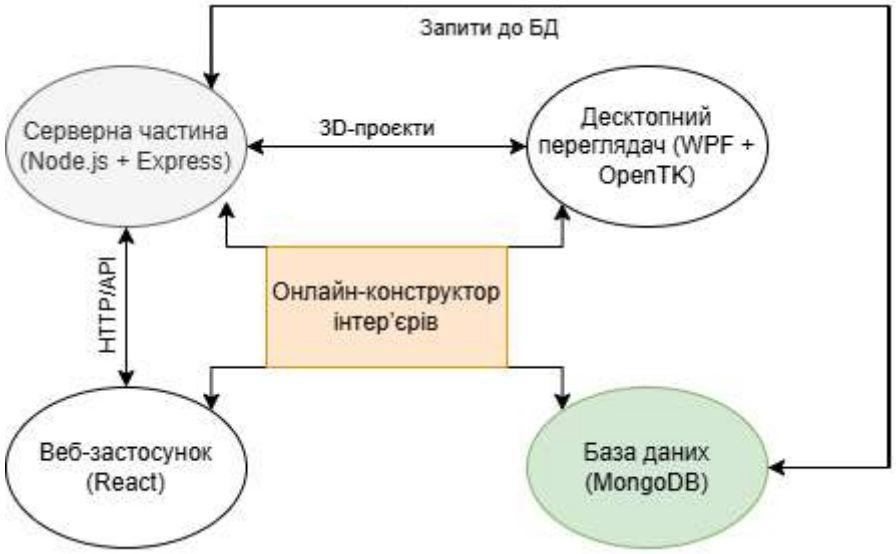

Рис. 2.1. Логічна структура розробленої системи «Онлайн-конструктор інтер'єрів»

2.2. Взаємодія між компонентами(веб,десктоп,сервер)

Організація взаємодії між складовими програмного комплексу «Онлайн-конструктор інтер'єрів» ґрунтується на клієнт-серверній архітектурі з використанням REST API як основного каналу комунікації. Такий підхід забезпечує гнучкість, масштабованість і можливість паралельного розвитку різних частин системи.

Веб-застосунок:

Веб-клієнт, реалізований на основі бібліотеки React, взаємодіє із сервером через HTTP-запити. Через REST API він отримує та передає дані: інформацію про користувача, структуру проекту, 3D-моделі, матеріали тощо. Запити передаються у форматі JSON, а авторизація здійснюється за допомогою токенів (JWT).

Десктопний переглядач:

Десктопний застосунок, написаний на WPF з використанням OpenTK, також комунікує із сервером через API. Його функція полягає у завантаженні збережених сцен із бази даних, їх локальній обробці та візуалізації в

інтерактивному 3D-інтерфейсі. При цьому редагування інтер’єру у десктопній версії не передбачено — лише перегляд.

Сервер:

Сервер, побудований на Node.js, виконує функції посередника між клієнтськими додатками та базою даних. Він:

- автентифікує користувачів,
- обробляє запити CRUD (створення, читання, оновлення, видалення),
- контролює доступ до ресурсів,
- зберігає та видає дані через API.

База даних:

Усі дані, пов’язані з користувачами, проектами, сценами та 3D-моделями, зберігаються в MongoDB — документо-орієнтованій СУБД. Такий формат дає змогу гнучко зберігати складні об’єкти (включно з вкладеними параметрами моделей) без жорсткої схеми.

Загальна логіка взаємодії виглядає наступним чином:

Веб- або десктопний клієнт → надсилає запити до API → сервер обробляє запит → звертається до бази даних → повертає клієнту відповідь.

Взаємний зв’язок між компонентами системи представлено на рисунку 2.2.

Рис. 2.2. Взаємодія між основними компонентами програмного комплексу

2.3. Архітектурні шаблони та принципи

Під час розробки програмного комплексу «Онлайн-конструктор інтер’єрів» було використано низку сучасних архітектурних шаблонів і принципів проектування, які забезпечують гнучкість, масштабованість і підтримуваність системи.

Архітектура клієнт-сервер

Основною структурною моделлю виступає архітектура клієнт-сервер, що реалізує архітектурний підхід із поділом на клієнтську та серверну частини. Такий підхід дозволяє централізовано обробляти дані на сервері, тоді як клієнти відповідають за інтерфейс роботи з користувачем.

REST API

Для комунікації між клієнтами (веб та десктоп) і сервером застосовується REST (Representational State Transfer). Цей шаблон передбачає використання HTTP-запитів для отримання та зміни даних. REST є легким у реалізації, добре підтримується більшістю мов і фреймворків, а також забезпечує відокремлення клієнта від внутрішньої логіки сервера.

Модель MVC (Model-View-Controller)

У структурі веб-застосунку (React) використовується принцип поділу на компоненти, що узгоджується з ідеєю MVC, де:

- Model — стан інтер’єру (розміри кімнати, об’єкти),
- View — React-компоненти для відображення сцен і елементів UI,
- Controller — функції керування, які реагують на події й оновлюють стан.

У десктопному застосунку (WPF) застосовується патерн MVVM (Model-View-ViewModel), який є стандартом для WPF-додатків:

- Model — об’єкти даних про інтер’єр та 3D-моделі,
- View — XAML-інтерфейс програми,
- ViewModel — логіка зв’язку між Model і View.

Принципи SOLID

При проєктуванні коду на стороні сервера і клієнтів дотримано принципів SOLID, зокрема:

- S (Single Responsibility) — кожен клас або модуль має одну відповідальність;
- (Open/Closed) — система легко розширюється новим функціоналом без зміни існуючого коду;
- L (Liskov Substitution) — об’єкти підкласів можуть заміняти об’єкти базових класів;
- I (Interface Segregation) — інтерфейси не містять зайвих методів;
- D (Dependency Inversion) — залежності вводяться через інтерфейси або DI-контейнери.

Розділення відповідальності (Separation of Concerns)

Важливим принципом при реалізації API та клієнтів є розділення відповідальності. Кожен модуль (реєстрація, збереження проектів, рендеринг) реалізується незалежно, що полегшує їх тестування та подальшу підтримку.

2.4. Алгоритми обробки 3d-сцен

Обробка тривимірних сцен є ключовим компонентом програмного комплексу «Онлайн-конструктор інтер’єрів», оскільки саме від якості та ефективності цієї частини залежить реалістичність візуалізації та зручність взаємодії користувача з інтерфейсом.

Алгоритми обробки 3D-сцен реалізуються як у веб-застосунку під час побудови та редагування інтер’єру, так і в десктопному застосунку при перегляді збережених проектів. Основними етапами обробки сцени є:

Завантаження 3D-моделей

Всі моделі меблів, елементів інтер’єру та приміщення зберігаються у форматах .obj, .glb або .gltf, які забезпечують збереження геометрії, матеріалів, текстур та анімацій. Для їх імпорту використовуються спеціалізовані

бібліотеки: у веб-застосунку — three.js, у десктопному — OpenTK з кастомним парсером.

Парсинг і побудова сцени

Після завантаження модель аналізується: формується сітка (mesh), текстури, нормалі, координати освітлення. Потім об’єкти додаються до сцени з урахуванням позиції, масштабу та орієнтації. Для цього використовується матрична трансформація (трансляція, масштабування, обертання).

Камера та навігація

У 3D-просторі реалізована віртуальна камера, яка дозволяє змінювати ракурс перегляду. Веб-інтерфейс підтримує drag-керування, zoom і обертання, а десктопна версія — режим першої особи з керуванням через мишу та клавіатуру.

Освітлення

Для досягнення базового реалізму застосовується комбіноване освітлення:

- ambient light — загальне розсіяне світло,
- directional light — імітація сонячного світла,
- (у перспективі) point light для локальних джерел світла (лампи, світильники).

Рендеринг сцени

У веб-застосунку використовується WebGL через three.js, що забезпечує апаратне прискорення графіки у браузері. У десктопному застосунку рендеринг виконується через OpenGL, з використанням OpenTK. Рендеринг включає обчислення положень вершин, нормалей, шейдинг та обробку матеріалів.

Оптимізація продуктивності

Для підвищення швидкодії застосовуються такі методи:

- culling — виключення з рендерингу об’єктів, які не потрапляють у поле зору;
- level of detail (LOD) — спрощення геометрії віддалених об’єктів;

- кешування текстур і моделей у локальній пам’яті;
- асинхронне завантаження великих моделей.

Збереження стану сцени

При збереженні проекту усі об’єкти сцени перетворюються у JSON-формат, що містить координати, ідентифікатори моделей, текстур і параметри, необхідні для реконструкції сцени при перегляді.

Висновки до розділу 2

У цьому розділі було здійснено проектування архітектури програмного комплексу «Онлайн-конструктор інтер’єрів», що охоплює як веб-, так і десктопну складову. Проведений аналіз дозволив визначити загальну архітектуру системи, сформовано логіку взаємодії між клієнтськими застосунками, сервером і базою даних.

Було обґрунтовано вибір архітектурних шаблонів, зокрема клієнт-серверної моделі з використанням REST API для забезпечення масштабованості та модульності. Крім того, розглянуто ключові принципи, яких слід дотримуватися під час побудови програмної системи, включаючи принципи SOLID, DRY та розділення відповідальностей.

Окрему увагу приділено алгоритмам обробки 3D-сцен — як у частині побудови та візуалізації інтер’єру, так і збереження та завантаження проектів. Було описано основні процеси, зокрема імпорт моделей, рендеринг, управління камерою, роботу з текстурами й освітленням, а також оптимізацію продуктивності.

Завдяки описаному підходу забезпечується:

- надійна взаємодія між усіма компонентами системи;
- швидке завантаження та коректне відображення сцен;
- можливість розширення функціоналу без критичних змін у коді;
- адаптивність до різних платформ і користувацьких сценаріїв.

Отже, запропонована архітектура програмного комплексу відповідає сучасним вимогам до побудови інтерактивних 3D-застосунків, забезпечує зручність, надійність та ефективність як для користувачів, так і для розробників.

