ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка застосунку для управління офісним простором»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Валентин ТКАЧИЩЕН
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Валентин ТКАЧИЩЕН

Керівник: Ігор ГАМАНЮК
ст. викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ткачищену Валентину Михайловичу

1. Тема кваліфікаційної роботи: «Розробка застосунку для управління офісним простором»
керівник кваліфікаційної роботи старший викладач кафедри ІПЗ Ігор ГАМАНЮК, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: аналітичні матеріали щодо систем управління офісним простором, теоретичні відомості з організації робочих місць та гнучкого офісу, технічна документація з реалізації хешування паролів та управління користувачами.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд застосунку для управління офісним простором, аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації застосунку для управління офісним простором.

3. Проєктування архітектури та модулів застосунку для управління офісним простором.
4. Програмна реалізація та тестування застосунку для управління офісним простором.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Діаграма прецедентів.
   5. Бізнес модель
   6. Діаграма діяльності.
   7. Діаграма станів.
   8. Діаграма пакетів.
   9. Діаграма послідовності
   10. Діаграма класів.
   11. Екранні форми.
   12. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Формування функціональних та нефункціональних вимог до застосунку	14.03-20.03.2025	
4	Проектування застосунку для управління офісним простором	21.03-28.03.2025	
5	Програмна реалізація застосунку	29.03-18.04.2025	
6	Тестування застосунку	19.04-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Валентин ТКАЧИЩЕН
(підпис)

Керівник
кваліфікаційної роботи _____ Ігор ГАМАНЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _52_ стор., __1__ табл., _21_ рис., _20_ джерел.

Мета роботи – покращити діяльність компанії шляхом автоматизації процесу управління офісним простором за рахунок створення застосунку з управління офісним простором.

Об'єкт дослідження – процес управління офісним простором.

Предмет дослідження – програмне забезпечення для управління офісним простором.

Короткий зміст роботи: У роботі проаналізовано особливості процесу управління офісним простором, визначено основні проблеми, пов'язані з неефективним використанням приміщень та робочих місць. Проведено порівняльний аналіз програмних аналогів (Condeco, Skedda, Robin Powered, OfficeRnD Hybrid), на основі чого сформульовано вимоги до власного програмного продукту. Розроблено архітектуру та реалізовано десктопний застосунок OfficePoint, який включає: бронювання робочих місць, перегляд графіка зайнятості, облік приміщень, генерацію звітів, авторизацію з підтримкою ролей (працівник, адміністрація, HR) та локалізацію інтерфейсу. У застосунку реалізовано фільтрацію за параметрами місця, автоматичне підставлення даних при бронюванні, а також збір пропозицій щодо покращення умов офісу. Для реалізації використано C# і платформу .NET, WPF для інтерфейсу, SQLite як вбудовану базу даних, Entity Framework Core для доступу до даних. Проведено тестування функціональних можливостей.

Сферою використання застосунку є автоматизація управління робочими місцями в організаціях, що мають гібридний або гнучкий формат офісної роботи.

КЛЮЧОВІ СЛОВА: ОФІСНИЙ ПРОСТІР, БРОНЮВАННЯ, C#, .NET, WPF, SQLITE, РОЛІ, ФІЛЬТРАЦІЯ, ЛОКАЛІЗАЦІЯ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment

ORM – Object-Relational Mapping

API – Application Programming Interface

UI – User Interface

UML – Unified Modeling Language

CSV – Comma-Separated Values

EF Core – Entity Framework Core

SPA – Single Page Application

WPF – Windows Presentation Foundation

DB – Database

DTO – Data Transfer Object

ВСТУП

Актуальність: Організація робочого простору є ключовим чинником ефективності сучасних компаній, особливо в умовах переходу до гібридних і гнучких форматів праці. Неефективне використання приміщень, перевантаження адміністративного персоналу та відсутність зручних механізмів контролю за офісною зайнятістю призводять до зниження продуктивності працівників і збільшення витрат. Зі зростанням цифровізації все більше організацій впроваджують автоматизовані системи управління простором, що дозволяють оптимізувати ресурси та зробити середовище праці адаптивнішим.

Одним із найзручніших способів вирішення таких задач є використання спеціалізованих програмних засобів. Вони забезпечують централізоване керування приміщеннями, бронювання робочих місць, формування звітності та взаємодію з працівниками через зворотний зв'язок. Такі інструменти дозволяють організаціям швидко адаптувати офісну інфраструктуру до змін у навантаженні, а працівникам — легко знаходити вільне місце для роботи, орієнтуючись на власні потреби та комфорт.

За даними досліджень, близько 60% компаній з понад 500 співробітниками вже впровадили або планують впровадити автоматизовані системи управління офісним простором[1]. Це підтверджує актуальність створення доступних, локалізованих та простих у використанні рішень, адаптованих до потреб українського бізнесу.

Об'єктом дослідження є процес управління офісним простором.

Предметом дослідження є програмне забезпечення для управління офісним простором.

Метою роботи є покращити діяльність компанії шляхом автоматизації процесу управління офісним простором за рахунок створення застосунку з управління офісним простором.

Методи дослідження: На початковому етапі проведено аналіз предметної галузі та програмних аналогів (Condeco, Skedda, Robin Powered, OfficeRnD Hybrid), що дозволило сформулювати ключові функціональні та нефункціональні вимоги до майбутнього програмного продукту.

Після цього було розроблено архітектуру застосунку, визначено основні технології реалізації: C#/.NET для бізнес-логіки, WPF для інтерфейсу, SQLite як систему управління базою даних, Entity Framework Core для доступу до даних. Розробка інтерфейсу базувалася на шаблоні MVVM з підтримкою локалізації.

Під час реалізації застосунку було інтегровано механізм авторизації з ролями, перегляд графіка зайнятості, бронювання місць, формування звітів та залишення пропозицій щодо покращення умов у приміщеннях.

Наукова новизна роботи визначається наступними функціональними складовими: підтримка ролей користувачів (працівник, адміністратор, відділ кадрів); система графічного перегляду зайнятості місць; можливість залишати пропозиції щодо покращення умов приміщень.

Застосунок постає як всебічний інструмент для автоматизованого управління офісним простором, що поєднує бронювання, аналітику та зворотний зв’язок в єдиному середовищі.

Практична значущість результатів полягає в можливості використання розробленого застосунку для ефективного управління офісним простором в організаціях з гнучким або гібридним форматом робот

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ УПРАВЛІННЯ ОФІСНИМ ПРОСТОРОМ

1.1 Програмне забезпечення для управління офісним простором

Програмне забезпечення для управління офісним простором — це цифрові інструменти, які дозволяють організаціям ефективно координувати використання приміщень, робочих місць і супутніх ресурсів. Такі системи зазвичай охоплюють модулі бронювання, аналітики, управління доступом і зворотного зв’язку. Їх основна мета — оптимізувати офісну інфраструктуру, зменшити витрати, підвищити продуктивність персоналу та створити комфортні умови для роботи.

Згідно з дослідженнями, цифрове робоче середовище визначається як сукупність інструментів і віртуальних просторів взаємодії, необхідних для підтримки робочого процесу співробітників, доступу до різноманітної важливої інформації, пов'язаної з роботою, та комунікації в межах організації та з зовнішнім середовищем[2].

Однак впровадження цифрових технологій в робоче середовище має не лише переваги. Інтенсивне використання цифрових технологій може призводити до гіперзв'язності та перевантаження працівників, що негативно впливає на їхнє психічне здоров'я та добробут[3].

Основні компоненти систем управління офісним простором включають:

- Модуль бронювання: дозволяє працівникам самостійно резервувати робочі місця або переговорні кімнати.
- Аналітичний модуль: відображає рівень завантаженості приміщень, статистику використання тощо.
- Рольова система доступу: визначає, хто має право на перегляд, бронювання чи керування ресурсами.
- Зворотний зв’язок: модуль, через який працівники можуть залишати

побажання або скарги щодо умов у приміщеннях.

- Панель адміністратора: дає змогу HR-відділу або адміністрації створювати звіти та слідкувати за динамікою використання простору.

Основні цілі застосунку для управління офісним простором:

- забезпечення рівного доступу до ресурсів;
- мінімізація конфліктів щодо розміщення;
- зниження адміністративного навантаження;
- збір аналітики для прийняття управлінських рішень;
- покращення комфорту працівників.

Оцінимо переваги та недоліки таких програмних рішень.

Переваги:

- Доступність: працівники можуть керувати бронюванням і переглядати завантаженість навіть із домашнього пристрою.
- Гнучкість: можливість адаптації до змінних потреб — зменшення/збільшення кількості місць, оновлення політик доступу.
- Економія часу: скорочення часу, витраченого на адміністративне управління ресурсами.
- Аналітична підтримка: наявність вбудованих звітів і графіків дає змогу обґрунтовано змінювати планування.
- Покращення умов: через зворотний зв’язок можна швидко реагувати на запити щодо змін.

Недоліки:

- Складність використання: багато систем мають перевантажений інтерфейс або складну логіку, що ускладнює швидке освоєння базового функціоналу.
- Вимога постійного підключення до інтернету: більшість рішень працює лише онлайн, що створює проблеми за нестабільного підключення.
- Надлишковий функціонал: універсальні платформи часто перевантажені опціями, непридатними для малих і середніх організацій.
- Відсутність підтримки української мови: більшість популярних

рішень не локалізовані українською, що ускладнює використання в державних установах, освітніх закладах чи серед внутрішніх команд без високого рівня володіння англійською.

1.2 Специфіка управління офісним простором

Управління офісним простором — це багатогранна діяльність, що передбачає координацію фізичного розміщення працівників, планування площ, управління ресурсами та забезпечення комфортного робочого середовища. У традиційній моделі офісу ці завдання виконувалися вручну або за допомогою базових офісних інструментів, однак у сучасному середовищі такі підходи виявилися недостатніми.

Особливої актуальності питання ефективного управління простором набуло після переходу багатьох компаній до гібридного формату роботи. Коли частина персоналу працює дистанційно, а інша — в офісі, постає потреба в гнучкому бронюванні робочих місць, зон для співпраці, переговорних кімнат тощо. Це вимагає впровадження цифрових рішень, що дають змогу не лише контролювати доступність ресурсів у режимі реального часу, а й аналізувати дані про завантаженість, виявляти неефективне використання площ та планувати зміни у розміщенні персоналу.

Згідно з дослідженням, проведеним у 2023 році, близько 60% компаній у країнах ЄС вже впровадили або планують впровадити системи для цифрового управління офісним простором[4]. Такі системи дозволяють керівникам оптимізувати використання нерухомості, зменшити витрати на оренду та обслуговування приміщень, а також формувати більш адаптивне середовище праці, орієнтоване на потреби працівників.

Управління офісним простором охоплює такі ключові компоненти:

- Планування простору — визначення кількості та типу робочих місць, зон спільної роботи, відпочинку, технічних приміщень тощо. На цьому етапі важливо враховувати специфіку роботи підрозділів, потреби у взаємодії

між командами та дотримання норм щільності.

- Облік та інвентаризація ресурсів — створення актуальної бази даних щодо наявних приміщень, їх площі, типів робочих місць, обладнання. Це забезпечує прозорість та швидкий доступ до інформації для прийняття рішень.
- Бронювання — одна з найбільш затребуваних функцій у гібридному середовищі. Вона дозволяє працівникам заздалегідь резервувати робочі місця, переговорні або інші зони, що зменшує конфлікти за ресурси та покращує організацію робочого часу.
- Аналіз використання простору — збір і обробка статистики щодо завантаженості приміщень у різні дні, години, сезони. Це дозволяє виявити недостатньо використовувані зони, прогнозувати пікові навантаження і приймати рішення щодо перепланування.
- Підтримка гнучкості — можливість оперативно змінювати конфігурацію робочих місць, переміщати меблі, створювати тимчасові зони або адаптуватися до зростання штату без потреби у переїзді.

Окремим викликом є необхідність адаптації офісних рішень до нормативних вимог, стандартів безбар'єрності, а також до культурних та мовних очікувань користувачів. Зокрема, у країнах Східної Європи поширеною проблемою залишається відсутність локалізації інтерфейсів цифрових систем, що ускладнює їхнє впровадження в державних установах або освітніх закладах.

Крім того, впровадження цифрових технологій в управління офісним простором може позитивно впливати на ефективність офісного адміністрування. Дослідження показують, що цифрова трансформація сприяє підвищенню продуктивності, автоматизації процесів та покращенню комунікації в організаціях[5].

1.3 Цільова аудиторія

Цільова аудиторія застосунку для управління офісним простором охоплює користувачів, які безпосередньо взаємодіють із приміщеннями або відповідають

за їх адміністрування. Врахування специфіки кожної ролі є важливим для побудови логіки доступу та користувацького інтерфейсу.

Ключові групи користувачів:

- Працівники. Основні користувачі системи, які бронюють робочі місця, переглядають їх доступність та можуть залишати пропозиції щодо покращення офісного простору. Для них важливі зручність, швидкість та локалізація інтерфейсу.
- Адміністрація. Має розширений доступ для керування даними про приміщення та робочі місця, а також для формування звітів і моніторингу використання ресурсів.
- Відділ кадрів. Здійснює реєстрацію нових працівників у системі та стежить за відповідністю офісного простору кадровій структурі компанії.
- Керівництво. Використовує агреговану статистику для прийняття рішень щодо оптимізації простору, бюджету або реконфігурації приміщень.

Таке розділення дозволяє системі забезпечити ефективну, безпечну та персоналізовану взаємодію кожної ролі з інтерфейсом відповідно до її повноважень та потреб.

1.4 Дослідження існуючих аналогів

На сьогодні існує низка готових рішень для управління офісним простором, які дозволяють здійснювати бронювання робочих місць, аналітику завантаженості приміщень та управління ресурсами компанії. Серед найбільш відомих аналогів можна виділити Condeco[6], Skedda[7], Robin Powered[8] та OfficeRnD Hybrid[9]. Розглянемо їх функціональні можливості, переваги та обмеження детальніше.