РОЗДІЛ 3
РОЗРОБКА БАЗИ ДАННИХ ТА ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Логічне та фізичне проектування бд

Ефективна робота програмного комплексу «Онлайн-конструктор інтер’єрів» неможлива без надійної бази даних, яка зберігає всю необхідну інформацію: від облікових записів користувачів до описів складних 3D-сцен. У цьому пункті розглядається процес логічного та фізичного проєктування БД, що визначає її структуру, взаємозв’язки між об’єктами та принципи збереження й обробки інформації.

Логічне проектування

На етапі логічного проектування визначаються основні сутності та їх взаємозв’язки. У рамках даного проекту було виокремлено такі сутності:

- Користувач (User) – зберігає персональні дані користувача, облікові дані, налаштування профілю.
- Проект (Project) – описує окрему сцену, створену користувачем.
- Кімната (Room) – містить інформацію про параметри приміщення (розміри, колір стін, покриття підлоги тощо).
- Меблі (Furniture) – набір елементів інтер’єру, які користувач додає до сцени.
- Модель (Model) – зберігає 3D-об’єкти, їх позиціювання, масштабування, орієнтацію.
- Текстура (Texture) – посилання на візуальні матеріали для рендерингу (поверхня меблів, стін, підлоги).

Нижче представлена ER-діаграма, яка відображає ці сутності та їхні зв’язки (див. рис. 3.1).

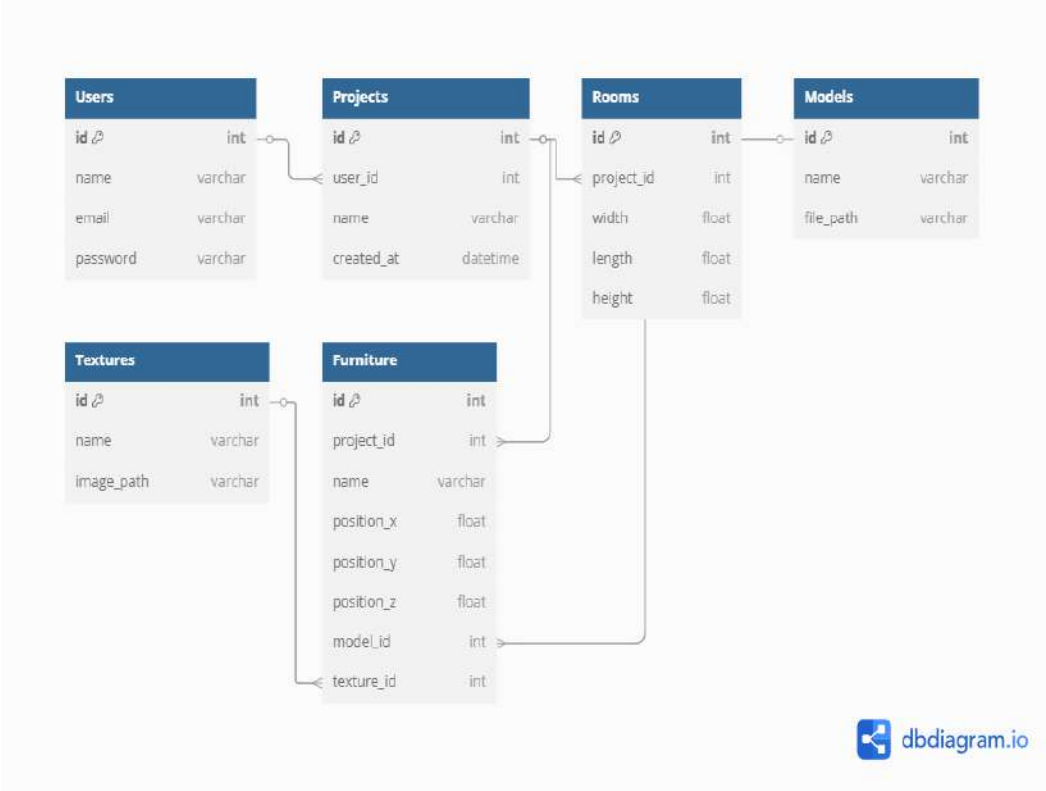

Рис. 3.1. Логічна структура бази даних програмного комплексу

Фізичне проектування

Після визначення логічної структури переходять до фізичної реалізації — вибору технології та структури збереження. У рамках цього проекту використовується документоорієнтована база даних MongoDB, яка оперує даними у форматі JSON-документів. Це забезпечує гнучкість та масштабованість, особливо при зберіганні складних об’єктів (наприклад, 3D-сцени з вкладеними елементами).

Приклад фізичного представлення колекції projects в MongoDB:

```
{
  "_id": "abc123",
  "userId": "u456",
  "name": "Living Room Project",
  "room": {
    "width": 5.0,
    "height": 3.0,
```

```
   "length": 7.0,
   "wallColor": "#DDEEFF"
  },
  "furniture": [
   {
    "modelId": "m789",
    "position": [1.2, 0.0, 3.4],
    "rotation": [0, 90, 0],
    "scale": [1, 1, 1]
   }
  ],
  "createdAt": "2025-06-01T12:00:00Z"
}
```


Такий формат дозволяє швидко зчитувати всю сцену одним запитом, що критично для продуктивної роботи веб- та десктопних застосунків. У структурі документа об’єднані всі необхідні атрибути, що стосуються проекту, кімнати та меблів.

Висновок:

Логічна модель БД охоплює всі ключові сутності та зв’язки між ними, а фізична модель, реалізована через MongoDB, забезпечує гнучке зберігання та масштабованість даних. Такий підхід дозволяє системі ефективно функціонувати як у справжньому часі, так і при обробці великої кількості даних.

3.2. Структура даних про користувачів та проекти

Для забезпечення функціонування системи, орієнтованої на створення, збереження та візуалізацію інтер’єрних 3D-проєктів, було реалізовано структуру даних, що охоплює ключові сутності — користувача (User) та проект (Project). Їхнє логічне представлення реалізовано за допомогою схем

Mongoose, що дозволяє ефективно взаємодіяти з базою даних MongoDB у середовищі Node.js.

Структура користувача

Модель User відповідає за збереження основної облікової інформації щодо користувачів системи. Схема включає такі поля, наведені в таблиці 3.1:

Таблиця 3.1

Опис полів сутності користувача

Назва поля	**Тип даних**	**Обов'язкове**	**Опис**
name	String	так	Ім'я користувача, мінімум 3 символи
email	String	так	Унікальний email, перевірка на формат
password	String	так	Пароль (хешується з використанням bcrypt)
createdAt	Date	так	Дата додавання запису (встановлюється автоматично)
updatedAt	Date	так	Останнє оновлення (генерується автоматично)

Також у моделі реалізовано метод comparePassword для порівняння паролів при авторизації.

Структура проекту

Модель Project безпосередньо пов'язана з користувачем (через поле userId) і містить повний набір параметрів, необхідних для формування інтер'єру. Детальний опис полів подано в таблиці 3.2.

Таблиця 3.2

Структура документа проекту в базі даних

Назва поля	**Тип даних**	**Обов’язкове**	**Опис**
userId	ObjectId (ref: User)	так	Зовнішній ключ користувача
name	String	так	Назва проєкту
room.width	Number	так	Ширина кімнати
room.height	Number	так	Висота кімнати
room.color	String	так	Колір стін
room.floorColor	String	так	Колір підлоги
room.lightIntensity	Number	ні	Інтенсивність освітлення (за замовчуванням – 1)
room.lightIntensity	Number	ні	Інтенсивність освітлення (за замовчуванням – 1)
furniture	Array	так	Масив об’єктів інтер’єру
createdAt	Date	так	Дата створення
updatedAt	Date	так	Дата останнього оновлення

Стосунок між сутностями

Стосунок між моделями User і Project є класичним прикладом «один-до-багатьох»: один користувач може мати кілька проектів, проте кожен проект належить лише одному користувачу. Завдяки механізму референцій у Mongoose (ref: 'User') забезпечується логічна зв’язність даних на рівні бази (дивись рис. 3.2).

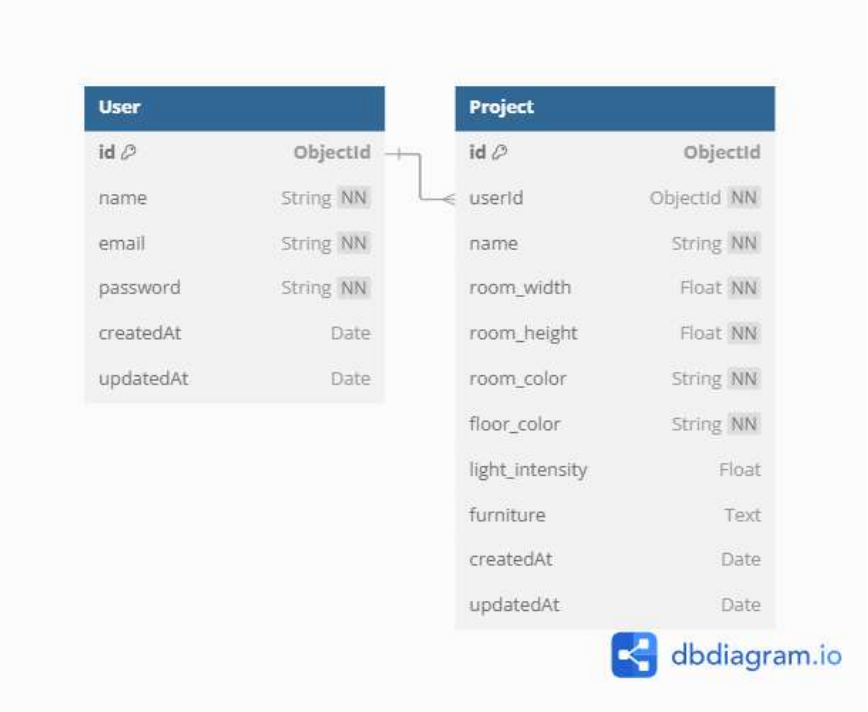

Рис. 3.2. Схема зв’язків між сутностями User та Project (ER-діаграма або UML-діаграма)

3.3. Механізм збереження 3d-моделей і текстур

Візуалізація тривимірних сцен, що відображають інтер’єрні проекти користувачів, неможлива без ефективного механізму збереження і керування 3D-моделями та пов’язаними з ними текстурами. Для досягнення цієї мети в системі реалізовано дві окремі, але взаємопов’язані схеми: Model і Furniture, що забезпечують зберігання цифрових об’єктів та їхніх візуальних характеристик.

Модель Model

Схема Model є базовим представленням тривимірного об'єкта, що може використовуватись у різних контекстах:

- name — текстове поле, яке вказує назву моделі. Це можуть бути узагальнені об’єкти типу "шафа", "стіл", "диван".
- filePath — шлях до відповідного файлу моделі на сервері. Файл містить геометричну інформацію (у форматі .obj, .fbx, .gltf або іншому).

- createdAt, updatedAt — автоматичні мітки часу створення й оновлення.

Ця структура дозволяє централізовано зберігати 3D-моделі та уникати дублювання при багаторазовому використанні тих самих об’єктів у різних проектах.

Модель Furniture

Модель Furniture слугує специфічною реалізацією об’єктів інтер’єру з додатковими візуальними параметрами:

- name — назва елемента меблів (наприклад, “Кутовий диван”).
- previewImage — шлях до зображення-прев’ю, що використовується у вебінтерфейсі для візуального вибору об'єкта користувачем.
- modelPath — шлях до пов’язаної тривимірної моделі.

Таким чином, Furniture виступає в ролі каталогу доступних об'єктів, які користувач може додати до кімнати у редакторі.

Порівняльну таблицю моделей можете глянути в Таблиці 3.3.

Таблиця 3.3

Спільні та відмінні поля моделів та меблів

Поле	**Model**	**Furniture**	**Призначення**
name	+	+	Назва моделі або меблевого об’єкта
filePath	+	–	Шлях до загального 3D-файлу
modelPath	–	+	Шлях до конкретної моделі меблів
previewImage	–	+	Зображення-прев’ю для вибору у веб-інтерфейсі
createdAt	+	–	Дата створення моделі
updatedAt	+	–	Дата останнього оновлення

Організація файлів на сервері

У рамках даного проекту створена структура директорій, що передбачає наявність папки /uploads/models для збереження 3D-моделей та пов’язаних зображень. Проте на поточному етапі розробки фізичні файли моделей (наприклад, .obj, .fbx) фактично знаходяться поза межами системи, у спеціально підготовлених директоріях із оригінальними моделями. Система працює зі шляхами до цих файлів, що задаються вручну або передаються при створенні об’єкта.

Такий підхід дозволяє:

- швидко інтегрувати наявні бібліотеки моделей;
- уникнути дублювання великих файлів у межах репозиторію;
- залишити можливість для подальшої реалізації автоматичного завантаження та зберігання у директорії /uploads (дивись рис.3.3).

Рис. 3.3. Структура папки /uploads/models – резервне сховище для майбутньої інтеграції 3D-файлів

3.4. Реалізація api для взаємодії з базою даних

Для забезпечення ефективної взаємодії між клієнтською частиною (React-додатком) та сервером, у системі реалізовано RESTful API. Комунікація організована через окремі маршрути, які обробляються відповідними контролерами, що оперують даними бази MongoDB за допомогою ODM-бібліотеки Mongoose.

Принципи побудови API

Всі маршрути згруповано відповідно до сутностей: користувачі, проекти, 3D-моделі, меблі. Запити до сервера проходять через проміжний рівень — authMiddleware, який перевіряє токен доступу, забезпечує автентифікацію та дозволяє контролерам працювати з req.user.

Всі маршрути, що пов’язані з проектами, зібрано у файлі projectRoutes.js. Таблиця 3.4 демонструє перелік ключових маршрутів і їх функціональність:

Таблиця 3.4

Опис маршрутів API для управління проектами

Метод	**Шлях**	**Призначення**
POST	/api/projects/	Створення нового проекту
GET	/api/projects/	Отримання всіх проектів, створених поточним користувачем
GET	/api/projects/user/:id	Отримання проектів за ідентифікатором користувача
PUT	/api/projects/:id	Оновлення існуючого проекту
DELETE	/api/projects/:id	Видалення проекту за ідентифікатором

Контролер обробки запитів

У файлі projectController.js реалізовано функції для кожного маршруту:

- saveProject — створення нового об’єкта Project на основі даних про кімнату (room), меблі (furniture) та назву (name);
- getProjects — повернення всіх проектів поточного користувача;
- updateProject, deleteProject — оновлення або видалення існуючих проєктів;
- getProjectsByUser — пошук проектів за userId.

Контролери використовують async/await, обробку помилок через try...catch і відповідають на запити за допомогою res.status().

Авторизація

Використання authMiddleware забезпечує безпечний доступ до API лише авторизованим користувачам. Мідлвар:

- отримує JWT-токен з заголовка Authorization;
- перевіряє його дійсність;
- додає в req.user ідентифікатор користувача для подальшої обробки запитів до БД (дивись рис. 3.4).

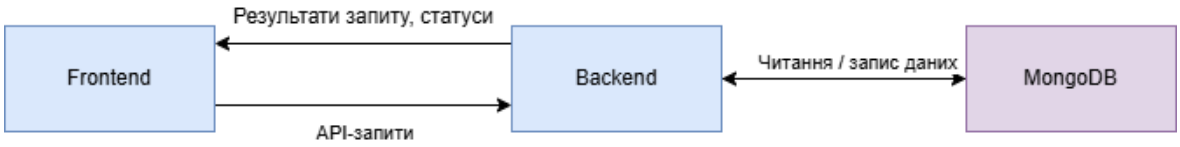

Рис. 3.4. Схема взаємодії між клієнтською частиною, сервером і базою даних

Висновки до розділу 3

У цьому розділі було виконано повноцінне проектування та реалізацію інформаційного забезпечення системи, що включає логічну структуру бази даних, формат збереження цифрових об’єктів та механізми взаємодії через API. Основну увагу приділено ключовим сутностям — користувачам, проектам, 3D-моделям та інтер’єрним об’єктам (меблям).

Модель User реалізує базову автентифікацію та ідентифікацію користувачів, тоді як Project зберігає параметри кімнати, вибір меблів і візуальні характеристики проекту. Окремо спроектовано структури Model та Furniture, що дозволяють ефективно управляти цифровими об’єктами, забезпечуючи гнучкість та масштабованість при майбутньому розширенні функціоналу.

API-інтерфейс, побудований за принципами REST, дозволяє здійснювати повний набір CRUD-операцій для проектів та інших сутностей. Він підтримує автентифікацію на основі JWT та захищає доступ до ресурсів

через middleware-логіку. Це дозволяє гарантувати безпечну і цілісну роботу з персональними даними користувачів.

Таким чином, створена архітектура бази даних та API є надійною основою для реалізації функціонального користувацького інтерфейсу, що забезпечить взаємодію з даними на рівні веб- і десктопного застосунку, що буде розглянуто у наступному розділі.

РОЗДІЛ 4
РЕАЛІЗАЦІЯ ВЕБ- І ДЕСКТОПНОГО ЗАСТОСУНКУ

4.1. Розробка веб інтерфейсу у react

Фронтенд-застосунок реалізовано з використанням JavaScript-бібліотеки React, яка забезпечує побудову компонентно-орієнтованих, динамічних інтерфейсів. Архітектура фронтенду побудована з урахуванням розділення логіки на сторінки, компоненти й сервіси для взаємодії з API.

Загальна структура

Директорія frontend/src містить такі основні частини:

- components/ — багаторазові інтерфейсні елементи (наприклад, Navbar, ProjectCard, FurnitureItem);
- pages/ — повноцінні сторінки, такі як LoginPage, Dashboard, EditorPage;
- services/ — окремі модулі для запитів до API (projectService.js, authService.js);
- App.js — головний компонент, який містить маршрутизацію (React Router) і базову логіку рендерингу;
- index.js — точка входу у застосунок.

Сторінки та функціональність

Основні сторінки веб-інтерфейсу включають:

- Головна сторінка (Dashboard) — це стартова сторінка онлайн-конструктора інтер'єру, яка дозволяє користувачу розпочати роботу над проєктом. На ній представлено короткий опис можливостей сервісу, приклади візуалізацій кімнат, а також пояснення, як працює система (дивись рис. 4.1).
- Редактор (EditorPage) — надає інструменти для додавання, переміщення та видалення 3D-об’єктів у межах сцени. Здійснюється

інтеграція з тривимірним візуалізатором (наприклад, через canvas/WebGL).

- Форма входу та реєстрації (Login/Register) — реалізує аутентифікацію через введення email і пароля. При успішному вході генерується JWT-токен.

Робота з API

Зв’язок із сервером забезпечено через бібліотеку Axios, яка використовується у сервісних модулях. Запити захищені JWT-токеном, який передається у заголовку Authorization.

Приклад запиту до API (отримання проєктів):

```
const getProjects = async () => {
  const res = await axios.get("/api/projects", {
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });
  return res.data;
};
```

Стан і маршрутизація

Для керування станом використано React Hooks (useState, useEffect), а для переходу між сторінками — React Router v6. Завдяки цьому інтерфейс є односторінковим (SPA) і працює без перезавантаження сторінки.

Рис. 4.1. Інтерфейс головної сторінки веб-додатку

4.2. Реєстрація, авторизація та керування проектами

Для забезпечення безпечного доступу до функціоналу системи реалізовано механізм реєстрації та авторизації користувачів на основі JWT-токенів. Користувачі після входу до системи отримують доступ до створення, редагування та перегляду власних 3D-проєктів.

Реєстрація та авторизація

Процес реєстрації та входу користувачів реалізовано за допомогою:

- Frontend: форм LoginForm та RegisterForm, розміщених на відповідних сторінках React-додатку.
- Backend: маршрутів /api/auth/register та /api/auth/login, оброблених у authController.js.