1.4.1 Condeco

Condeco — це корпоративний онлайн-застосунок для управління офісним простором, що дозволяє здійснювати бронювання робочих місць, переговорних

кімнат та інших офісних ресурсів. Система розроблена з орієнтацією на великі компанії та спрямована на підвищення ефективності використання офісної інфраструктури. Основна ідея полягає в забезпеченні централізованого контролю над простором, автоматизації процесу бронювання і підтримці гнучких сценаріїв роботи.

Основні функції Condeco включають:

- Бронювання ресурсів: робочих місць, кімнат, обладнання через вебінтерфейс або мобільний застосунок.
- Інтеграція з календарями: синхронізація з Microsoft Outlook, Google Calendar та іншими сервісами.
- Аналітика простору: відображення статистики завантаженості, формування звітів для оптимізації використання.
- Ролі доступу: налаштування прав для різних груп користувачів.
- Мобільна підтримка: застосунок доступний на iOS та Android.
- Гнучке конфігурування: можливість задавати правила бронювання відповідно до політик компанії.
- Підтримка гібридного формату роботи: забезпечення прозорого планування присутності працівників в офісі.
- Інтерфейс для адміністраторів: контроль за всім простором із можливістю швидкого перегляду зайнятості.
- Вбудовані нагадування: автоматичні повідомлення про майбутні бронювання або конфлікти.
- Сумісність із корпоративними середовищами: підтримка авторизації через SSO та корпоративні каталоги.

Переваги:

- Широкий спектр функцій для управління офісним простором.
- Підтримка великих корпоративних структур і гнучких сценаріїв роботи.
- Глибока інтеграція з поширеними календарними сервісами.
- Потужна аналітика для прийняття управлінських рішень.

- Мобільність і доступність на різних пристроях.

Недоліки:

- Висока вартість впровадження і підтримки.
- Складна початкова конфігурація та потреба в навчанні персоналу.
- Відсутність української локалізації, що ускладнює використання в локальному середовищі
- Надмірна кількість функцій для невеликих організацій

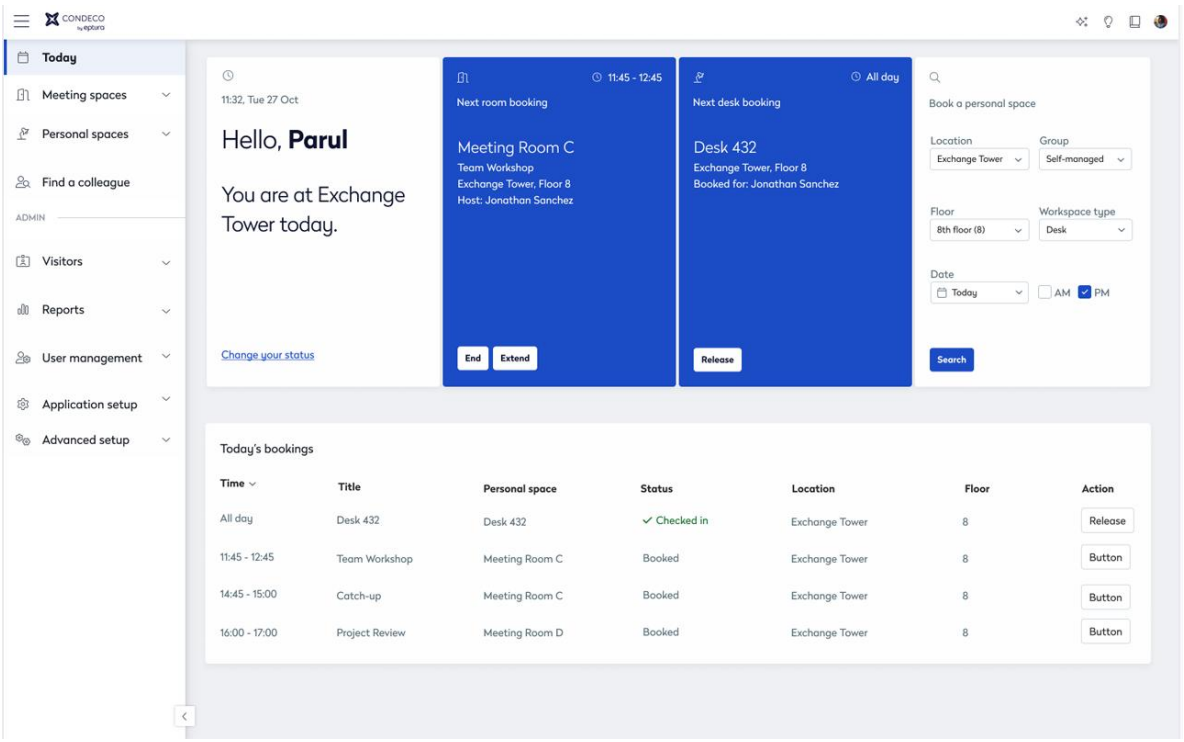


Рис. 1.1 Скріншот інтерфейсу Condeco

1.4.2 Skedda

Skedda — це хмарний застосунок для управління доступом до робочих просторів, призначений для організацій різного масштабу: від корпоративних офісів до коворкінгів, освітніх закладів і некомерційних структур. Основна ідея полягає в забезпеченні гнучкого, але простого у використанні інструменту для адміністрування приміщень, із можливістю налаштування доступу за ролями, графіками й політиками.

Основні функції Skedda включають:

- Бронювання ресурсів: можливість онлайн-бронювання приміщень і робочих місць через веб-інтерфейс.
- Налаштування правил доступу: визначення прав доступу залежно від типу користувача, часу доби чи дня тижня.
- Інтеграція з календарними сервісами: підтримка синхронізації з Microsoft 365, Google Workspace.
- Автоматизація за допомогою Zapier: інтеграція з іншими платформами для спрощення робочих процесів.
- Оплата бронювань: підтримка онлайн-оплат (Stripe), що може бути корисним для коворкінгів та подібних середовищ.
- Гнучке керування користувачами: визначення ролей, груп і дозволів для організаційного адміністрування.
- Веб-інтерфейс: немає потреби в окремій установці застосунку — доступ через браузер із будь-якого пристрою.

Переваги:

- Простота використання з високим ступенем кастомізації.
- Інтеграція з Microsoft 365, Google Calendar, Zapier.
- Можливість прийому онлайн-оплат без додаткового кодування.
- Гнучка система правил для доступу та резервування.
- Оптимізований під браузери — не потребує встановлення.

Недоліки:

- Відсутність мобільного застосунку — обмеження зручності для мобільних користувачів.
- Обмежена аналітика — базова звітність, без глибокого аналізу використання.
- Більшість функцій доступні лише в платних тарифах.
- Відсутність української локалізації — лише англійський інтерфейс.

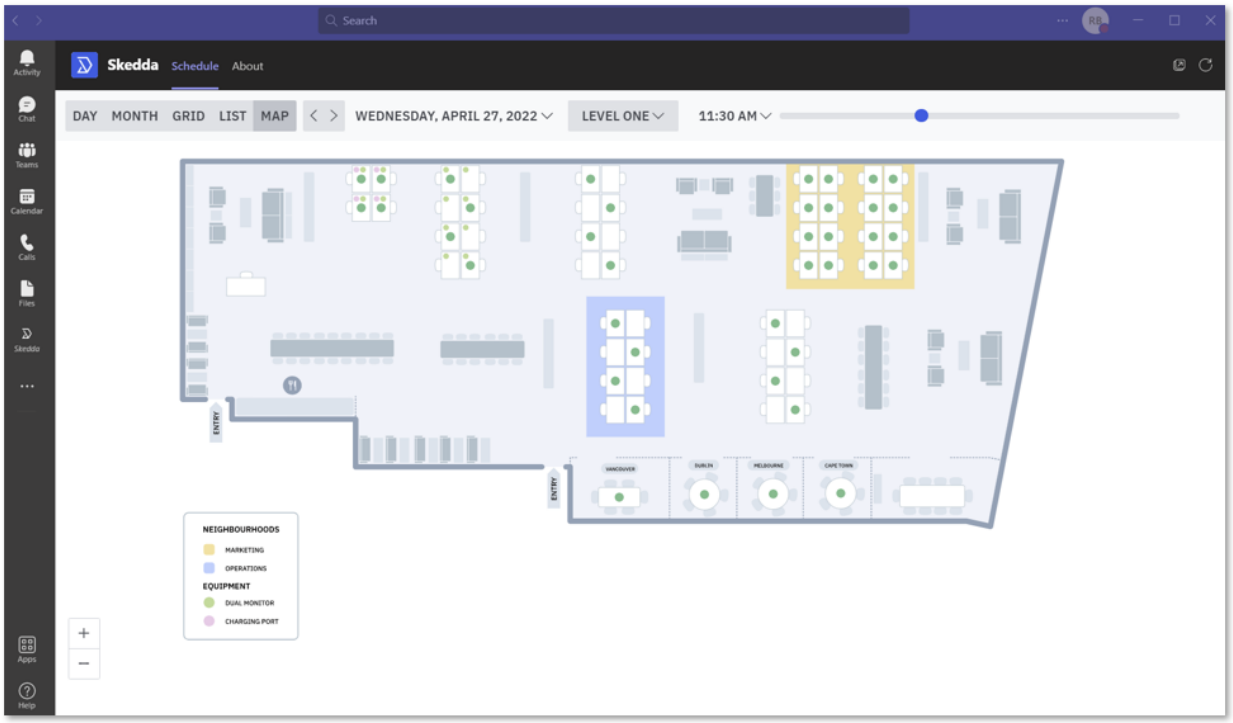


Рис. 1.2 Скріншот інтерфейсу Skedda

1.4.3 Robin Powered

Robin Powered — це багатофункціональний застосунок для управління гібридним офісом, який дозволяє організаціям координувати присутність працівників, бронювання робочих місць і переговорних кімнат, а також отримувати аналітичну інформацію про використання простору. Основна концепція полягає в створенні гнучкого офісного середовища, яке відповідає сучасним вимогам змішаної роботи (remote + office).

Основні функції Robin Powered включають:

- Бронювання робочих місць та кімнат: резервування через веб або мобільний застосунок із можливістю перегляду доступності в реальному часі.
- Координація присутності: система планування відвідувань, що дозволяє командам бачити, хто працює з офісу в конкретний день.
- Інтеграція з календарями: сумісність із Google Calendar та Microsoft Outlook.

- Мобільний застосунок: доступ до всіх функцій з iOS та Android-пристроїв.
- Панель адміністратора: налаштування політик бронювання, управління користувачами та просторами.
- Аналітика офісного використання: звіти щодо активності працівників, завантаженості робочих зон та тенденцій.

Переваги:

- Підтримка гібридного формату роботи з акцентом на координацію команд.
- Повнофункціональний мобільний застосунок.
- Сучасний та інтуїтивний інтерфейс.
- Аналітика та інструменти оптимізації використання простору.
- Гнучкі правила доступу й бронювання.

Недоліки:

- Висока вартість при використанні всіх функцій.
- Обмеження для малих організацій, яким не потрібна складна аналітика.
- Відсутність локалізації українською мовою.
- Обмежена підтримка користувацьких налаштувань у базових тарифах.

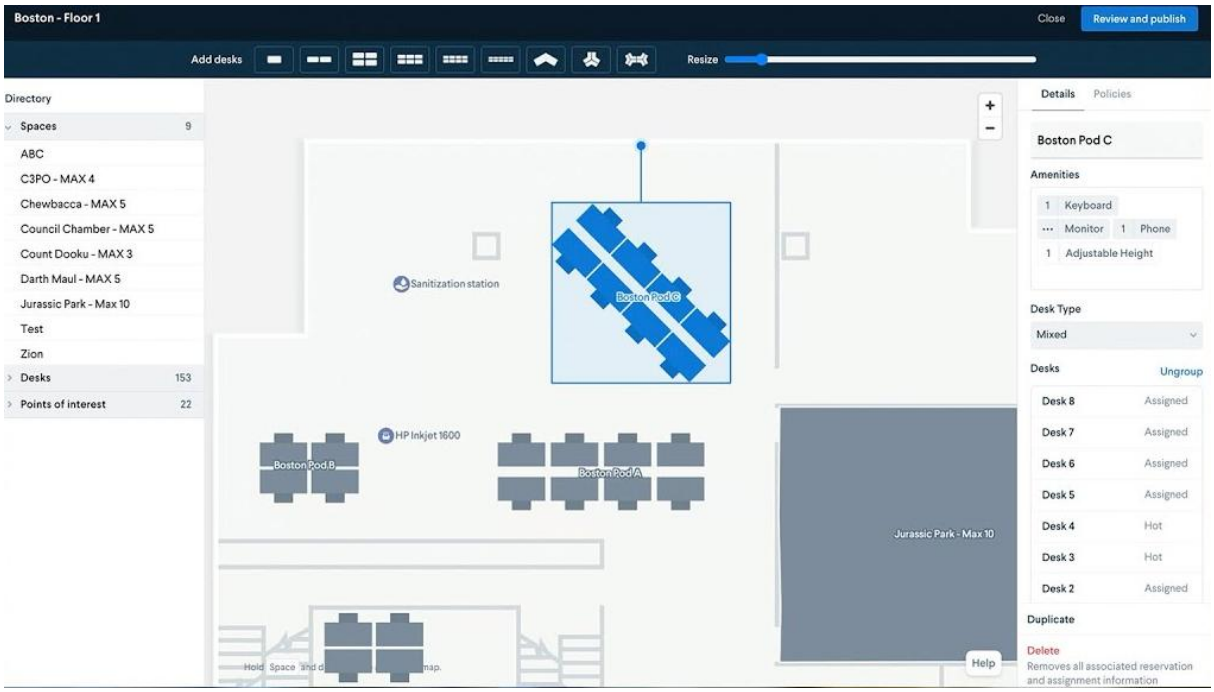


Рис. 1.3 Скріншот інтерфейсу Robin Powered

1.4.4 OfficeRnD Hybrid

OfficeRnD Hybrid — це корпоративна платформа для управління гібридним офісом, орієнтована на організації, які впроваджують гнучкий графік роботи та розподілені команди. Система дозволяє працівникам бронювати робочі місця й кімнати для зустрічей, а адміністраторам — керувати простором, контролювати доступ, формувати звітність і приймати рішення на основі аналітики.