Приклад запиту на реєстрацію:

```
const register = async (userData) => {
  await axios.post(«/api/auth/register», userData);
};
```

Обробка запиту на сервері:

```
const user = new User({ name, email, password });
await user.save();
```

Після успішної авторизації користувач отримує JWT-токен, який зберігається в localStorage та додається до кожного запиту через заголовок Authorization.

Middleware для захисту API

На бекенді реалізовано middleware authMiddleware.js, який перевіряє наявність токена, його дійсність і додає до запиту поле req.user, що містить ID автентифікованого користувача.

```
Const token = req.headers.authorization?.split(« «)[1];
const decoded = jwt.verify(token, process.env.JWT_SECRET);
req.user = { id: decoded.id };
```

Це забезпечує доступ до ресурсів виключно для авторизованих користувачів.

Керування проектами

Після входу користувач потрапляє на панель управління проектами (Dashboard), де реалізовано:

- відображення списку всіх власних проектів;
- кнопки для створення нового, редагування та видалення;
- передачу даних через API /api/projects.

Приклад створення проекту:

```
const createProject = async (data) => {
  await axios.post(«/api/projects», data, {
    headers: { Authorization: `Bearer ${token}` },
  });
};
```

Результат:

Проект зберігається у базі даних MongoDB з полями: name, room, furniture, userId.

Інтерфейс керування

Кожен проект виводиться через компонент ProjectCard, що містить:

- назву проекту;
- кнопку «Переглянути/редагувати»;
- кнопку «Видалити»;
- дату створення.

Користувач може швидко переміщатися між сторінками, завдяки React Router, і бачити лише свої проекти, фільтровані за userId (дивись рис.4.2).

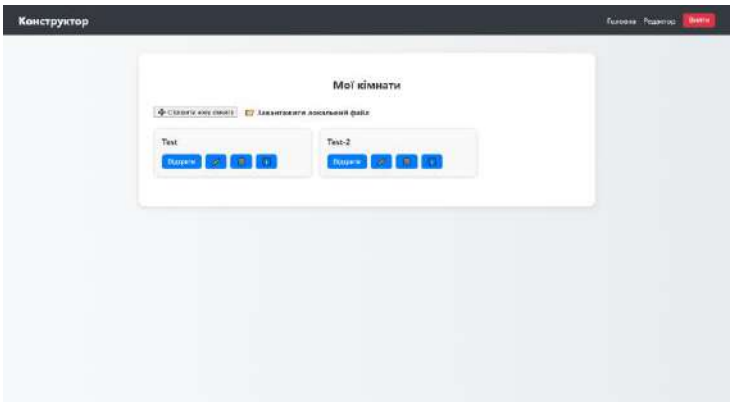

Рис. 4.2. Інтерфейс панелі керування кімнатами (проектами) у вебзастосунку після авторизації

4.3. Розробка десктопного переглядача на WPF+OpenGL

Окрім вебзастосунку, для користувачів реалізовано десктопну програму на базі Windows Presentation Foundation (WPF) з інтеграцією OpenGL для відображення тривимірних сцен. Такий підхід забезпечує високу ефективність під час роботи з великими 3D-проєктами, зокрема в офлайн-режимі.

Загальна структура застосунку

Проект реалізовано мовою C# з використанням WPF як базового фреймворку для побудови інтерфейсу. Основні частини структури:

- MainWindow.xaml / MainWindow.xaml.cs — головне вікно з рендерингом сцени;
- Views/ — сторінки перегляду;
- Models/ — класи-репрезентації об’єктів сцени;
- Services/ — завантаження моделей, обробка JSON, кешування;
- Rendering/ — основна логіка відображення 3D-моделей з OpenGL;
- config.json — конфігураційний файл з параметрами сцени або середовища.

Інтерфейс користувача

Інтерфейс десктопного переглядача побудований за принципом «мінімалізму» — користувач бачить:

- основну сцену з кімнатою;
- навігаційні елементи (наприклад, завантаження файлу, кнопки зміни положення об’єктів);
- індикатори обраного об’єкта.

Управління здійснюється за допомогою миші та клавіатури — обертання камери, масштабування, вибір елементів сцени. Логічна структура десктопного застосунку представлена в таблиці 4.1.

Обробка 3D-моделей

Застосунок підтримує завантаження моделей у форматі .obj, що обробляються у модулі Rendering/ObjLoader.cs (або аналогічному). Відображення реалізовано за допомогою:

- OpenTK або іншої OpenGL-бібліотеки;
- матриць трансформації для обертання, переміщення та масштабування;
- шейдерів для візуальних ефектів (освітлення, тіні тощо).

Зв’язок із проектами

Застосунок може працювати як автономно (офлайн), завантажуючи збережені JSON-файли або 3D-ресурси, так і потенційно в синхронізації з backend, у майбутньому — через REST API або локальне кешування даних.

Таблиця 4.1

Компоненти клієнтської частини перегляду 3D-сцен

Назва компонента	**Призначення**
MainWindow	Основне вікно перегляду сцени
SceneRenderer	Обробка та відображення моделей через OpenGL
ModelLoaderService	Завантаження .obj-моделей
ProjectService	Завантаження JSON-конфігурацій проекту

4.4. Реалізація рендерингу у 3d-сцен

Візуалізація тривимірного інтер’єру є ключовою частиною функціоналу системи. У десктопному застосунку відображення сцени реалізовано з використанням OpenGL у поєднанні з WPF. Такий підхід забезпечує високу продуктивність та візуальну деталізацію при обробці складних 3D-композицій.

Приклад відображення сцени у середовищі десктопного застосунку подано на рисунку 4.3.

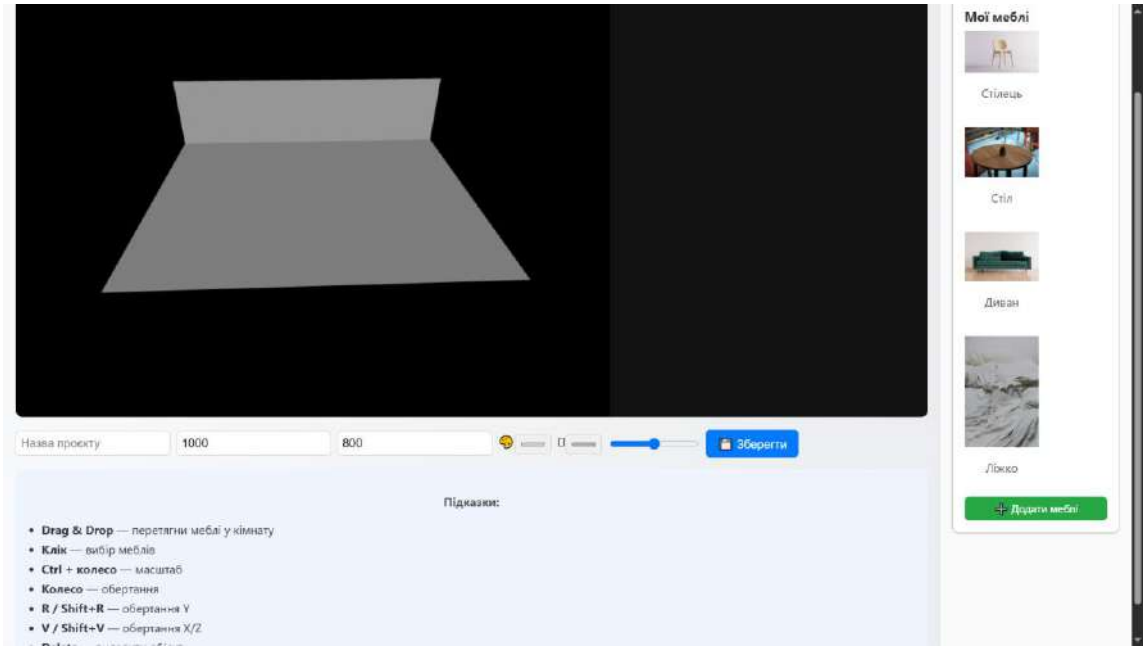

Рис. 4.3. Вікно десктопного застосунку з 3D-сценою та панеллю меблів

Основні етапи рендерингу

Процес рендерингу сцени включає послідовність таких кроків:

- Завантаження моделі — файл у форматі .obj обробляється спеціальним парсером (наприклад, ObjLoader.cs), де визначаються вершини, грані та UV-координати.
- Ініціалізація сцени — створення OpenGL-контексту, буферів, текстур.

- Побудова сцени — трансформація моделей, застосування позиціювання згідно координат у проекті.
- Накладення текстур та освітлення — застосування матеріалів, кольору підлоги, інтенсивності освітлення.
- Рендеринг — виклик основної функції DrawScene() для виводу кадру.

Камера та навігація

У сцені реалізовано камеру типу OrbitCamera (або аналогічну), яка дозволяє:

- обертати сцену навколо центра;
- масштабувати за допомогою коліщатка миші;
- переміщатися по осі X/Y через клавіші або drag.

Це забезпечує користувачеві повний огляд проекту з різних ракурсів.

Джерела світла

У сцені використовується базове фонове та направлене освітлення, що забезпечує глибину і реалістичність відображення. В деяких реалізаціях застосовуються прості шейдери з підтримкою освітлення за моделлю Фонга або Ламберта.

Параметр lightIntensity задається у JSON-структурі проєкту і передається в рендерер. Основні параметри 3D-сцени, що впливають на візуалізацію, наведено у таблиці 4.2.

Таблиця 4.2

Параметри 3D-сцени в JSON-структурі проекту

Параметр	**Опис**	**Джерело**
width	Ширина кімнати у метрах	JSON → room.width
height	Висота кімнати	JSON → room.height
floorColor	Колір підлоги	JSON → room.floorColor
lightIntensity	Яскравість сцени	JSON → lightIntensity
furniture[]	Масив 3D-моделей з позицією	JSON → furniture

Продуктивність

Оскільки для рендерингу використовується OpenGL, обчислення відбуваються апаратно (через GPU), що дозволяє працювати навіть з кількома десятками об’єктів без втрати FPS.