Основні функції OfficeRnD Hybrid включають:

- Планування відвідувань офісу: користувачі можуть обирати дні присутності та синхронізувати графіки з колегами.
- Бронювання ресурсів: резервування робочих місць і переговорних кімнат через веб або мобільний застосунок.
- Інтеграція з Microsoft Teams, Outlook, Google Calendar: синхронізація з наявними корпоративними сервісами.
- Scheduling Alignment: функція вирівнювання графіків для командної присутності в офісі.

- Аналітика і звітність: інструменти для моніторингу завантаженості офісу, поведінки користувачів та оптимізації використання простору.
- Гнучке налаштування політик: адміністративна панель дозволяє задавати права доступу до робочих зон, обмеження за часом чи категоріями користувачів.
- Мобільна підтримка: застосунок доступний для iOS і Android, що забезпечує мобільність користувачів.

Переваги:

- Функціонал для узгодження присутності команд (Scheduling Alignment).
- Повна інтеграція з календарями та Microsoft Teams.
- Аналітичні звіти для прийняття управлінських рішень.
- Висока гнучкість у налаштуванні політик бронювання.
- Мобільний доступ до всіх функцій системи.

Недоліки:

- Висока вартість для малого бізнесу.
- Складність первинного налаштування політик і доступів.
- Відсутність української мови інтерфейсу.
- Зосередженість на середньому і великому бізнесі, що обмежує доступність для невеликих команд.

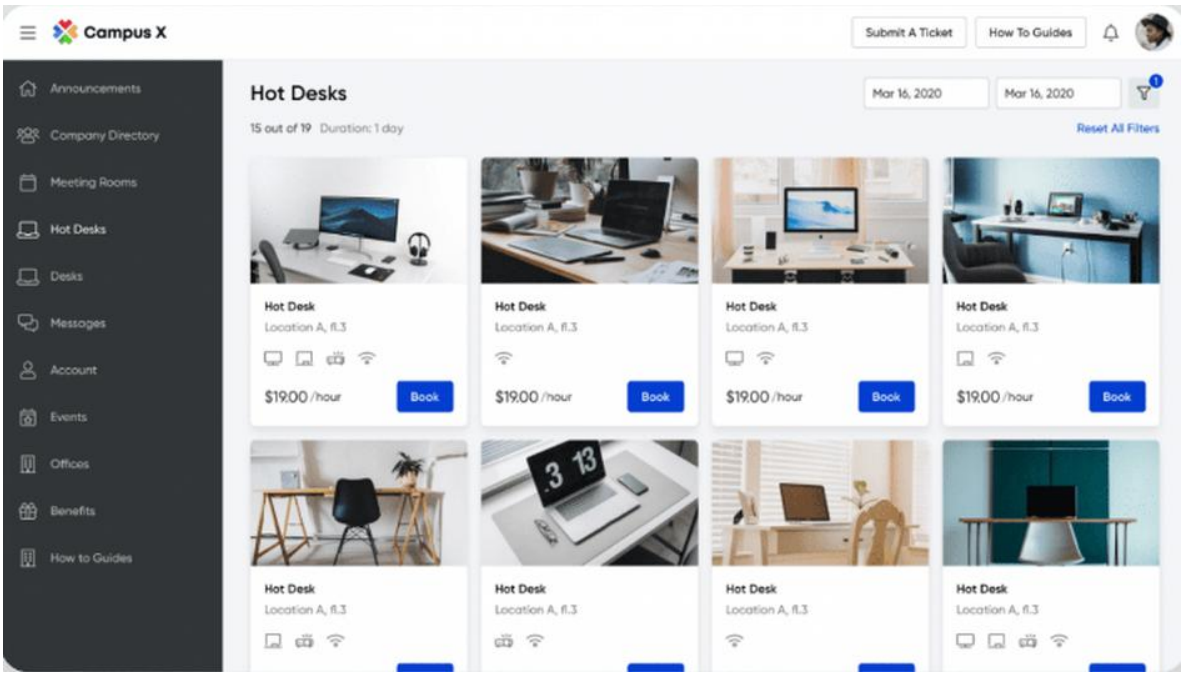


Рис. 1.4 Скріншот інтерфейсу OfficeRnD Hybrid

В таблиці 1.1 зведено результати аналізу існуючих застосунків для управління офісним простором.

Таблиця 1.1

Зведена таблиця аналізу існуючих застосунків для управління офісним простором та їх порівняння

Показник	**Condeco**	**Skedda**	**Robin Powered**	**OfficeRnD Hybrid**
Бронювання робочих місць	Підтримує робочі місця, переговорні, обладнання	Гнучке керування розкладом	Гнучке бронювання, інтеграція із планами приміщень	Широкі можливості бронювання, власна мобільна аплікація
Аналітика	Розширена, з діаграмами та KPI	Базова статистика	Статистика на основі IoT-сенсорів	Потужна аналітика використання офісного простору
Інтеграція з календарями	Microsoft Outlook, Google Calendar	Google Calendar	Microsoft Teams, Google, Slack	Google, Outlook, Microsoft Teams
Ролі користувачів	Гнучке управління правами доступу	Права адміністрування	Ролі на основі організації та команд	Ролі користувачів, права на бронювання

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих застосунків для управління офісним простором та їх порівняння

Показник	Condeco	Skedda	Robin Powered	OfficeRnD Hybrid
Мобільний доступ	iOS, Android	Через веб-інтерфейс	iOS, Android	Власний застосунок для мобільних платформ
Локалізація	Англійська, інші мови, без української	Англійська	Англійська, обмежена локалізація	Англійська, локалізація обмежена
Цільова аудиторія	Великі корпорації з розподіленими офісами	Малі та середні бізнеси, коворкінги	Команди, що працюють гібридно	Гнучкі команди, простори з плаваючими робочими місцями
Особливості	Інтеграція з системами доступу, аналітика, бронювання ресурсів	Простота та легкість у налаштуванні, безкоштовний план	Підтримка сенсорів, візуалізація офісного простору	Повна система управління гібридним простором
Недоліки	Висока вартість, складне впровадження, англійська мова	Обмежені налаштування, відсутність сенсорної інтеграції	Деякі функції доступні лише при наявності сенсорів	Висока вартість, надмірна складність для невеликих команд

1.3 Визначення вимог до застосунку

Проаналізувавши особливості предметної галузі, цільову аудиторію та функціонал існуючих аналогів, було визначено ключові вимоги до застосунку для управління офісним простором:

Функціональні вимоги:

- Реєстрація та авторизація користувачів з поділом на ролі (працівник, відділ кадрів, адміністрація).
- Управління інформацією про працівників: додавання, редагування, перегляд і видалення даних.
- Ведення реєстру приміщень: збереження параметрів, типів і площ офісних кімнат.

- Робота з робочими місцями: додавання, редагування та видалення, прив’язка до конкретного приміщення.
- Система бронювання робочих місць із перевіркою доступності.
- Перегляд графіку зайнятості із можливістю фільтрації за приміщенням, робочим місцем та датою.
- Формування звітів про використання робочих місць, динаміку бронювань, активність працівників.
- Можливість працівникам залишати побажання та пропозиції щодо приміщень.
- Панель адміністрування для управління основними сутностями системи.

Нефункціональні вимоги:

- Інтерфейс повинен бути адаптивним і зрозумілим для користувачів з базовими навичками комп’ютерної грамотності.
- Підтримка двомовного інтерфейсу (українська, англійська).
- Система має забезпечувати одночасну роботу великої кількості користувачів (до 1000 осіб).
- Захист доступу до даних через систему авторизації з ролями.
- Забезпечення швидкодії: основні операції (перегляд, фільтрація, бронювання) не повинні перевищувати 2 секунд.
- Підтримка роботи на сучасних версіях ОС Windows.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) — це програмне середовище, яке об'єднує в собі всі основні інструменти, необхідні для написання, налагодження, тестування та підтримки програмного забезпечення. Таке середовище дозволяє розробнику працювати ефективніше, спрощуючи доступ до інструментів компіляції, автодоповнення коду, керування пакетами, інтеграції з системами контролю версій тощо. Більшість сучасних IDE також підтримують інтерактивне середовище розробки інтерфейсів користувача, що є особливо корисним при створенні десктопних застосунків.

Microsoft Visual Studio — сучасне інтегроване середовище розробки, яке забезпечує повну підтримку C#, .NET та технології WPF[10]. Visual Studio надає зручний редактор коду, потужний інструментарій для налагодження, можливості профілювання, перевірку коду в реальному часі, а також інтеграцію з системами контролю версій, такими як Git. Завдяки наявності вбудованого XAML-дизайнера, розробка інтерфейсу користувача стає зручною та наочною.

Visual Studio підтримує розробку багаторівневих десктопних додатків, що використовують шаблон MVVM (Model-View-ViewModel), і містить інструменти для створення баз даних, модулів обробки даних та UI одночасно. Для полегшення інтеграції між шарами застосунку активно використовуються можливості .NET tooling та система IntelliSense.

Крім того, Visual Studio пропонує такі функціональні можливості:

- повноцінна інтеграція з .NET SDK і WPF;
- потужна система підсвічування синтаксису та рефакторингу;
- підтримка Entity Framework Core;
- відлагодження баз даних SQLite прямо з IDE;
- тестування за допомогою MSTest і xUnit;

- локалізація інтерфейсу через ресурси .resx;
- створення підписаних збірок і інсталяторів.

Середовище дозволяє ефективно організувати розробку командного проєкту, використовуючи репозиторії GitHub, управління задачами (Tasks), гілками (Branches) і запитами на зміну коду (Pull Requests).

2.2 Мова програмування

Мова програмування — це формалізований набір синтаксичних правил, призначений для написання програм, які можуть бути виконані комп'ютером. Вибір мови визначає архітектурний стиль, швидкість розробки, зручність підтримки, а також сумісність із цільовою платформою. Мови високого рівня дозволяють писати читабельний та масштабований код, використовуючи об'єктно-орієнтований, процедурний або функціональний підхід. Для розробки десктопних застосунків із графічним інтерфейсом часто обираються строго типізовані мови з підтримкою шаблонів проєктування та доступом до потужних API.

C# — сучасна об'єктно-орієнтована мова програмування, яка є частиною платформи .NET[11]. Її синтаксис є зручним, виразним та інтуїтивно зрозумілим, що значно пришвидшує процес написання та підтримки коду. Завдяки суворій типізації, C# забезпечує високу стабільність програмного забезпечення, дозволяючи уникати численних помилок ще на етапі компіляції.

C# є ідеальним вибором для створення настільних застосунків, особливо у поєднанні з технологією WPF, що забезпечує потужні можливості для побудови інтерфейсів. Платформа .NET підтримує розробку в архітектурі MVVM, яка сприяє чистому розділенню логіки, даних та представлення, що важливо для підтримуваності та масштабованості проєкту[12].

До переваг використання C# можна віднести:

- Підтримку LINQ для запитів до колекцій та баз даних;
- Інтеграцію з Entity Framework, яка дозволяє працювати з базою даних у вигляді об'єктів;
- Асинхронне програмування (async/await), що забезпечує високу чуйність інтерфейсу;
- Серйозну підтримку локалізації засобами ресурсних файлів (.resx);
- Систему атрибутів і подій, яка дозволяє гнучко управляти поведінкою елементів;
- Безпечну роботу з пам’яттю завдяки автоматичному збиранню сміття (Garbage Collector).

C# також добре підтримується у Visual Studio, де розробники мають доступ до системи підказок IntelliSense, статичного аналізу коду, інструментів профілювання продуктивності та розширених можливостей налагодження.

2.3 Фреймворк розробки інтерфейсу

Windows Presentation Foundation (WPF) — потужний графічний фреймворк від Microsoft, призначений для розробки сучасних настільних застосунків на платформі .NET[13]. WPF поєднує можливості декларативного програмування за допомогою мови XAML з об’єктно-орієнтованими підходами C#, що дозволяє ефективно будувати гнучкі, масштабовані та естетично привабливі інтерфейси.

Однією з ключових переваг WPF є його підтримка шаблонів і стилів, які дають змогу створювати уніфікований дизайн для всіх візуальних компонентів. Це особливо корисно для корпоративних застосунків, які мають зберігати єдину візуальну ідентичність. Крім того, WPF підтримує двостороннє зв’язування даних (data binding), що спрощує синхронізацію інтерфейсу з бізнес-логікою застосунку.

Фреймворк також забезпечує:

- Механізми ресурсів (.resx, .xaml) для локалізації інтерфейсу;
- Тригери та анімації, що покращують взаємодію з користувачем;
- Інструменти валідації форм введення;
- Підтримку шаблону MVVM (Model-View-ViewModel) — стандартного підходу до побудови WPF-застосунків, що дозволяє ефективно відокремити логіку від представлення;
- Можливість побудови адаптивного інтерфейсу для різних розмірів екранів і вікон.

2.4 Система управління базами даних

Система управління базами даних (СУБД) — це програмне забезпечення, яке забезпечує створення, зберігання, модифікацію та доступ до структурованих даних. Вона дозволяє виконувати запити, формувати зв’язки між таблицями, забезпечувати цілісність даних і підтримувати транзакції. СУБД бувають серверні та вбудовані: перші використовуються у великих розподілених системах, другі — в автономних застосунках, де потрібна легкість, автономність і висока продуктивність без потреби окремого серверного середовища.

SQLite — легковага вбудована реляційна система управління базами даних, яка ідеально підходить для десктопних застосунків малого та середнього масштабу[14]. Її особливість полягає в тому, що база даних зберігається у вигляді одного файлу на диску, не потребує окремого серверного ПЗ і забезпечує високу продуктивність при мінімальних апаратних вимогах.

У контексті застосунку для управління офісним простором, SQLite зручно використовувати для зберігання даних про:

- працівників та облікові записи;
- приміщення, їх типи та площі;
- робочі місця з можливістю фільтрації за доступністю;
- записи бронювань;

- залишені працівниками пропозиції;
- дані авторизації та ролей користувачів.

СУБД забезпечує повну підтримку стандарту SQL, дозволяє будувати складні запити, зв’язувати таблиці зовнішніми ключами, створювати індекси для пришвидшення пошуку.