Також реалізовано буферизацію сцен для швидкого перерисовування та можливість оптимізації через frustum culling (опційно).

4.5. Тестування програмного забезпечення

Для перевірки коректності роботи системи було проведено комплексне тестування веб- та десктопного застосунку. Тестування охоплювало функціональні, інтеграційні та ручні перевірки з акцентом на стабільність, відповідність вимогам і зручність взаємодії користувача з системою.

Мета тестування

Основною метою тестування було:

- перевірити працездатність критичних модулів системи;
- виявити помилки в логіці взаємодії між компонентами;
- оцінити стабільність рендерингу 3D-сцен;
- переконатися у правильності обробки API-запитів та відповідей;
- перевірити поведінку інтерфейсу при некоректних діях користувача.

Методика

Тестування проводилось у двох основних середовищах (див. табл. 4.3):

Таблиця 4.3

Середовища, платформи та інструменти тестування застосунку

Середовище	**Платформа**	**Інструменти/Метод**
Веб	React	Ручне тестування, DevTools, Postman
Десктоп	WPF + OpenGL	Ручне тестування, перевірка рендерингу, лог-файли

У вебзастосунку особливу увагу приділено авторизації, створенню та редагуванню проєктів, а також взаємодії з API. У десктопній частині — відображенню сцени, додаванню меблів, реакції на дії користувача (обертання, масштабування).

Випробувані сценарії

Було протестовано такі типові сценарії:

1. Реєстрація та вхід до системи з валідними/невалідними даними.
2. Створення проекту, збереження параметрів кімнати, додавання меблів.
3. Редагування вже існуючого проекту.
4. Робота з 3D-сценою у десктопному застосунку:
    - завантаження сцени;
    - переміщення камери;
    - відображення меблів;
    - обертання/масштабування.

Результати

Підсумки перевірки ключових функцій наведено в таблиці 4.4, яка демонструє відповідність фактичних результатів очікуваним.

У результаті тестування встановлено, що:

- система стабільно працює у межах визначених функцій;
- обробка запитів API відповідає очікуваним результатам;
- 3D-сцена успішно рендериться в десктопному переглядачі;
- можливі помилки (наприклад, порожні поля при реєстрації або некоректні формати даних) обробляються з виводом повідомлення для користувача;
- функціонал збереження і відновлення проєктів працює без втрати даних.

Таблиця 4.4

Результати функціонального тестування основних сценаріїв

№	Сценарій	Очікуваний результат	Фактичний результат	Статус
1	Вхід з коректним email/паролем	Перехід до Dashboard	Вхід виконано успішно	Успішно
2	Створення нового проекту	Відображення проекту у списку	Проект створено	Успішно
3	Відкриття проекту в EditorPage	Візуалізація сцени з меблями	Рендеринг відбувся	Успішно
4	Завантаження сцени у десктопі	Відображення об’єктів у 3D	Сцена завантажилась	Успішно
5	Введення некоректного email	Повідомлення про помилку	Помилка виведена	Оброблено

Висновки до розділу 4

У цьому розділі було реалізовано основні інтерфейсні компоненти системи як для вебплатформи, так і для десктопного середовища. Застосування технології React для фронтенду дозволило створити адаптивний, інтерактивний та зручний інтерфейс, що забезпечує повноцінну роботу з проектами, а також авторизацію користувачів через JWT-токени.

Реалізований RESTful API гарантує надійну взаємодію між клієнтом і сервером, забезпечує CRUD-операції з проектами та підтримує механізми захисту й перевірки доступу.

Десктопний застосунок, створений на базі WPF з використанням OpenGL, дозволяє візуалізувати 3D-сцени з високою продуктивністю. Інтеграція функціоналу управління камерою, освітленням і об’єктами інтер’єру робить перегляд сцени гнучким і зручним для кінцевого користувача.

Проведене тестування підтвердило стабільність та функціональну повноту програмного комплексу. Виявлені незначні помилки були виправлені

на етапі перевірки, і всі ключові сценарії взаємодії пройшли успішну перевірку.

Таким чином, реалізована система повністю відповідає технічному завданню, а створена архітектура дозволяє в майбутньому розширювати функціонал, включати нові типи об’єктів та вдосконалювати UX.

ВИСНОВКИ

Під час виконання дипломної роботи було створено програмний комплекс для створення, редагування та перегляду інтер’єрних 3D-проєктів. Робота охоплює повний цикл розробки: від аналізу предметної області до впровадження веб- і десктопного інтерфейсів та їх тестування.

На етапі аналізу було визначено основні функціональні вимоги до системи, досліджено існуючі рішення та обґрунтовано вибір технологій. Зокрема, обрано стек на основі Node.js + MongoDB для серверної частини, React — для вебінтерфейсу, та WPF + OpenGL — для десктопного переглядача.

Розроблено базу даних, яка включає сутності User, Project, Furniture, Model, а також реалізовано повноцінне API для взаємодії між клієнтською частиною та сервером. Особливу увагу приділено забезпеченню авторизації користувачів, захисту даних і масштабованості структури.

У фронтенді реалізовано сучасний інтерфейс, що підтримує створення та редагування проектів. У десктопному застосунку реалізовано 3D-візуалізацію сцени з підтримкою керування камерою, освітленням і об’єктами інтер’єру.

Проведене тестування підтвердило, що система відповідає поставленим вимогам, демонструє стабільність, коректність роботи та зручність використання.

Отже, мету дипломної роботи було повністю досягнуто. Створений програмний комплекс може бути використаний як основа для професійного застосування у сфері дизайну інтер’єру, а також легко адаптується до інших задач, пов’язаних з 3D-моделюванням і візуалізацією.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Абрамян М. Э. Visual C# на примерах / Абрамян М. Э. – СПб. : БХВ-Петербург, 2008. – 496 с.
2. Петцольд Ч. Программирование для Microsoft Windows на C# / Петцольд Ч. – М. : ИТД «Русская Редакция», 2002. – 576 с.
3. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft NET Framework 4.5 на языке C# / Рихтер Дж. – Питер, 2013. – 896 с.
4. Цехнер Марио. OpenGL. Программирование трёхмерной графики / Цехнер Марио. – М. : Питер, 2011. – 416 с.
5. Зеленков Ю. Введение в базы данных / Зеленков Ю. – СПб. : Питер, 2003.
6. Кузнецов С. Д. Основы современных баз данных / Кузнецов С. Д. – М. : Интернет-Университет Информационных Технологий; БИНОМ. Лаборатория знаний. – 2007. – 484 с.
7. Флэнаган Д. JavaScript. Подробное руководство / Флэнаган Д. – СПб. : Символ-Плюс, 2007. – 960 с.
8. Burnette E. Hello, Android. Программирование для Android 2-е изд. / Burnette E. – M. : Sourcebooks, Inc., 2009. – 250 с.
9. MongoDB: The Definitive Guide / Честейн К., Браун К. – O'Reilly Media, 2019. – 400 с.
10. Платформа ADO.NET Entity Framework [Электронный ресурс]. – Режим доступа : http://msdn.microsoft.com/ru-ru/library/bb399572.aspx.
11. React Documentation [Электронный ресурс]. – Режим доступа : https://reactjs.org/docs/getting-started.html.
12. Node.js Documentation [Электронный ресурс]. – Режим доступа : https://nodejs.org/en/docs/.
13. Mongoose ODM Documentation [Электронный ресурс]. – Режим доступа : https://mongoosejs.com/docs/.
14. OpenTK Library [Электронный ресурс]. – Режим доступа : https://opentk.net/learn/.

15. Postman API Platform [Электронный ресурс]. – Режим доступа : https://www.postman.com/.

16. Киммел П. Программирование на C#. Советы эксперта / Киммел П. – М. : Вильямс, 2009. – 720 с.

17. Тролсен Э., Японикс П. Pro C# 10 и .NET 6 / Тролсен Э., Японикс П. – М. : Apress, 2022. – 1200 с.

18. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э. и др. – СПб. : Питер, 2020. – 384 с.

19. Макконнелл С. Совершенный код. Мастер-класс / Макконнелл С. – М. : Вильямс, 2004. – 896 с.

20. Макрейнолдс Т., Блайт Д. Программирование графики с использованием OpenGL / Макрейнолдс Т., Блайт Д. – М. : ДМК Пресс, 2007. – 672 с.

21. Фоли Дж. Компьютерная графика: Принципы и практика / Фоли Дж. и др. – М. : Вильямс, 2014. – 1175 с.

22. Мембри П., Плагге Э., Хокстра Т. MongoDB. Подробное руководство / Мембри П. и др. – М. : ДМК Пресс, 2011. – 328 с.

23. Дейли Б. NoSQL и MongoDB за 24 часа / Дейли Б. – М. : Эксмо, 2013. – 400 с.

24. Сегуин К. Краткое руководство по MongoDB [Электронный ресурс]. – Режим доступа : https://github.com/karlseguin/the-little-mongodb-book.

25. Смит Дж. Entity Framework Core на практике / Смит Дж. – М. : Manning Publications, 2022. – 520 с.