Переваги SQLite:

- не потребує розгортання серверної інфраструктури;
- швидка інтеграція з .NET і WPF;
- чудово працює в поєднанні з локальними користувацькими даними;
- відкритий код і активна підтримка.

2.5 Система контролю версій

Система контролю версій (СКВ) є ключовим інструментом у процесі створення програмного забезпечення. Вона дозволяє розробникам ефективно відслідковувати зміни у вихідному коді, керувати різними версіями програми, працювати в команді та зменшувати ризики втрати даних.

Однією з найпоширеніших і найпотужніших систем контролю версій є Git — розподілена СКВ з відкритим кодом, яка дозволяє кожному розробнику зберігати повну копію історії проєкту[15]. Git підтримує створення окремих гілок (branches) для різних функціональних модулів або експериментів, забезпечує механізми об'єднання змін (merge) та відкату до попередніх версій (revert, reset).

У поєднанні з Git активно використовується GitHub — онлайн-платформа для хостингу репозиторіїв, яка розширює можливості Git і забезпечує:

- централізоване зберігання коду;
- зручний інтерфейс для перегляду змін;
- управління запитами на злиття (pull requests);
- інструменти командної співпраці, такі як система задач (issues) і wiki-документація;

- автоматизацію CI/CD-процесів через GitHub Actions.

Завдяки GitHub, реалізація циклу розробки, тестування і релізу може бути організована за допомогою розгалуженої структури гілок (наприклад, main, develop, feature, hotfix), що відповідає сучасним підходам до DevOps та Agile-розробки[16].

Використання Git та GitHub значно підвищує ефективність, відтворюваність та прозорість процесу розробки програмного забезпечення, навіть у невеликих командах або при індивідуальній роботі.

2.6 Засоби доступу до бази даних (ORM)

Object-Relational Mapping (ORM) — це підхід до роботи з базами даних, який дозволяє розробникам взаємодіяти з таблицями як з об’єктами мови програмування, замість написання сирих SQL-запитів. Це значно спрощує розробку, покращує читабельність коду, зменшує ризик помилок та дозволяє реалізувати принципи інкапсуляції при роботі з даними.

У середовищі .NET широко використовується *Entity Framework Core* (EF Core) — сучасна ORM-бібліотека з відкритим кодом, що підтримується Microsoft[17]. Вона дозволяє:

- автоматично створювати структуру таблиць із C#-класів (Code First);
- виконувати CRUD-операції (create, read, update, delete) через LINQ-запити;
- формувати складні SQL-запити без написання SQL;
- використовувати міграції для контролю версій схеми БД;
- підтримувати зв’язки між сутностями (один до одного, один до багатьох тощо);
- забезпечувати lazy loading або eager loading при потребі.

EF Core особливо зручний у поєднанні з SQLite, оскільки не вимагає ручного керування підключенням або трансформацією даних у DTO-об’єкти.

2.7 Засоби для експорту даних у документи

Експорт даних у зовнішні формати є важливою функцією сучасного програмного забезпечення, особливо у сферах, де потрібна звітність, обмін інформацією або подальша обробка даних. Одним із найпоширеніших форматів для таких цілей є Microsoft Excel (.xlsx), який підтримується більшістю офісних систем і дозволяє представити дані у табличному вигляді з форматуванням, заголовками та фільтрами.

Для реалізації експорту у формат Excel у .NET-середовищі широко застосовується бібліотека *ClosedXML* — зручна надбудова над OpenXML SDK, яка дозволяє створювати, читати і редагувати .xlsx файли без потреби встановлення Excel або іншого стороннього ПЗ[18].

ClosedXML підтримує:

- створення таблиць із даних будь-якої структури;
- налаштування форматування (шрифт, колір, обрамлення, ширина стовпців);
- збереження кількох аркушів в одному файлі;
- додавання фільтрів, підсумкових рядків і сортування;
- встановлення заголовків і автоматичне розміщення даних у стовпцях;
- збереження файлу на диск або передача як потік (stream).

Завдяки простому API, бібліотека дозволяє створити звіт буквально в кілька рядків коду, що робить її зручною для інтеграції в бізнес-логіку застосунків різного типу.

2.8 Засоби тестування програмного забезпечення

Тестування програмного забезпечення є невід’ємною частиною розробки, що забезпечує контроль якості, перевірку відповідності логіки реалізації функціональним вимогам та зменшення кількості дефектів у кінцевому продукті.

Особливо важливим є модульне тестування (unit testing) — метод перевірки окремих функцій, класів або компонентів у відриві від решти системи.

У середовищі .NET одним із найпопулярніших інструментів для модульного тестування є *xUnit.net* — сучасна, розширювана і швидка бібліотека для створення автотестів. Вона підтримується .NET Foundation та активно використовується в open-source і корпоративних проєктах[19].

Основні особливості xUnit:

- атрибутна система визначення тестів [Fact], [Theory] тощо;
- простий і лаконічний синтаксис для створення тестових методів;
- підтримка тестових параметрів та даних для теорій;
- можливість виконання асинхронних тестів;
- підтримка життєвого циклу тестів (setup/teardown через конструктори і IDisposable);
- інтеграція з Visual Studio, GitHub Actions, Azure Pipelines та іншими CI/CD-середовищами.

xUnit дозволяє побудувати набір автоматичних перевірок для ключових компонентів проєкту: логіки обробки бронювань, перевірки доступності місць, формування звітів, авторизації та інших частин бізнес-логіки.

2.9 Засоби локалізації інтерфейсу

Локалізація програмного забезпечення — це процес адаптації інтерфейсу та вмісту програми до різних мов та культурних стандартів користувачів. У десктопних застосунках на платформі .NET широке застосування має ресурсна модель локалізації, заснована на використанні .resx-файлів (Resource Files), що дозволяють зберігати перекладені строки, а також графіку, повідомлення про помилки, тексти кнопок тощо.

У межах середовища WPF, локалізація реалізується через:

- створення окремих ресурсних файлів для кожної мови;

- використання ключів (ім’я ресурсу), прив’язаних до відповідних елементів інтерфейсу;
- підключення ресурсів за допомогою механізму ResourceManager;
- динамічне перемикання мови інтерфейсу під час виконання застосунку;
- збереження обраної мови користувача в налаштуваннях програми.
- Переваги такого підходу:
- повна підтримка двомовності або багатомовності без дублювання UI-логіки;
- зручність оновлення перекладів без зміни коду;
- можливість інтеграції з зовнішніми системами перекладу або автогенерації ресурсів;
- підтримка культурно-чутливого форматування чисел, дат, валют тощо.

У поєднанні з шаблоном MVVM, локалізація реалізується централізовано через ViewModel або окремі сервіси локалізації, що забезпечує повторне використання ресурсу у всіх частинах застосунку.

3 ПРОЄКТУВАННЯ

3.1 Аналіз функціональності системи

Функціональні можливості системи відображають основні сценарії використання, що виникають під час взаємодії користувачів із застосунком. Для формалізації та узгодження вимог проєкту було використано діаграму прецедентів, яка демонструє ролі користувачів та пов’язані з ними дії в межах системи.

У межах системи розрізняють три основні типи користувачів: працівники, працівники відділу кадрів та адміністративно-господарського відділу. Кожна з цих ролей має власний набір функцій, необхідних для виконання службових обов’язків або персональної взаємодії з офісною інфраструктурою. Працівники можуть переглядати графік зайнятості, бронювати робочі місця, а також керувати власними уподобаннями щодо умов праці. Відділ кадрів здійснює облік та керування персоналом, а адміністративний підрозділ оперує приміщеннями та робочими місцями, проводить аналітику та приймає рішення щодо розміщення ресурсів.

Діаграма прецедентів дозволяє наочно показати взаємозв’язки між ролями користувачів і системою. Такий підхід є ефективним інструментом моделювання початкової функціональності, забезпечуючи чітке розуміння вимог до системи та полегшуючи подальше проєктування архітектури.

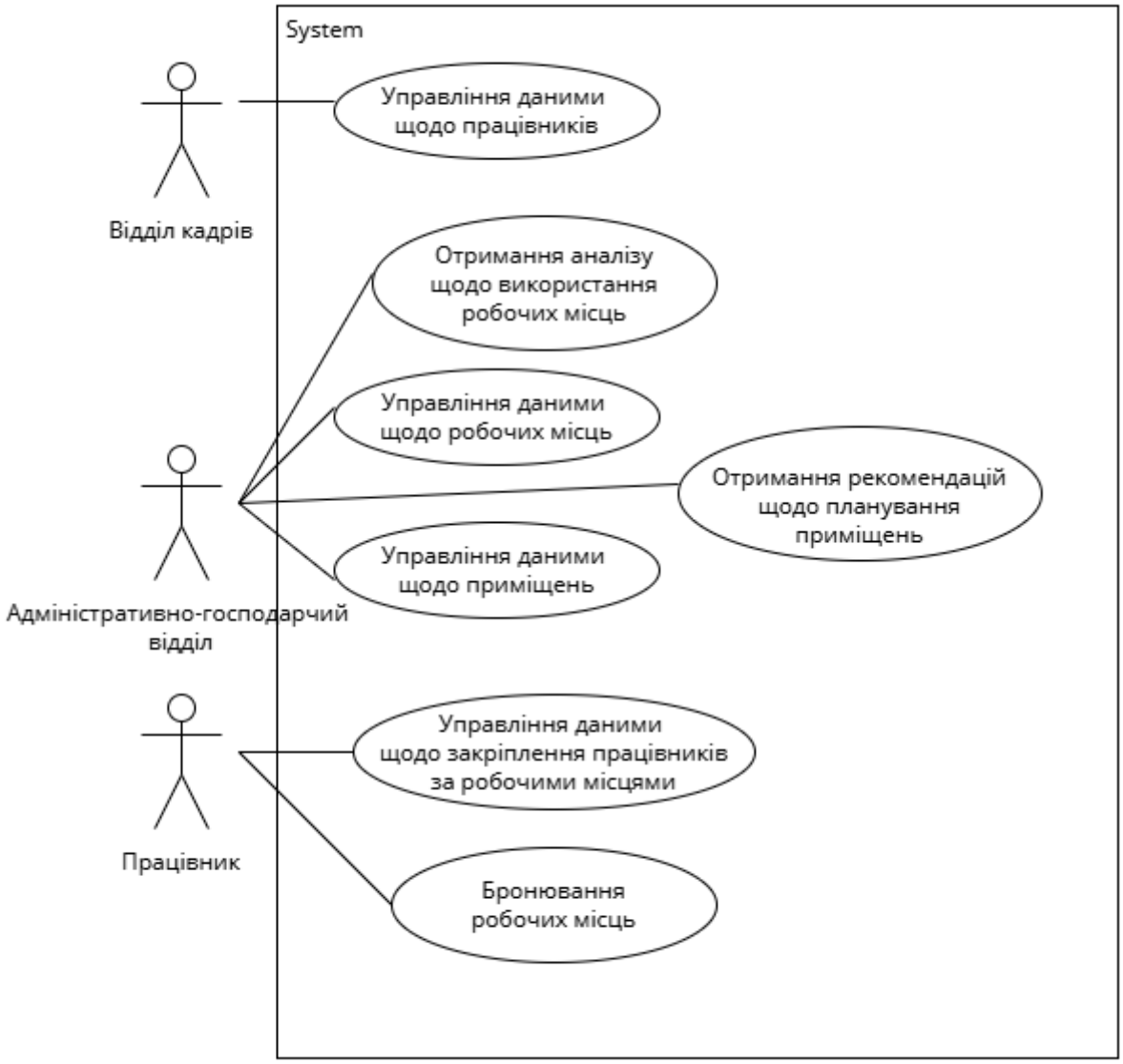


Рис. 3.1 Діаграма прецедентів системи управління офісним простором

Використання діаграм прецедентів у процесі моделювання програмного забезпечення дозволяє формалізувати взаємодію між користувачами та системою ще до етапу реалізації, знижуючи ризики непорозуміння між розробниками, аналітиками та зацікавленими сторонами. UML-діаграми, зокрема діаграми прецедентів, залишаються базовим засобом візуалізації функціональних вимог і відіграють ключову роль у ранніх стадіях розробки програмного забезпечення[20].

3.2 Моделювання предметної області

Для представлення ключових об'єктів системи та зв'язків між ними було використано діаграму предметної галузі. Такий тип моделі дає змогу візуалізувати сутності, з якими працює система, а також їхні атрибути, типи зв'язків та ролі у бізнес-логіці застосунку.

У центрі моделі розташовано клас WorkPlace, що представляє робоче місце. Кожне робоче місце має унікальний ідентифікатор, тип, площу, а також належить певному приміщенню (Room). У свою чергу, Room містить інформацію про назву, тип і площу приміщення. Працівники (Employee) можуть бути прив'язані до робочих місць — цей зв'язок реалізується як багато-до-багатьох (оскільки кілька працівників можуть змінюватися на одному робочому місці у різні години або дні).

Окремим об'єктом є Analysis, що використовується для збору аналітичних даних про використання простору — початку, завершення та дати формування аналітичного звіту. Він взаємодіє з усіма основними сутностями — робочими місцями, приміщеннями та працівниками — забезпечуючи виведення зведеної інформації про зайнятість і навантаження.

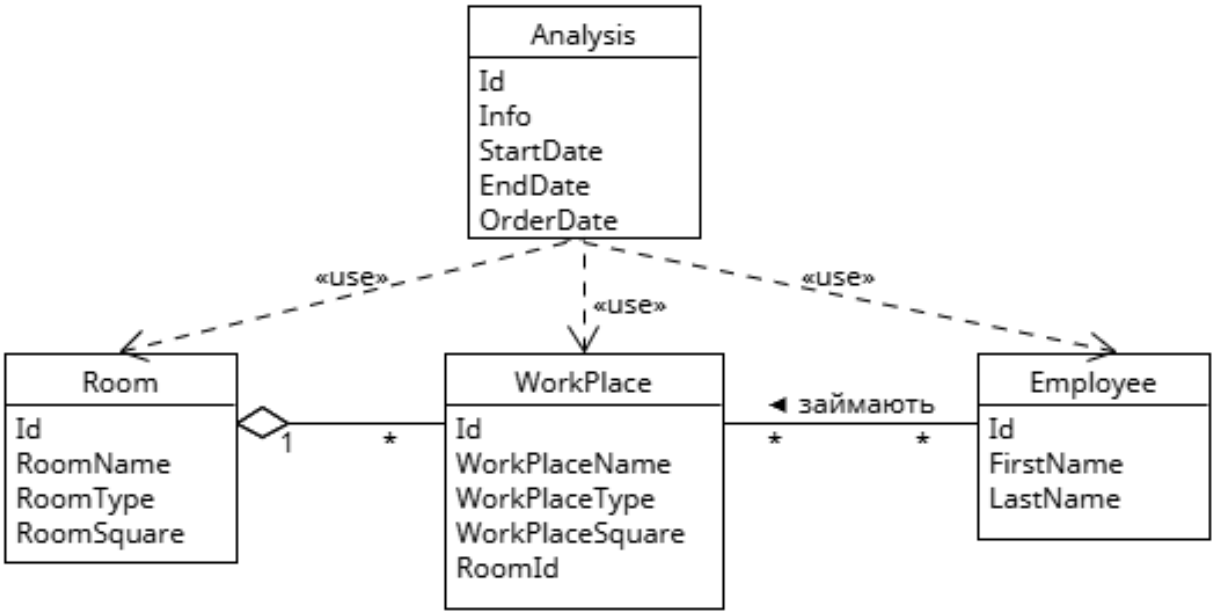


Рис. 3.2 Діаграма предметної галузі

Такий підхід дозволяє чітко структурувати доменну модель проєкту, що надалі лягає в основу проєктування бази даних і бізнес-логіки застосунку.

3.3 Моделювання процесів

Для опису алгоритмів поведінки системи та логіки виконання дій було використано діаграму діяльності. Вона дозволяє візуально представити послідовність кроків, що виконуються користувачем або системою під час роботи з основними функціями застосунку.

Алгоритм дій демонструє, як система реагує на перевищення порогу завантаженості робочих місць (рис. 3.3). Процес починається з ініціації бронювання робочого місця, після чого система виконує перевірку рівня зайнятості приміщення. Якщо рівень зайнятості перевищує 80%, здійснюється додаткова перевірка на можливість розміщення нового робочого місця у межах наявного приміщення.

У випадку, якщо площа не дозволяє розміщення додаткових місць, система формує повідомлення про необхідність найму нового приміщення. Після цього виконується процедура оренди і встановлення необхідної кількості робочих місць, що дозволяє забезпечити безперервний доступ працівників до робочого середовища.

Цей сценарій демонструє гнучкість та адаптивність системи в умовах змінного навантаження, а також автоматизацію управлінських рішень щодо розширення ресурсів. Модель дозволяє передбачити поведінку системи при високій завантаженості та забезпечити масштабованість інфраструктури при зростанні потреби.

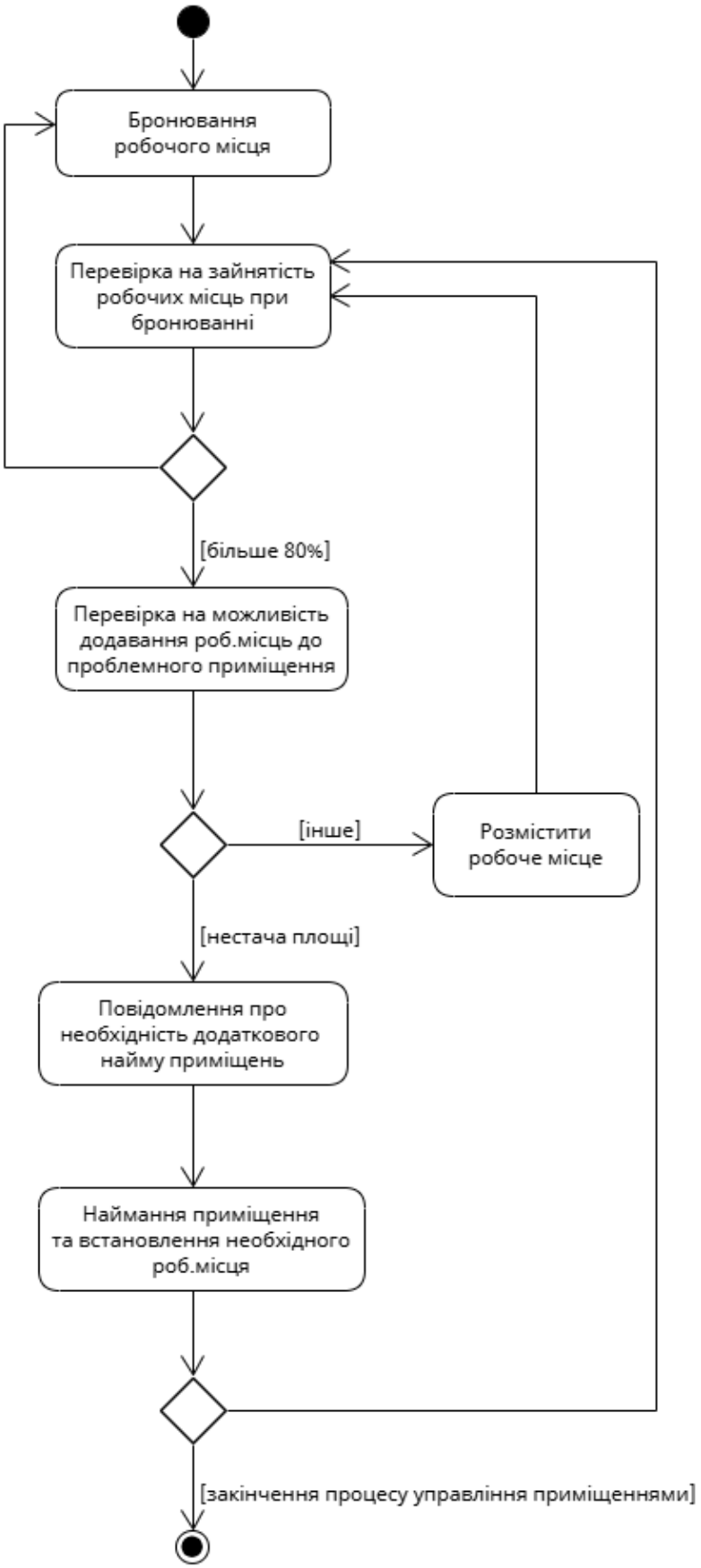


Рис. 3.3 Діаграма діяльності

3.4 Моделювання станів застосунку

Для моделювання поведінки користувача та переходів між різними етапами взаємодії із системою використано діаграму станів. Вона дозволяє відобразити логіку переходів між різними станами застосунку залежно від ролі користувача, обраних дій і поточного контексту.

На початку взаємодії користувач потрапляє до вікна входу, де може змінити мову інтерфейсу або ввести облікові дані. Після успішної авторизації користувач спрямовується на головну сторінку, яка адаптована до його ролі — працівник, адміністратор або представник відділу кадрів. Відповідно до призначеної ролі, інтерфейс пропонує набір відповідних дій, таких як перегляд графіка зайнятості, бронювання, робота з приміщеннями або доступ до звітів.

Алгоритм переходів між станами демонструє, як система надає доступ до функціональності на основі ролі користувача, а також як забезпечується логічний перехід між ключовими вікнами, такими як вхід, головне меню, графік зайнятості тощо.

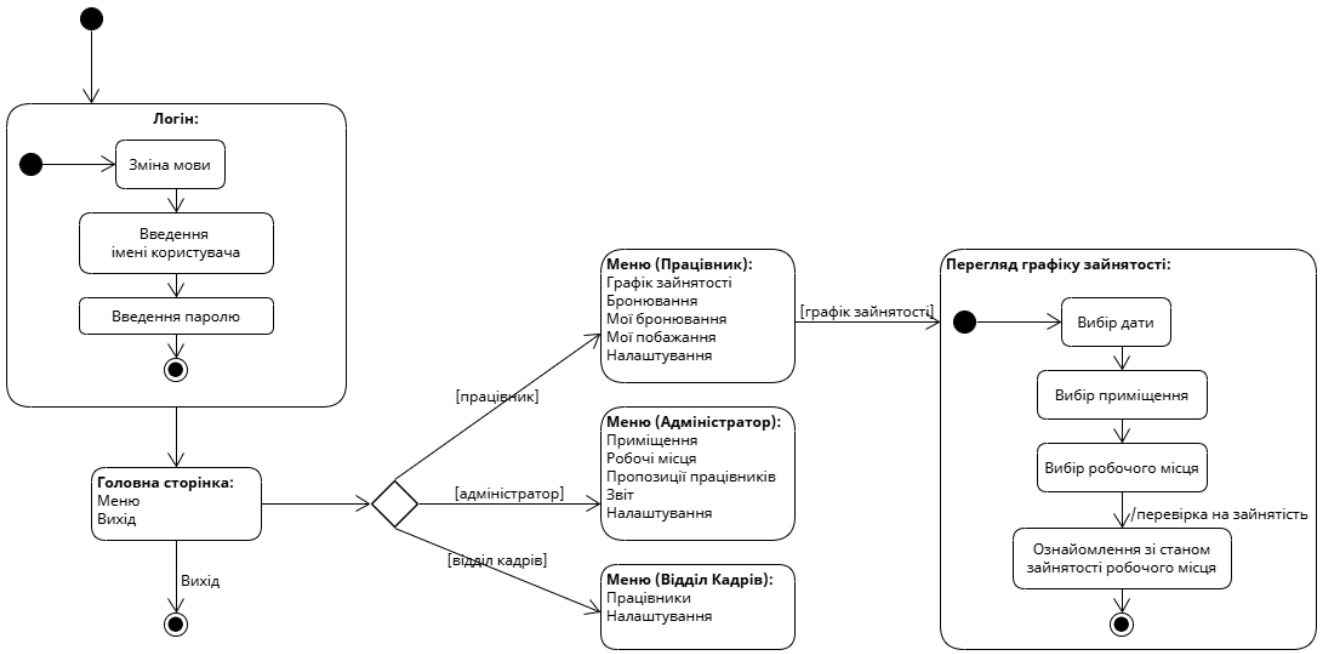


Рис. 3.4 Діаграма станів застосунку

3.5 Логічна архітектура

Для представлення внутрішньої організації програмного забезпечення було використано діаграму пакетів. Цей тип діаграми дозволяє структурно відобразити модулі (пакети), з яких складається система, а також залежності між ними.

Архітектура застосунку побудована за принципом логічного розділення відповідальностей, що відповідає тришаровій архітектурі:

- Презентаційний шар: відповідає за взаємодію з користувачем та відображення інформації.
- Бізнес-логіка: містить основні правила та логіку обробки даних.
- Шар доступу до даних: забезпечує збереження та отримання даних з бази даних.

Кожен модуль взаємодіє з іншими через чітко визначені інтерфейси, що спрощує масштабування та тестування. Наприклад, модуль бронювання об'єднує функціональність, пов’язану зі створенням, редагуванням і збереженням інформації про резервування, а також із перевіркою доступності робочих місць. Модуль звітів містить компоненти, які відповідають за агрегацію статистики та експорт даних. Користувацький інтерфейс ізольовано від бізнес-логіки, що дає змогу легко змінювати дизайн без втручання в ядро застосунку.

Використання діаграми пакетів (див. рисунок 3.5) для представлення тришарової архітектури дозволяє чітко структурувати систему, зменшити зв'язність між компонентами та полегшити підтримку та розвиток програмного забезпечення. Такий підхід сприяє підтримуваності, повторному використанню коду та підвищенню надійності при зміні або розширенні функціоналу.

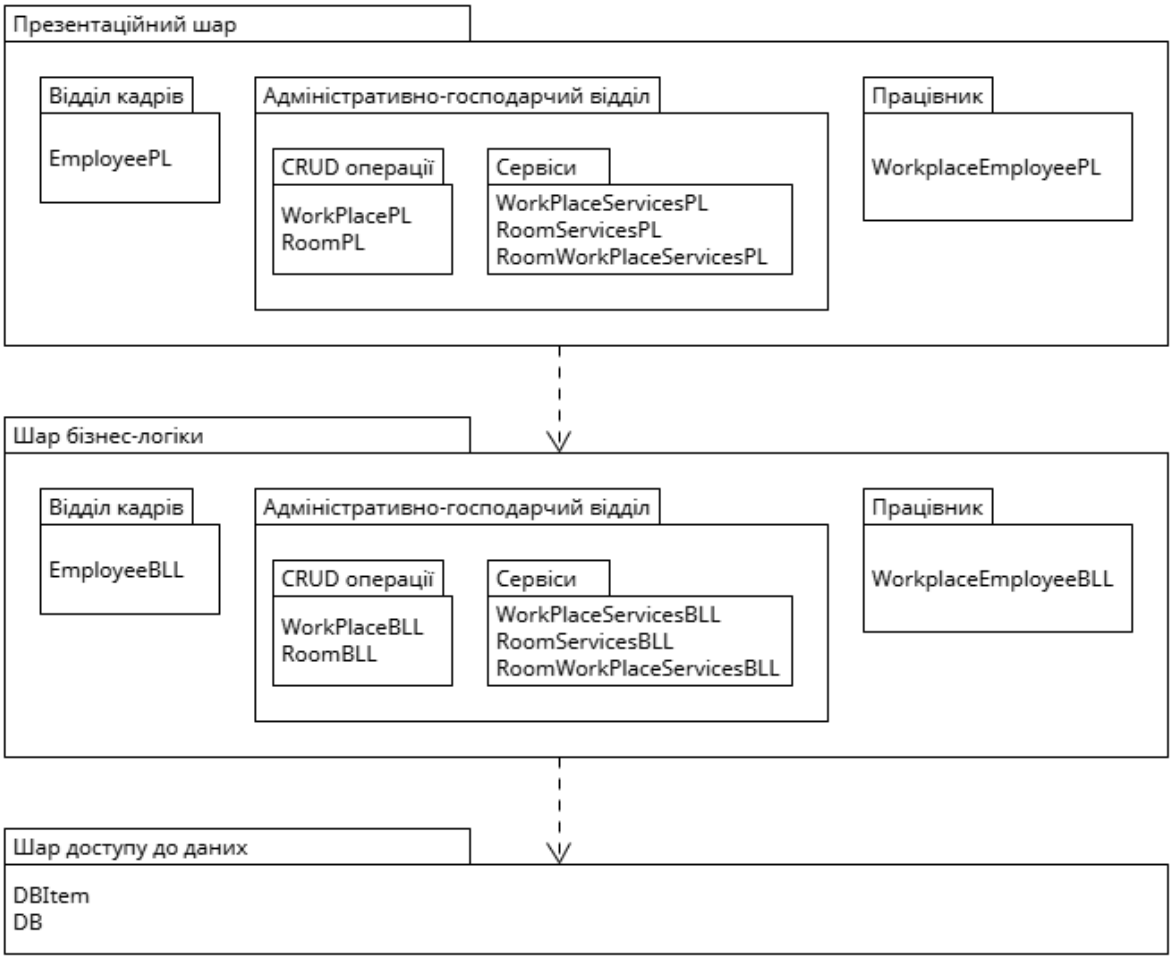


Рис. 3.5 Діаграма пакетів системи

3.6 Моделювання сценаріїв взаємодії

Для візуалізації послідовності обміну повідомленнями між об’єктами системи під час виконання конкретного сценарію використано діаграму послідовності. Вона дозволяє показати часовий порядок викликів методів, які забезпечують реалізацію функціоналу в межах одного логічного процесу.