ДОДАТКИ

Додаток А

Код програмного забезпечення на мові JavaScript (фронтенд)

```
import React, { useState, useRef, useEffect } from "react";
import RoomCanvas from "../components/RoomCanvas";
import FurnitureSelector from "../components/FurnitureSelector";
import UploadFurnitureForm from "../components/UploadFurnitureForm";
import FurnitureEditorPanel from "../components/FurnitureEditorPanel";
import { saveProject, updateProject } from "../services/api";
import axios from "axios";
import { useLocation } from "react-router-dom";
import { useNotification } from "../context/NotificationContext";
import "../styles/Editor.css";

const Editor = () => {
  const [name, setName] = useState("");
  const [roomWidth, setRoomWidth] = useState(1000);
  const [roomHeight, setRoomHeight] = useState(800);
  const [wallColor, setWallColor] = useState("#cccccc");
  const [floorColor, setFloorColor] = useState("#aaaaaa");
  const [lightIntensity, setLightIntensity] = useState(1);
const [showUploadForm, setShowUploadForm] = useState(false);
  const [reloadKey, setReloadKey] = useState(0);
  const [selectedObject, setSelectedObject] = useState(null);
  const [projects, setProjects] = useState([]);
  const roomCanvasRef = useRef(null);
  const location = useLocation();
  const showNotification = useNotification();
  const importedProject = location.state?.importedProject;
  const editingProjectId = location.state?.projectId;

  useEffect(() => {
    if (importedProject) {
      setName(importedProject.name || "");
      setRoomWidth(importedProject.room?.width || 1000);
      setRoomHeight(importedProject.room?.height || 800);
      setWallColor(importedProject.room?.color || "#cccccc");
      setFloorColor(importedProject.room?.floorColor || "#aaaaaa");
      setLightIntensity(importedProject.room?.lightIntensity ?? 1);

roomCanvasRef.current?.loadFurnitureFromProject(importedProject.furniture || []);
    } else if (editingProjectId) {
axios.get(`/api/projects/${editingProjectId}`).then(({ data }) => {
        setName(data.name);
        setRoomWidth(data.room.width);
        setRoomHeight(data.room.height);
        setWallColor(data.room.color);
        setFloorColor(data.room.floorColor);
        setLightIntensity(data.room.lightIntensity ?? 1);
        roomCanvasRef.current?.loadFurnitureFromProject(data.furniture || []);
      });
    }
  }, [importedProject, editingProjectId]);

  const handleRoomDimensionChange = (setter) => (e) => {
    setter(+e.target.value);
  };
  const handleSave = async () => {
    if (!name.trim()) {
      showNotification("Введіть назву проєкту", "error");
      return;
    }
    const furnitureState = roomCanvasRef.current?.getFurnitureState?.() || [];
    const projectData = {
      name,
```

Продовження додатку А

```
      room: {
       width: roomWidth,
       height: roomHeight,
       color: wallColor,
       floorColor,
       lightIntensity,
      },
      furniture: furnitureState,
    };
    try {
      if (editingProjectId) {
        await updateProject(editingProjectId, projectData);
        showNotification("Зміни збережено", "success");
      } else {
        await saveProject(projectData);
        showNotification("Проєкт створено", "success");
      }
  } catch (err) {
      console.error("Помилка збереження", err);
      showNotification("Помилка збереження проєкту", "error");
    }
  };

  const handleFurnitureClick = (item) => {
    roomCanvasRef.current?.addFurnitureItem(item);
  };

  return (
    <div className="editor-layout">
      <div className="editor-main">
        <RoomCanvas
          ref={roomCanvasRef}
          width={800}
          height={600}
roomWidth={roomWidth}
          roomHeight={roomHeight}
          wallColor={wallColor}
floorColor={floorColor}
          lightIntensity={lightIntensity}
          onSelectObject={setSelectedObject}
          showEditorPanel={true}
          showFurniturePanel={true}
        />

        <div className="editor-room-controls">
          <input
            type="text"
            value={name}
            onChange={(e) => setName(e.target.value)}
            placeholder="Назва проєкту"
          />
          <input
            type="number"
            value={roomWidth}
    onChange={handleRoomDimensionChange(setRoomWidth)}
            placeholder="Ширина"
          />
<input
            type="number"
            value={roomHeight}
            onChange={handleRoomDimensionChange(setRoomHeight)}
            placeholder="Висота"
          />
          <label>
            🎨
            <input type="color" value={wallColor} onChange={(e) => setWallColor(e.target.value)} />
          </label>
```

Продовження додатку А

```
        <label>
          
          <input
            type="color"
            value={floorColor}
            onChange={(e) => setFloorColor(e.target.value)}
      />
        </label>
        <input
          type="range"
min="0"
          max="2"
          step="0.1"
          value={lightIntensity}
          onChange={(e) => setLightIntensity(+e.target.value)}
        />
        <button onClick={handleSave}>💾 Зберегти</button>
      </div>

      <FurnitureEditorPanel selectedObject={selectedObject} />

      <div className="editor-hint">
        <p><strong>Підказки:</strong></p>
        <ul>
          <li><strong>Drag & Drop</strong> — перетягни меблі у кімнату</li>
  <li><strong>Клік</strong> — вибір меблів</li>
          <li><strong>Ctrl + колесо</strong> — масштаб</li>
          <li><strong>Колесо</strong> — обертання</li>
          <li><strong>R / Shift+R</strong> — обертання Y</li>
          <li><strong>V / Shift+V</strong> — обертання X/Z</li>
<li><strong>Delete</strong> — видалити об'єкт</li>
        </ul>
       </div>
     </div>
     <div className="editor-sidebar">
       <div className="furniture-selector-panel">
         <h3>Мої меблі</h3>
         <FurnitureSelector onSelect={handleFurnitureClick} reloadTrigger={reloadKey} />
         <button onClick={() => setShowUploadForm(!showUploadForm)}>➕ Додати меблі</button>
         {showUploadForm && (
           <UploadFurnitureForm
             onUploadSuccess={() => {
               setShowUploadForm(false);
               setReloadKey((prev) => prev + 1);
             }}
           />
         )}
       </div>
     </div>
</div>
  );
};
export default Editor;

/*=================*/

import React, { useEffect } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchProjects } from '../features/projectSlice';

const ProjectList = () => {
  const dispatch = useDispatch();
  const projects = useSelector((state) => state.projects.items);

  useEffect(() => {
    dispatch(fetchProjects());
  }, [dispatch]);
```

Продовження додатку А

```
 return (
   <div>
     <h1>Мої Проекти</h1>
{projects.map((project) => (
       <div key={project.id}>
         <h2>{project.name}</h2>
         <button>Редагувати</button>
       </div>
     ))}
   </div>
 );
};

export default ProjectList;

/*=================*/

import axios from "axios";
const API = axios.create({
  baseURL: "http://localhost:5000/api",
});
API.interceptors.request.use((config) => {
  const token = localStorage.getItem("token");
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
export const loginUser = (credentials) => API.post("/auth/login", credentials);
export const registerUser = (data) => API.post("/auth/register", data);
export const fetchUserProjects = (userId) =>
  API.get(`/projects/user/${userId}`);
export const saveProject = (data) => API.post("/projects", data);
export const updateProject = (id, data) => API.put(`/projects/${id}`, data);
export const deleteProject = (id) => API.delete(`/projects/${id}`);
export const fetchFurniture = () => API.get("/furniture");

/*=================*/

import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Login from './pages/Login';
import Register from './pages/Register';
import Home from './pages/Home';
import Editor from './pages/Editor';
import Header from './components/Header';
import PrivateRoute from './components/PrivateRoute';
import ModeSelection from './pages/ModeSelection';
import Cabinet from './pages/Cabinet';
import Download from './pages/Download';
import { NotificationProvider } from './context/NotificationContext';
function App() {
  return (
    <Router>
      <NotificationProvider>
        <Header />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route
            path="/editor"
       element={
             <PrivateRoute>
               <Editor />
             </PrivateRoute>
           }
         />
         <Route
```

Продовження додатку А

```
          path="/mode-selection"
          element={
            <PrivateRoute>
              <ModeSelection />
            </PrivateRoute>
          }
        />
        <Route
          path="/cabinet"
          element={
            <PrivateRoute>
              <Cabinet />
            </PrivateRoute>
          }
        />
        <Route
          path="/download"
          element={
            <PrivateRoute>
              <Download />
            </PrivateRoute>
          }
        />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
      </Routes>
    </NotificationProvider>
  </Router>
 );
}
export default App;

/*=================*/

import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import App from './App';
import store from './store';
import './styles/styles.css';
ReactDOM.render(
 <Provider store={store}>
  <App />
 </Provider>,
 document.getElementById('root')
);

/*=================*/

   const [color, setColor] = useState('#ffffff');

 useEffect(() => {
  if (selectedObject) {
   selectedObject.traverse((child) => {
    if (child.isMesh && child.material && child.material.color) {
     const currentColor = `#${child.material.color.getHexString()}`;
     setColor(currentColor);
     return;
    }
   });
  }
 }, [selectedObject]);

 const handleColorChange = (e) => {
  const newColor = e.target.value;
  setColor(newColor);

  if (selectedObject) {
```

Продовження додатку А

```
    selectedObject.traverse((child) => {
      if (child.isMesh && child.material && child.material.color) {
        child.material.color.set(newColor);
      }
    });
  }
 };

 const handleTextureApply = (textureUrl) => {
  if (!selectedObject) return;

  const loader = new THREE.TextureLoader();
  loader.load(textureUrl, (texture) => {
    selectedObject.traverse((child) => {
      if (child.isMesh && child.material) {
        child.material.map = texture;
        child.material.needsUpdate = true;
      }
    });
  });
 };

 if (!selectedObject) return null;

 return (
  <div className="furniture-editor-panel">
    <h4>Редагувати обраний об'єкт</h4>
    <label>Колір:</label>
    <input type="color" value={color} onChange={handleColorChange} />
    <br />
    <TextureLibrary onSelectTexture={handleTextureApply} />
  </div>
 );
};

export default FurnitureEditorPanel;


/*=================*/


      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });

    if (onUploadSuccess) onUploadSuccess();
    setName('');
    setModelFile(null);
    setImageFile(null);
    alert('Меблі успішно завантажено!');
  } catch (err) {
    console.error(err);
    setError('Помилка завантаження.');
  } finally {
    setLoading(false);
  }
 };

 return (
  <form onSubmit={handleSubmit} style={{ marginTop: '1rem' }}>
    <h3>Додати власну модель меблів</h3>
    <input
      type="text"
      value={name}
      onChange={(e) => setName(e.target.value)}
      placeholder="Назва меблі"
      required
```

Продовження додатку А

```
      /><br />
      <input
        type="file"
        accept=".obj"
        onChange={(e) => setModelFile(e.target.files[0])}
        required
      /><br />
      <input
        type="file"
        accept="image/*"
        onChange={(e) => setImageFile(e.target.files[0])}
        required
      /><br />
      <button type="submit" disabled={loading}>
        {loading ? 'Завантаження...' : 'Завантажити'}
      </button>
      {error && <p style={{ color: 'red' }}>{error}</p>}
    </form>
  );
};

export default UploadFurnitureForm;

/*=================*/

import React, { useState, useEffect } from 'react';
import * as THREE from 'three';
import TextureLibrary from './TextureLibrary';

const FurnitureEditorPanel = ({ selectedObject }) => {
import React, { useState } from 'react';
import axios from 'axios';

const UploadFurnitureForm = ({ onUploadSuccess }) => {
  const [name, setName] = useState('');
  const [modelFile, setModelFile] = useState(null);
  const [imageFile, setImageFile] = useState(null);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!name || !modelFile || !imageFile) {
      setError('Усі поля обов’язкові.');
      return;
    }

    setLoading(true);
    setError(null);

    const formData = new FormData();
    formData.append('name', name);
    formData.append('model', modelFile);
    formData.append('image', imageFile);

    try {
      const response = await axios.post('/api/furniture', formData, {
import React, { useRef, useEffect, forwardRef, useImperativeHandle } from "react";
import * as THREE from "three";
import { OrbitControls } from "three/examples/jsm/controls/OrbitControls";
import { OBJLoader } from "three/examples/jsm/loaders/OBJLoader";

const RoomCanvas = forwardRef(
  (
    {
      width,
      height,
      roomWidth,
```

Продовження додатку А

```
    roomHeight,
    wallColor,
    floorColor,
    lightIntensity,
    onSelectObject,
    showFurniturePanel,
    showEditorPanel,
  },
  ref
) => {
  const mountRef = useRef();
  const sceneRef = useRef();
  const cameraRef = useRef();
  const rendererRef = useRef();
  const controlsRef = useRef();
  const selectedObjectRef = useRef();
  const raycaster = useRef(new THREE.Raycaster());
  const mouse = useRef(new THREE.Vector2());
  const isDragging = useRef(false);

  useImperativeHandle(ref, () => ({
    getFurnitureState: () => {
      return sceneRef.current.children
        .filter(obj => obj.userData?.modelPath)
        .map(obj => {
          let color = "#ffffff";
          obj.traverse(child => {
            if (child.isMesh && child.material?.color) {
              color = child.material.color.getStyle();
            }
          });
          return {
            modelPath: obj.userData.modelPath,
            position: {
              x: obj.position.x,
              y: obj.position.y,
              z: obj.position.z,
            },
            rotation: {
              x: obj.rotation.x,
              y: obj.rotation.y,
              z: obj.rotation.z,
            },
            scale: {
              x: obj.scale.x,
              y: obj.scale.y,
              z: obj.scale.z,
            },
            color,
          };
        });
    },
    loadFurnitureFromProject: (items) => {
      const loader = new OBJLoader();
      const scene = sceneRef.current;
      const toRemove = scene.children.filter(obj => obj.userData?.modelPath);
      toRemove.forEach(obj => scene.remove(obj));

      items.forEach(item => {
        loader.load(item.modelPath, (object) => {
          const group = new THREE.Group();
          object.traverse(child => {
            if (child.isMesh) {
              child.material = child.material.clone();
              group.add(child.clone());
            }
          });
```

```
      group.scale.set(item.scale.x, item.scale.y, item.scale.z);
      group.position.set(item.position.x, item.position.y, item.position.z);
      group.rotation.set(item.rotation.x, item.rotation.y, item.rotation.z);
      group.userData.modelPath = item.modelPath;

      if (item.color) {
        group.traverse(child => {
          if (child.isMesh && child.material?.color) {
            child.material.color.setStyle(item.color);
          }
        });
      }

      scene.add(group);
    });
  });
 },
 addFurnitureItem: (item) => {
  const loader = new OBJLoader();
  const scene = sceneRef.current;

  loader.load(item.modelPath, (object) => {
    const group = new THREE.Group();
    object.traverse(child => {
      if (child.isMesh) {
        child.material = child.material.clone();
        group.add(child.clone());
      }
    });

    const boundingBox = new THREE.Box3().setFromObject(group);
    const size = new THREE.Vector3();
    boundingBox.getSize(size);
    const maxDimension = Math.max(size.x, size.y, size.z);
    const scaleFactor = 1 / maxDimension;
    group.scale.setScalar(scaleFactor * 0.5);

    const height = size.y * scaleFactor * 0.5;
    group.position.set(0, height, 0);
    group.userData.modelPath = item.modelPath;

    scene.add(group);
  });
 },
}));

useEffect(() => {
 const scene = new THREE.Scene();
 sceneRef.current = scene;

 const camera = new THREE.PerspectiveCamera(60, width / height, 0.01, 500);
 cameraRef.current = camera;

 const renderer = new THREE.WebGLRenderer({ antialias: true });
 renderer.setSize(width, height);
 rendererRef.current = renderer;
 mountRef.current.appendChild(renderer.domElement);

 scene.add(new THREE.AmbientLight(0xffffff, lightIntensity));
 const dirLight = new THREE.DirectionalLight(0xffffff, lightIntensity);
 dirLight.position.set(10, 10, 10);
 scene.add(dirLight);

 const scaledWidth = roomWidth / 100;
 const scaledHeight = roomHeight / 100;

 const floor = new THREE.Mesh(
  new THREE.PlaneGeometry(scaledWidth, scaledHeight),
```

Продовження додатку А

```
  new THREE.MeshStandardMaterial({ color: floorColor })
);
floor.rotation.x = -Math.PI / 2;
floor.name = "floor";
scene.add(floor);

const wall = new THREE.Mesh(
  new THREE.PlaneGeometry(scaledWidth, 2.5),
  new THREE.MeshStandardMaterial({ color: wallColor })
);
wall.position.y = 1.25;
wall.position.z = -scaledHeight / 2;
wall.name = "wall";
scene.add(wall);

const getCameraOffset = () => {
  const base = Math.max(scaledWidth, scaledHeight) * 1.2;
  let offsetX = 0;
  if (showFurniturePanel && !showEditorPanel) offsetX = -0.3 * scaledWidth;
  if (!showFurniturePanel && showEditorPanel) offsetX = 0.3 * scaledWidth;
  if (showFurniturePanel && showEditorPanel) offsetX = -0.15 * scaledWidth;
  return { x: offsetX, z: base };
};

const { x, z } = getCameraOffset();
camera.position.set(x, 2.5, z);
camera.lookAt(0, 0, 0);

const controls = new OrbitControls(camera, renderer.domElement);
controls.target.set(0, 0, 0);
controls.update();
controlsRef.current = controls;

const animate = () => {
  requestAnimationFrame(animate);
  controls.update();
  renderer.render(scene, camera);
};
animate();

const el = renderer.domElement;

const onMouseDown = (event) => {
  const bounds = el.getBoundingClientRect();
  mouse.current.x = ((event.clientX - bounds.left) / bounds.width) * 2 - 1;
  mouse.current.y = -((event.clientY - bounds.top) / bounds.height) * 2 + 1;

  raycaster.current.setFromCamera(mouse.current, camera);
  const intersects = raycaster.current.intersectObjects(scene.children, true);

  let clicked = false;

  for (const intersect of intersects) {
    const obj = intersect.object.parent;
    if (obj.userData?.modelPath) {
      clicked = true;
      if (selectedObjectRef.current === obj) {
        selectedObjectRef.current = null;
        isDragging.current = false;
        onSelectObject?.(null);
      } else {
        selectedObjectRef.current = obj;
        isDragging.current = true;
        onSelectObject?.(obj);
      }
      break;
    }
  }
```

Продовження додатку А

```
  if (!clicked) {
    selectedObjectRef.current = null;
    isDragging.current = false;
    onSelectObject?.(null);
  }
};

const onMouseMove = (event) => {
  if (!isDragging.current || !selectedObjectRef.current) return;

  const bounds = el.getBoundingClientRect();
  mouse.current.x = ((event.clientX - bounds.left) / bounds.width) * 2 - 1;
  mouse.current.y = -((event.clientY - bounds.top) / bounds.height) * 2 + 1;

  raycaster.current.setFromCamera(mouse.current, camera);
  const intersects = raycaster.current.intersectObjects(scene.children, true);
  const floorHit = intersects.find(i => i.object.name === "floor");

  if (floorHit) {
    const obj = selectedObjectRef.current;
    const halfW = roomWidth / 100 / 2;
    const halfH = roomHeight / 100 / 2;
    const snap = (v, step = 0.1) => Math.round(v / step) * step;
    let x = snap(floorHit.point.x);
    let z = snap(floorHit.point.z);
    x = Math.max(-halfW, Math.min(halfW, x));
    z = Math.max(-halfH, Math.min(halfH, z));
    obj.position.x = x;
    obj.position.z = z;
  }
};

const onMouseUp = () => (isDragging.current = false);

const onWheel = (event) => {
  if (!selectedObjectRef.current) return;
  if (event.ctrlKey) {
    const scaleChange = event.deltaY < 0 ? 1.05 : 0.95;
    selectedObjectRef.current.scale.multiplyScalar(scaleChange);
  } else {
    const rotationChange = event.deltaY < 0 ? 0.1 : -0.1;
    selectedObjectRef.current.rotation.y += rotationChange;
  }
};

const onKeyDown = (event) => {
  if (!selectedObjectRef.current) return;
  const obj = selectedObjectRef.current;
  if (event.key === "Delete") {
    scene.remove(obj);
    selectedObjectRef.current = null;
    onSelectObject?.(null);
  } else if (event.key === "r") {
    obj.rotation.y += Math.PI / 2;
  } else if (event.key === "R") {
    obj.rotation.y -= Math.PI / 2;
  } else if (event.key === "v") {
    obj.rotation.x += Math.PI / 2;
  } else if (event.key === "V") {
    obj.rotation.z += Math.PI / 2;
  }
};

el.addEventListener("mousedown", onMouseDown);
el.addEventListener("mousemove", onMouseMove);
el.addEventListener("mouseup", onMouseUp);
el.addEventListener("wheel", onWheel);
window.addEventListener("keydown", onKeyDown);
```

```
    return () => {
      el.removeEventListener("mousedown", onMouseDown);
      el.removeEventListener("mousemove", onMouseMove);
      el.removeEventListener("mouseup", onMouseUp);
      el.removeEventListener("wheel", onWheel);
      window.removeEventListener("keydown", onKeyDown);
      mountRef.current?.removeChild(renderer.domElement);
    };
  }, [
    width,
    height,
    roomWidth,
    roomHeight,
    wallColor,
    floorColor,
    lightIntensity,
    showFurniturePanel,
    showEditorPanel,
    onSelectObject,
  ]);

  return (
    <div
      ref={mountRef}
      className="room-canvas-wrapper"
      onDrop={(e) => {
        e.preventDefault();
        const data = e.dataTransfer.getData("application/json");
        if (data) {
          try {
            const item = JSON.parse(data);
            if (item.modelPath) ref.current?.addFurnitureItem(item);
          } catch (e) {
            console.error("Drop error:", e);
          }
        }
      }}
      onDragOver={(e) => e.preventDefault()}
      tabIndex={0}
    />
  );
 }
);

export default RoomCanvas;

/*=================*/

import React, { createContext, useContext, useState, useCallback } from "react";
import "../styles/Notification.css";

const NotificationContext = createContext();

export const NotificationProvider = ({ children }) => {
 const [message, setMessage] = useState("");
 const [visible, setVisible] = useState(false);

 const notify = useCallback((msg, timeout = 3000) => {
   setMessage(msg);
   setVisible(true);
   setTimeout(() => setVisible(false), timeout);
 }, []);

 return (
   <NotificationContext.Provider value={{ notify }}>
     {children}
     {visible && <div className="notification-popup">{message}</div>}
   </NotificationContext.Provider>
```