Сценарій, зображений на діаграмі, демонструє взаємодію об’єктів під час виконання типового процесу бронювання робочого місця працівником. Користувач спершу ініціює запит на перегляд доступних робочих місць у задану

дату. Система викликає відповідні методи для перевірки графіка зайнятості, формує відповідь, і користувач обирає потрібний слот.

Далі здійснюється передача параметрів бронювання у форму, яка автоматично підставляє значення часу початку, завершення та вибраного місця. Після підтвердження з боку користувача дані зберігаються в базі, і система оновлює графік.

Ця діаграма ілюструє логіку взаємодії між інтерфейсом, контролером, сервісами перевірки доступності й збереження даних, забезпечуючи послідовну роботу з усіма основними компонентами системи.

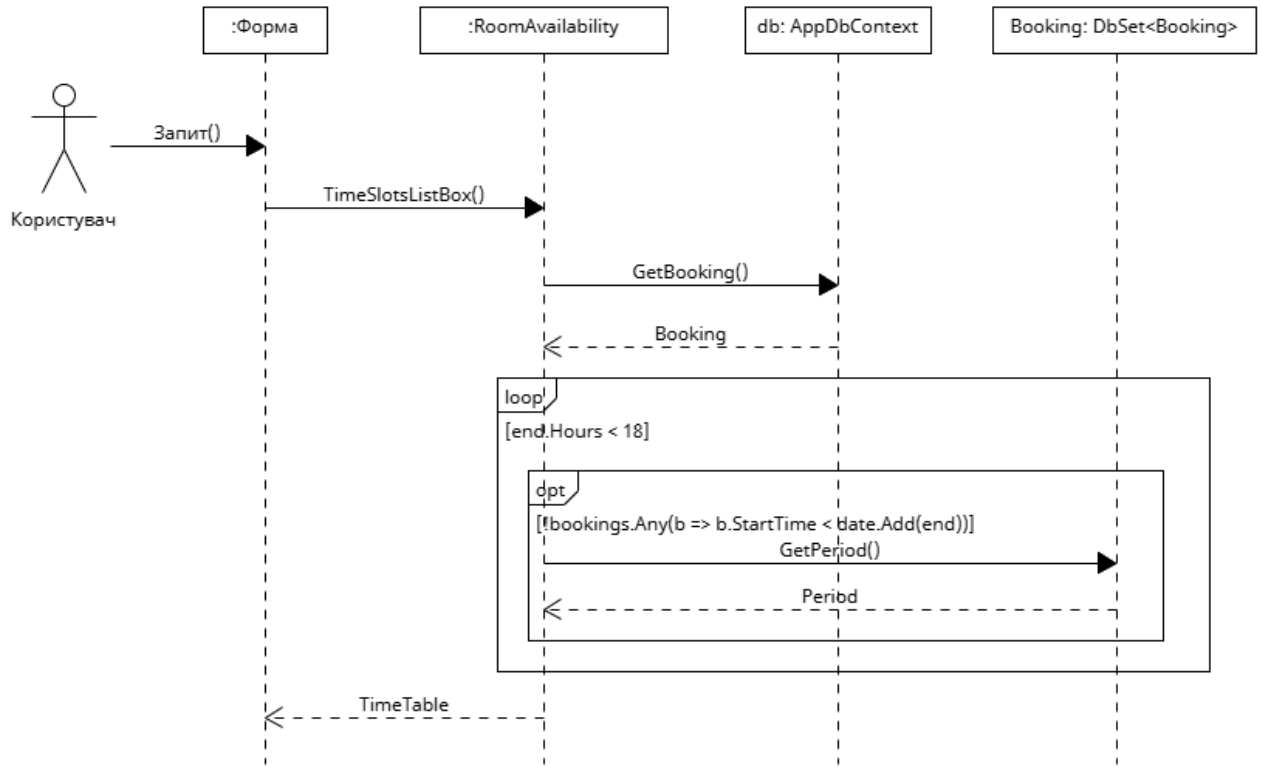


Рис. 3.6 Діаграма послідовності

3.7 Структура класів застосунку

Для представлення об’єктної моделі програмного забезпечення було використано діаграму класів. Вона дозволяє описати основні сутності системи, їхні атрибути, методи та зв’язки між класами, що є основою для реалізації логіки застосунку.

У системі управління офісним простором виділено декілька ключових класів:

- User — базовий клас, що містить атрибути логіну, пароля та ролі. Відповідає за автентифікацію та авторизацію.
- Employee — розширення користувача, яке містить інформацію про працівника: ПІБ, відділ, посаду.
- Room — містить опис приміщення, включно з його типом, розміром та кодом.
- Workplace — зв’язаний із Room та представляє конкретне робоче місце, що може бути забронюване.
- Booking — об'єкт, що фіксує інформацію про бронювання: час початку, час завершення, користувача, робоче місце.
- RoomSuggestion — клас для пропозицій працівників щодо поліпшення умов або функціональності приміщень.

Класи пов’язані між собою через відношення асоціації, агрегації та залежності. Зокрема, Booking асоціюється з Employee та Workplace, а Workplace з Room. Це забезпечує логічну цілісність моделі та узгодженість у збереженні даних.

Дана модель створена відповідно до принципів об’єктно-орієнтованого проектування та дозволяє гнучко масштабувати застосунок у майбутньому.

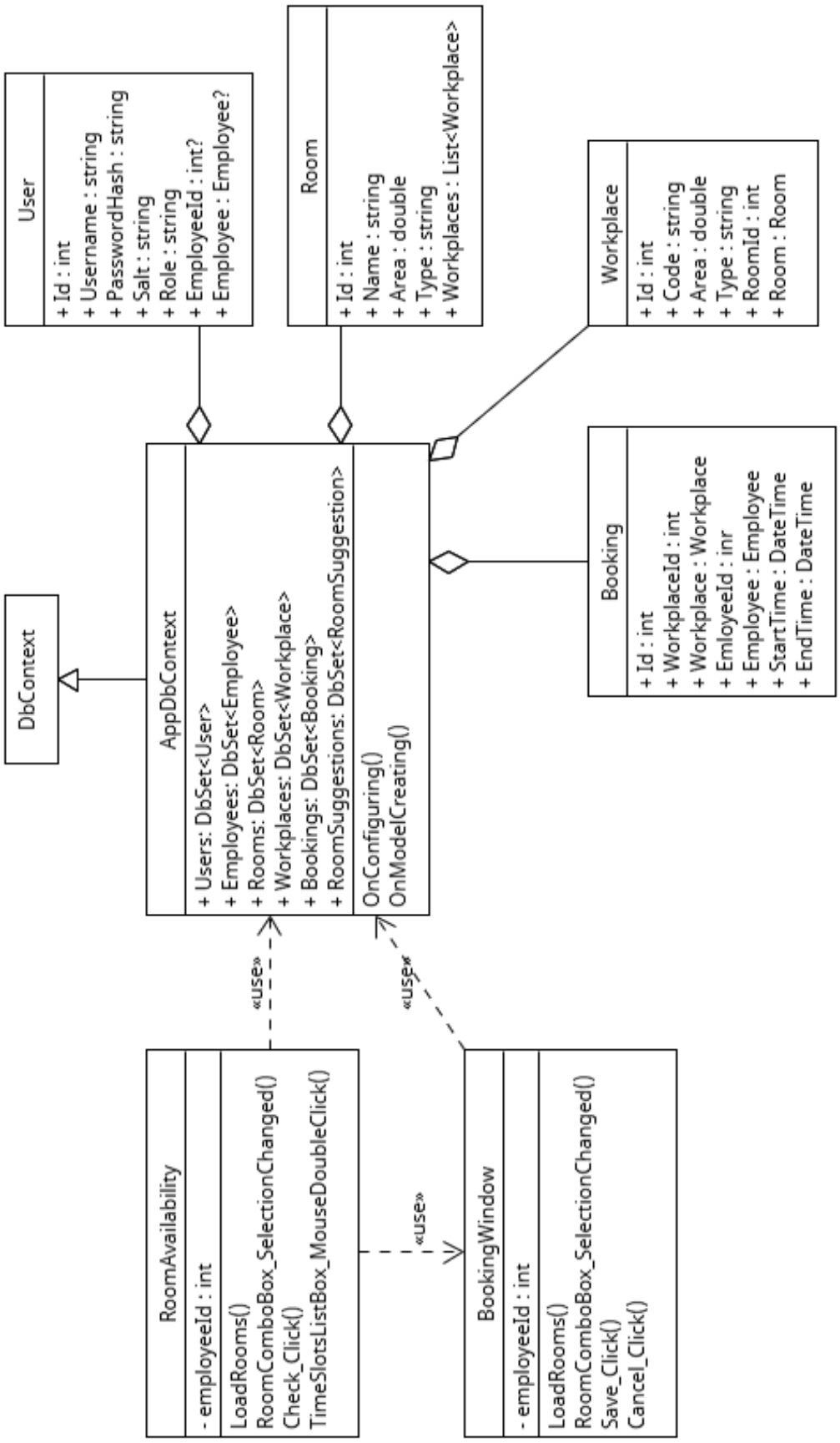


Рис. 3.7 Діаграма класів системи

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Реалізація інтерфейсу користувача

Інтерфейс користувача реалізовано з використанням технології Windows Presentation Foundation (WPF) — сучасної платформи для створення настільних застосунків під Windows. Основною перевагою WPF є підтримка декларативного підходу до опису інтерфейсів через мову XAML, а також можливість чіткого розділення візуальної частини та логіки програми.

Під час розробки застосовано шаблон проєктування MVVM (Model-View-ViewModel), що забезпечує високу модульність, підтримуваність та зручність тестування коду. Кожне вікно представлено окремим View, який зв’язаний із відповідною ViewModel за допомогою двобічного зв’язування (two-way binding).

Загальні підходи до побудови інтерфейсу:

- Навігаційна структура: застосунок реалізовано як набір окремих вікон, кожне з яких відповідає за певну функціональність (перегляд, редагування, створення).
- Data Binding: усі візуальні компоненти прив’язані до властивостей ViewModel, що дозволяє оновлювати інтерфейс у режимі реального часу без зайвої імперативної логіки.
- Команди (ICommand): взаємодія з інтерфейсом реалізована через команди, що забезпечує повну підтримку MVVM-патерна та зменшує залежність між представленням і логікою.
- Локалізація: реалізовано підтримку української та англійської мов за допомогою ресурсних файлів .resx та механізму перемикання мови в інтерфейсі.
- Динамічні обмеження: інтерфейс автоматично адаптується до ролі користувача, відкриваючи лише доступні функції (наприклад, перегляд звітів — лише для адміністратора, а графік зайнятості — лише для працівника).

Ключові екрани інтерфейсу:

- вікно авторизації (LoginWindow): реалізовано вхід користувачів до системи з валідацією даних (див. рис. 4.1).

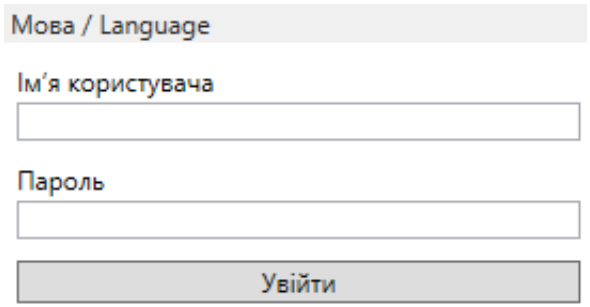


Рис. 4.1 Вікно авторизації

- вікно перегляду зайнятості (RoomAvailabilityWindow): працівник може обрати приміщення та робоче місце, побачити вільні години, а при натисканні на вільний слот — автоматично перейти до бронювання (див. рис. 4.2).

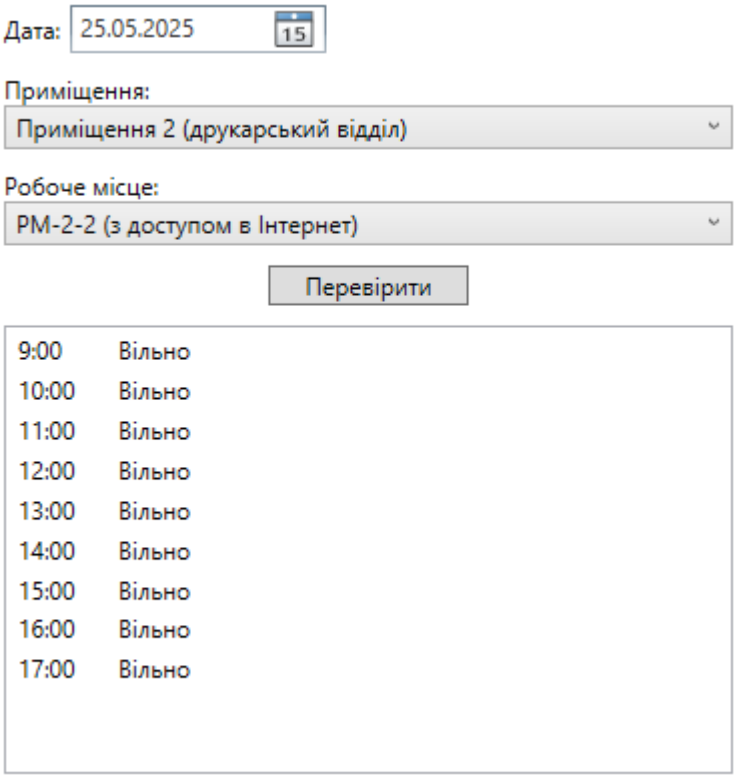


Рис. 4.2 Вікно перегляду зайнятості

- вікно бронювання (BookingWindow): користувач створює бронювання з попередньо заповненими полями дати, часу та місця. Реалізовано автоматичне визначення найбільшої кількості підряд вільних слотів (див. рис. 4.3).

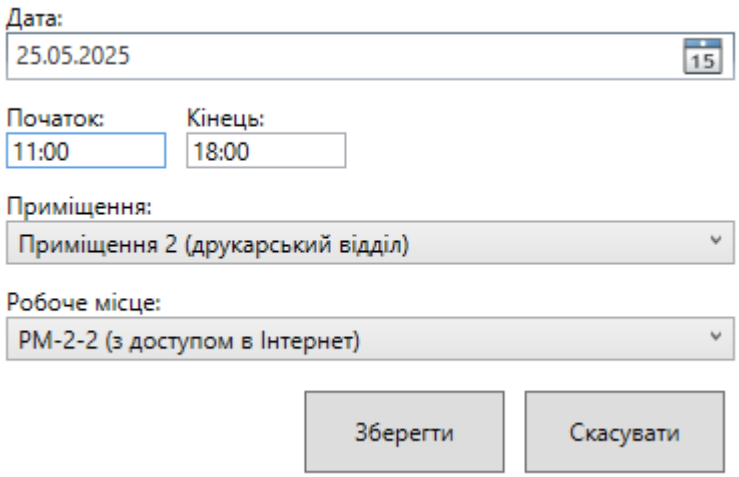


Рис. 4.3. Вікно бронювання

- вікно управління працівниками (EmployeeListWindow): HR може додавати нових працівників, редагувати їхні дані та створювати облікові записи з вибором ролі (див. рис. 4.4).

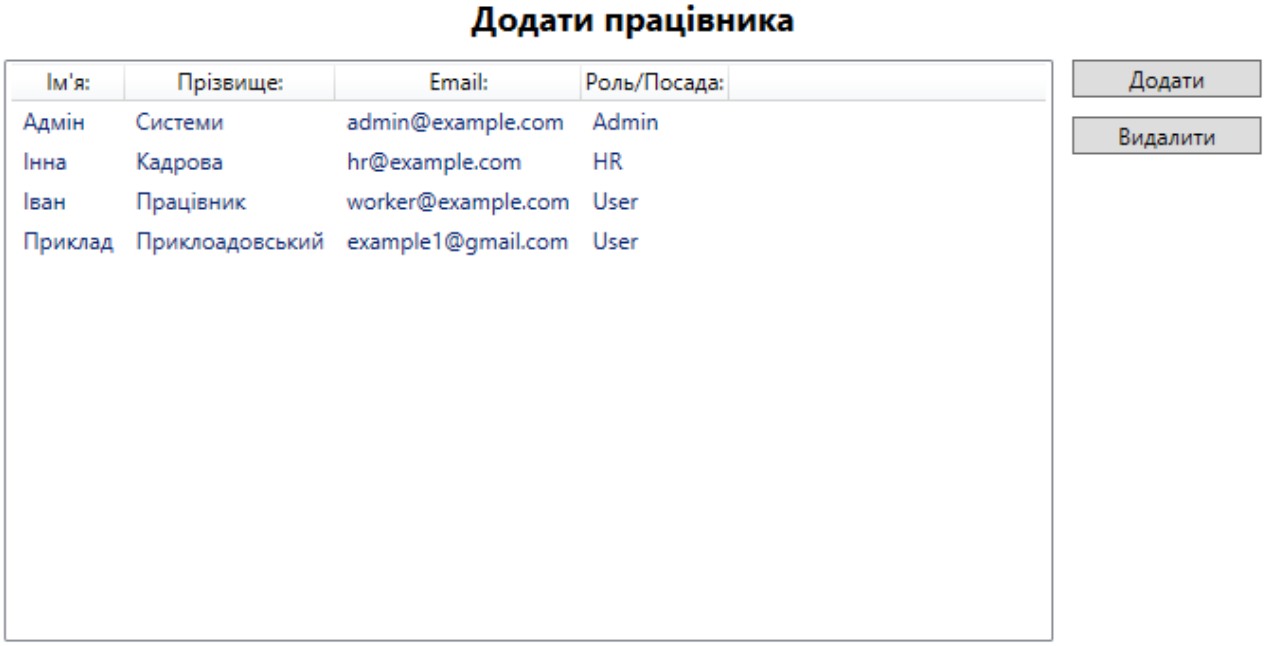


Рис. 4.4 Вікно управління працівниками

- вікна управління робочими місцями (WorkplaceListWindow) та приміщеннями (RoomListWindow): реалізовано можливість створення, редагування та видалення об’єктів офісного простору з фільтрацією за типом (див. рис. 4.5, 4.6).

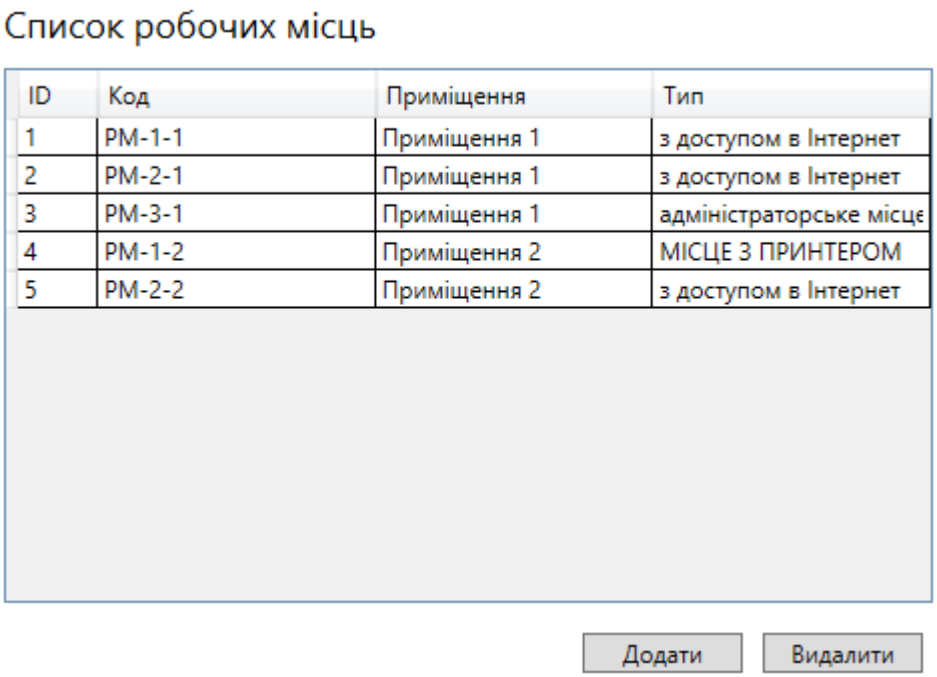


Рис. 4.5 Вікно управління робочими місцями

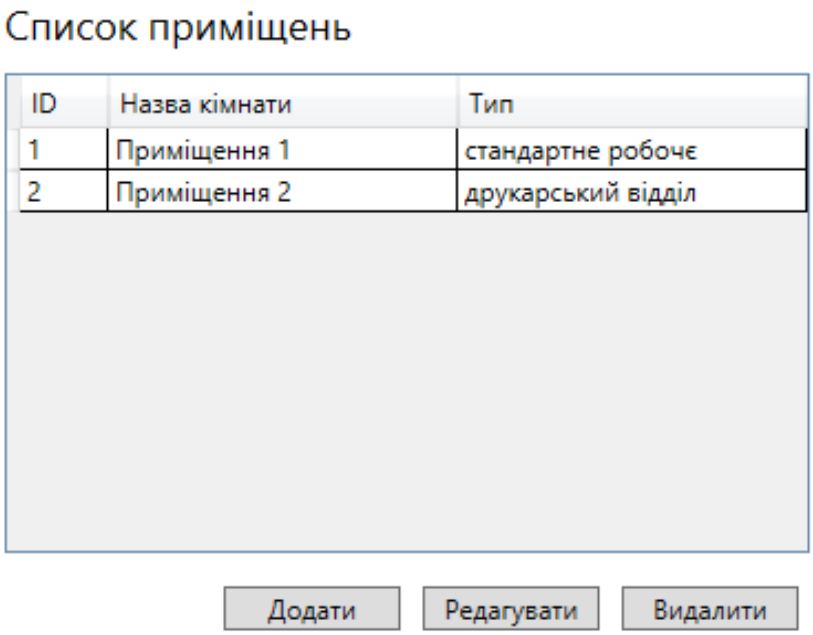


Рис. 4.6 Вікно управління приміщеннями

- вікна пропозицій працівників (MySuggestionsWindow для працівників та AdminSuggestionsWindow для адміністраторів): працівники залишають побажання щодо змін у приміщеннях, а адміністратор може позначати їх як переглянуті чи виконані (див. рис. 4.7, 4.8).

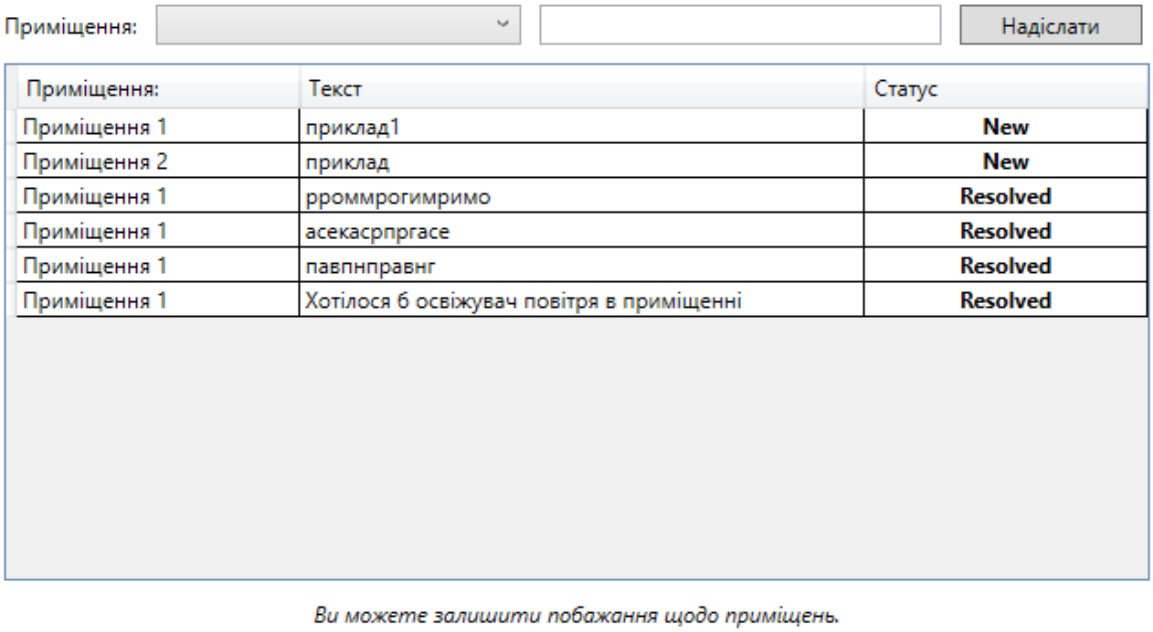


Рис. 4.7 Вікно пропозицій працівників (з позиції працівника)

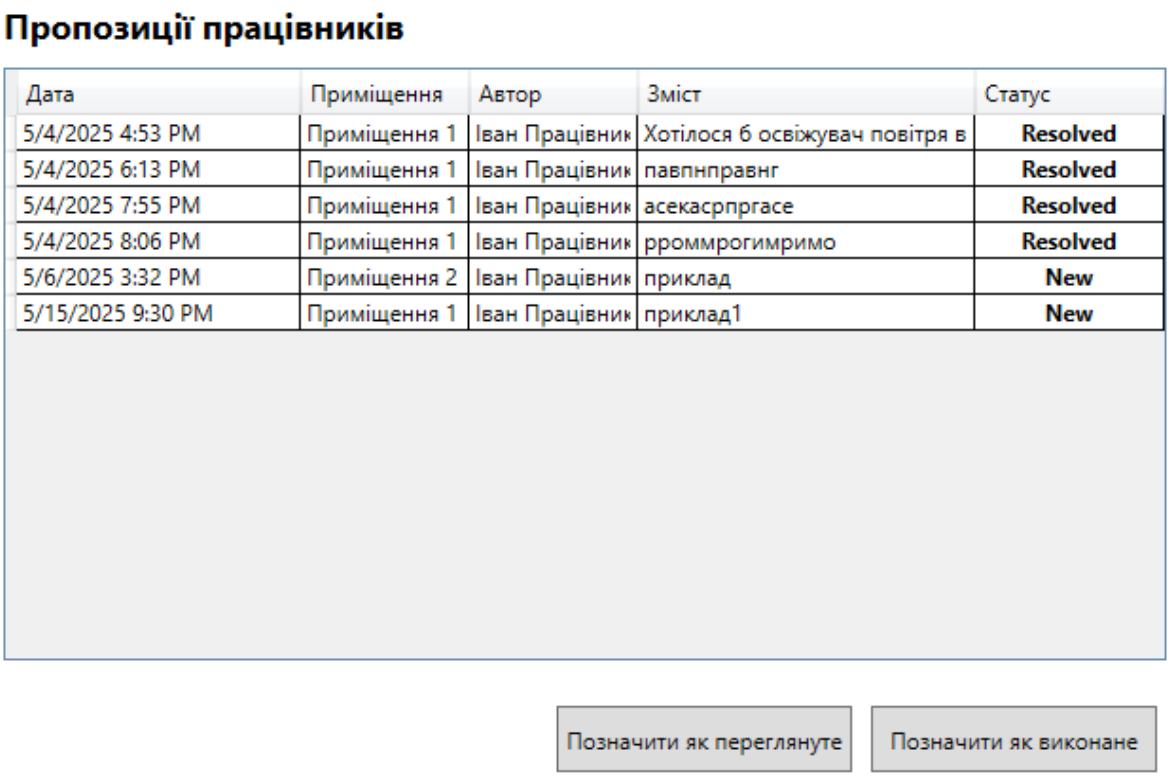


Рис. 4.8 Вікно пропозицій працівників (з позиції адміністратора)

- вікно звітування (ReportWindow): адміністратори можуть отримати звіт за певний період щодо зайнятості певного приміщення і отримати рекомендацію від системи залежно від відсотка завантаженості, з можливістю подальшого експорту звіту у вигляді XLSX або CSV файлу (див. рис. 4.9).