Продовження додатку А

```
  );
};

export const useNotification = () => {
  const context = useContext(NotificationContext);
  if (!context) {
    throw new Error("useNotification must be used within a NotificationProvider");
  }
  return context.notify;
};

/*=================*/

import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';
import axios from 'axios';

export const fetchProjects = createAsyncThunk('projects/fetchProjects', async () => {
  const response = await axios.get('/api/projects');
  return response.data;
});

export const projectSlice = createSlice({
  name: 'projects',
  initialState: {
    items: [],
    status: 'idle',
  },
  reducers: {},
  extraReducers: (builder) => {
    builder
      .addCase(fetchProjects.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(fetchProjects.fulfilled, (state, action) => {
        state.status = 'succeeded';
        state.items = action.payload;
      })
      .addCase(fetchProjects.rejected, (state) => {
        state.status = 'failed';
      });
  },
});

export default projectSlice.reducer;
```

Додаток Б

Код серверної частини на Node.js

```
    res.json(projects);
  } catch (err) {
    res.status(500).json({ error: 'Помилка при отриманні проєктів' });
  }
};
exports.getProjectsByUser = async (req, res) => {
  try {
    const { userId } = req.params;
    const projects = await Project.find({ userId });
    res.json(projects);
  } catch (err) {
    res.status(500).json({ error: 'Не вдалося отримати проєкти користувача' });
  }
};
exports.updateProject = async (req, res) => {
  try {
    const updated = await Project.findByIdAndUpdate(req.params.id, req.body, {
      new: true,
    });    res.json(updated);
  } catch (err) {
    res.status(500).json({ error: 'Не вдалося оновити проєкт' });
  }
};
exports.deleteProject = async (req, res) => {
  try {
    await Project.findByIdAndDelete(req.params.id);
    res.json({ success: true });
  } catch (err) {
    res.status(500).json({ error: 'Не вдалося видалити проєкт' });
  }
};

/*=================*/

  try {
    const { name, email, password } = req.body;
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ errors: { email: 'Email вже використовується' } });
    }
    const user = await User.create({ name, email, password });
    const token = createToken(user);
    res.status(201).json({
      token,
      user: {
        _id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (err) {
    const errors = handleErrors(err);
    res.status(400).json({ errors });
  }
};
exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await User.findOne({ email });
    const isMatch = user && await user.comparePassword(password);
    if (!user || !isMatch) {
      return res.status(401).json({
        errors: {
```

Продовження додатку Б

```
        email: 'Невірний email або пароль',
        password: 'Невірний email або пароль'
       }
      });
    }
    const token = createToken(user);
    res.status(200).json({
      token,
      user: {
        _id: user._id,
        name: user.name,
        email: user.email
      }
    });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Серверна помилка' });
  }
};

/*=================*/

const Project = require('../models/Project');
exports.saveProject = async (req, res) => {
  try {
    console.log("USER ID:", req.user?.id);
    console.log("REQUEST BODY:", req.body);
    const { name, room, furniture } = req.body;
    const userId = req.user.id;
    const newProject = new Project({
      userId,
      name,
      room,
      furniture,
    });
    await newProject.save();
    res.status(201).json(newProject);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Не вдалося зберегти проєкт' });
  }
};
exports.getProjects = async (req, res) => {
  try {
    const userId = req.user.id;
    const projects = await Project.find({ userId });
const User = require('../models/User');
const jwt = require('jsonwebtoken');
const createToken = (user) => {
  return jwt.sign({ id: user._id }, process.env.JWT_SECRET, { expiresIn: '7d' });
};
const handleErrors = (err) => {
  const errors = { name: '', email: '', password: '' };
  if (err.code === 11000 && err.keyPattern?.email) {
    errors.email = 'Email вже використовується';
    return errors;
  }
  if (err.message.includes('User validation failed')) {
    Object.values(err.errors).forEach(({ properties }) => {
      if (errors[properties.path] !== undefined) {
        errors[properties.path] = properties.message;
      }
    });
  }
  return errors;
};
exports.register = async (req, res) => {
const mongoose = require('mongoose');
```

Продовження додатку Б

```
const bcrypt = require('bcrypt');
const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, 'Введіть ім’я'],
    minlength: [3, 'Ім’я має містити щонайменше 3 символи'],
    trim: true,
  },
  email: {
    type: String,
    required: [true, 'Введіть email'],
    unique: true,
    lowercase: true,
    trim: true,
    match: [/.+\@.+\..+/, 'Введіть правильний email']
  },
  password: {
    type: String,
    required: [true, 'Введіть пароль'],
    minlength: [6, 'Пароль має містити щонайменше 6 символів']
  },
}, { timestamps: true });
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  this.password = await bcrypt.hash(this.password, 10);
  next();
});
userSchema.methods.comparePassword = async function (password) {
  return await bcrypt.compare(password, this.password);
};
module.exports = mongoose.model('User', userSchema);

/*=================*/

const mongoose = require('mongoose');
const ProjectSchema = new mongoose.Schema({
  userId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  name: {
    type: String,
    required: true
  },
  room: {
    width: {
      type: Number,
      required: true
    },
    height: {
      type: Number,
      required: true
    },
    color: {
      type: String,
      required: true
    },
    floorColor: {
      type: String,
      required: true
    },
    lightIntensity: {
      type: Number,
      default: 1
    }
  },
  furniture: {
```

Продовження додатку Б

```
    type: Array,
    required: true
  }
}, {
  timestamps: true
});
module.exports = mongoose.model('Project', ProjectSchema);
/*=================*/

const mongoose = require('mongoose');
const furnitureSchema = new mongoose.Schema({
  name: { type: String, required: true },
  previewImage: { type: String, required: true },
  modelPath: { type: String, required: true }
});
module.exports = mongoose.model('Furniture', furnitureSchema);

/*=================*/

const mongoose = require('mongoose');
const modelSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  filePath: {
    type: String,
    required: true,
  },
}, {
  timestamps: true,
});
module.exports = mongoose.model('Model', modelSchema);

/*=================*/

const express = require('express');
const router = express.Router();
const projectController = require('../controllers/projectController');
const authMiddleware = require('../middleware/authMiddleware');
router.post('/', authMiddleware, projectController.saveProject);
router.get('/', authMiddleware, projectController.getProjects);
router.get('/user/:userId', authMiddleware, projectController.getProjectsByUser);
router.put('/:id', authMiddleware, projectController.updateProject);
router.delete('/:id', authMiddleware, projectController.deleteProject);
module.exports = router;

/*=================*/

const express = require('express');
const router = express.Router();
const authController = require('../controllers/authController');
router.post('/register', authController.register);
router.post('/login', authController.login);
module.exports = router;

/*=================*/

const jwt = require('jsonwebtoken');
module.exports = (req, res, next) => {
  const token = req.header('Authorization')?.split(' ')[1];
  if (!token) return res.status(401).json({ message: 'Немає токена' });
  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    console.log("TOKEN PAYLOAD:", decoded);
    req.user = decoded;
    next();
  } catch (err) {
```

Продовження додатку Б

```
    res.status(401).json({ message: 'Недійсний токен' });
  }
/*=================*/
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
require('dotenv').config();

const authRoutes = require('./routes/authRoutes');
const projectRoutes = require('./routes/projectRoutes');
const furnitureRoutes = require('./routes/furnitureRoutes');

const app = express();
app.use(cors());
app.use(express.json());
app.use(express.static('public'));

mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('MongoDB connected'))
.catch((err) => console.log(err));

app.use('/api/auth', authRoutes);
app.use('/api/projects', projectRoutes);
app.use('/api/furniture', furnitureRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

/*=================*/

const seedFurniture = async () => {
  await mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  });
  const furnitureList = [
    {
      name: 'Стілець',
      previewImage: '/images/chair.png',
      modelPath: '/models/chair.obj',
    },
    {
      name: 'Стіл',
      previewImage: '/images/table.png',
      modelPath: '/models/table.obj',
    },
    {
      name: 'Диван',
      previewImage: '/images/sofa.png',
      modelPath: '/models/sofa.obj',
    },
  ];
  try {
    await Furniture.deleteMany();
    await Furniture.insertMany(furnitureList);
    console.log('✅ Меблі успішно додані!');
  } catch (err) {
    console.error('❌ Помилка додавання меблів:', err.message);
  } finally {
    await mongoose.disconnect();
    process.exit();
  }
};
seedFurniture();
```