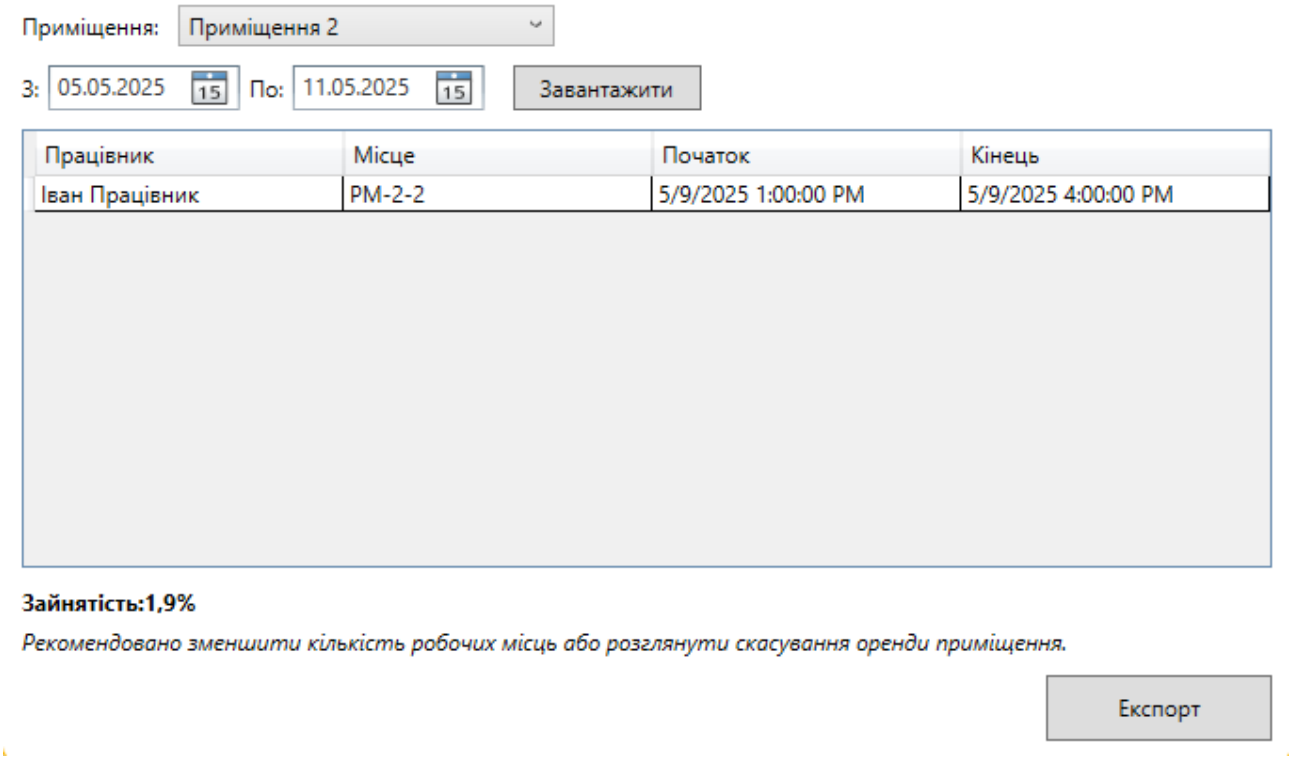


Рис. 4.9 Вікно звітування

Для кожного вікна використано стандартні компоненти WPF: Grid, StackPanel, DataGrid, ComboBox, TextBox, Label, Button, ListView, DatePicker, TimePicker тощо. Оформлення реалізовано через стилі та шаблони з урахуванням зручності для користувача.

Для зручності взаємодії між компонентами ViewModel застосовано механізми впровадження подій (EventAggregator) та викликів через делегати, що дозволяє організувати навігацію між вікнами без прямого доступу до їхніх екземплярів.

4.2 Реалізація клієнтської частини

Клієнтська частина застосунку реалізована на платформі Windows Presentation Foundation (WPF) з використанням мови програмування C#. WPF дозволяє створювати графічні інтерфейси з підтримкою сучасного дизайну, шаблонів стилізації, прив’язки даних та анімацій. Клієнтська частина відповідає за взаємодію користувача з системою, обробку введення та візуалізацію даних.

Основні кроки реалізації клієнтської частини:

1. Проєкт було створено у середовищі Visual Studio з типом шаблону “WPF App (.NET Framework)”. У структурі проєкту виділено окремі вікна (Window) для кожної суттєвої функціональності: авторизація, перегляд графіка, бронювання, управління працівниками, робочими місцями тощо.

2. Усі інтерфейси реалізовано у вигляді окремих XAML-файлів із прив’язаною логікою в C# (code-behind). Наприклад:

- LoginWindow.xaml – авторизація
- RoomAvailabilityWindow.xaml – перегляд зайнятості
- BookingWindow.xaml – бронювання
- EmployeeListWindow.xaml, WorkplaceListWindow.xaml – управління відповідними сутностями

Для організації стилів було створено ресурси, що забезпечують візуальну узгодженість додатку.

3. Основною технікою для зв’язку інтерфейсу з даними є прив’язка WPF (Binding). Це дозволяє забезпечити автоматичне оновлення інтерфейсу при зміні даних без додаткового коду оновлення UI вручну.

4. Усі взаємодії з базою даних здійснюються через AppDbContext з використанням Entity Framework Core. Наприклад, у вікні RoomAvailabilityWindow під час вибору приміщення відбувається оновлення доступних слотів:

```
TimeSlotsListBox.ItemsSource = GetAvailableTimeSlots(selectedRoomId);
```

5. Усі форми введення (логін, паролі, імена тощо) мають попередню перевірку заповненості полів і валідацію. У разі помилки виводяться підказки.

6. Клієнтська частина підтримує зміну мови інтерфейсу за допомогою ресурсів (.resx). Мова може змінюватися прямо з інтерфейсу (у LoginWindow), і нові тексти застосовуються без перезапуску програми.

7. Навігація між різними частинами застосунку реалізована за допомогою відкриття та закриття відповідних вікон. Під час переходу до вікна бронювання, обрані дані автоматично передаються:

```
var bookingWindow = new BookingWindow(selectedRoomId, selectedHour);
bookingWindow.Show();
```

Клієнтська частина реалізована таким чином, щоб бути інтуїтивно зрозумілою, масштабованою та придатною до локалізації**.** Це дозволяє легко адаптувати застосунок під різні категорії користувачів — працівників, HR-відділ та адміністрацію.

4.3 Реалізація серверної частини

Серверна частина застосунку реалізована за допомогою ORM-бібліотеки Entity Framework Core, що входить до екосистеми .NET. Вона забезпечує доступ до бази даних SQLite, автоматизуючи роботу з таблицями та зв'язками за допомогою класів-моделей. Уся логіка роботи з даними зосереджена у класі AppDbContext, що є контекстом бази даних.

4.3.1. Ініціалізація контексту бази даних

Для збереження та доступу до даних було створено клас AppDbContext, який наслідує DbContext:

```
public class AppDbContext : DbContext
{
    5 references
    public DbSet<User> Users { get; set; }
    5 references
    public DbSet<Employee> Employees { get; set; }
    13 references
    public DbSet<Room> Rooms { get; set; }
    10 references
    public DbSet<Workplace> Workplaces { get; set; }
    8 references
    public DbSet<Booking> Bookings { get; set; }
    5 references
    public DbSet<RoomSuggestion> RoomSuggestions { get; set; }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder options)
    {
        var dbPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "..", "..", "..", "office.db");
        var fullPath = Path.GetFullPath(dbPath);

        options.UseSqlite($"Data Source={fullPath}");
    }
```

4.3.2. Створення моделей сутностей

Кожна сутність системи — користувач, працівник, приміщення, робоче місце — описується відповідною моделлю з полями. Приклад класу Room:

```
public class Room
{

    public int Id { get; set; }

    public string Name { get; set; } = "";

    public double Area { get; set; }

    public string Type { get; set; } = "";

    1 reference
    public ICollection<Workplace> Workplaces { get; set; } = new List<Workplace>();
}
```

Також реалізовано зв’язки між сутностями: один-до-багатьох (приміщення → робочі місця), один-до-одного (користувач → працівник), та багато-до-одного (бронювання → працівник, робоче місце).

4.3.3. Зв’язок із клієнтською частиною

Контекст бази даних використовується у клієнтських вікнах для виконання операцій читання, запису та оновлення:

```
using (var context = new AppDbContext())
{
    var rooms = context.Rooms.Include(r => r.Workplaces).ToList();
    RoomsComboBox.ItemsSource = rooms;
}
```

4.3.4. Міграція та ініціалізація

На початку запуску програми перевіряється наявність бази даних і створюється структура, якщо вона відсутня:

```
using (var context = new AppDbContext())
{
    context.Database.Migrate();
}
```

Для цього були додані початкові міграції через CLI-команду:

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

4.3.5. Інкапсуляція бізнес-логіки

Операції, пов’язані з бронюванням, перевіркою доступності слотів, створенням пропозицій та генерацією звітів, реалізовано у вигляді методів у вікнах або допоміжних класах. Наприклад, перевірка доступності слотів:

```
bool IsAvailableSlot(Workplace wp, DateTime time)
{
    return !context.Bookings.Any(b => b.WorkplaceId == wp.Id && b.StartTime <= time && b.EndTime > time);
}
```

Серверна частина застосунку реалізована як частина десктопного рішення, що об'єднує локальний ORM-контекст, моделі з повними зв'язками, використання LINQ-запитів та автоматичну генерацію бази даних. Такий підхід дозволяє ефективно обробляти локальні дані без підключення до зовнішнього сервера, що спрощує розгортання системи.

4.4 Завантаження проєкту на GitHub

Для зберігання та спільної розробки програмного забезпечення було використано сервіс GitHub — популярну хмарну платформу, засновану на системі контролю версій Git. Репозиторій дозволяє зберігати код, відстежувати зміни, створювати гілки, вирішувати конфлікти, а також забезпечує доступність проєкту для інших учасників команди.

Перед завантаженням проєкту на GitHub необхідно ініціалізувати репозиторій у локальній папці з проєктом. Це виконується в терміналі або командному рядку за допомогою команди:

```
git init
```

Після ініціалізації система створює приховану директорію .git, яка містить усі метадані репозиторію.

Для підключення до вже створеного репозиторію на GitHub потрібно задати його URL:

```
git remote add origin https://github.com/VestaxUA/OfficeSpaceManager
```

Усі необхідні файли проєкту додаються до зони індексації за допомогою:

```
git add
```

Далі виконується фіксація змін у локальному репозиторії з відповідним описом:

```
git commit -m "Initial commit with implementation of booking and workspace features"
```

Останнім кроком є завантаження закомічених змін у віддалений репозиторій:

```
git push -u origin master
```

У результаті виконання цієї команди весь проєкт стане доступним у репозиторії на GitHub, де з ним можна працювати з будь-якого пристрою або співпрацювати з іншими розробниками.

У репозиторії зберігається як код клієнтської частини (WPF), так і вся логіка роботи з базою даних (Entity Framework), ресурси для локалізації, інтерфейсні компоненти, тестові класи та вихідна база даних (office.db).

4.5 Тестування застосунку

Для перевірки коректності реалізованої бізнес-логіки було проведено модульне тестування окремих компонентів серверної частини застосунку. Модульні тести дозволяють виявити помилки в ізольованих частинах коду та забезпечити стабільність основних функцій системи після кожної зміни.

Усі тести зберігаються в окремій теці Tests згідно з прийнятими рекомендаціями щодо структури проєкту. Для написання тестів було використано фреймворк xUnit, який є стандартом для тестування у середовищі .NET.

Основні переваги застосування модульного тестування:

- Ізоляція — кожен тест охоплює окремий метод або функцію, що дозволяє легко ідентифікувати джерело помилки.
- Автоматизація — тести можна запускати після кожної зміни коду, що сприяє безперервній інтеграції.
- Надійність — впевненість у стабільній роботі застосунку навіть після масштабних змін.
- Придатність до рефакторингу — наявність тестів спрощує зміну структури коду без втрати функціональності.

Приклади модульних тестів :

1. Перевірка розрахунку загальної зайнятості приміщення:

```
[Fact]
0 references
public void CalculateOccupancyPercentage_WhenSingleBookingOneDay_ShouldReturnCorrectValue()
{
    var bookings = new List<Booking>
    {
        new Booking { WorkplaceId = 1, StartTime = DateTime.Today, EndTime = DateTime.Today.AddHours(2) }
    };

    var start = DateTime.Today;
    var end = DateTime.Today.AddDays(1);

    var workplaceIds = bookings.Select(b => b.WorkplaceId).Distinct().ToList();
    var totalSlots = workplaceIds.Count * 9 * (int)(end - start).TotalDays;
    var usedSlots = bookings.Count;
    double percentage = totalSlots > 0 ? (usedSlots / (double)totalSlots) * 100 : 0;

    Assert.Equal(11.11, Math.Round(percentage, 2));
}
```

Мета: перевірити правильність розрахунку тривалості зайнятості. Очікування: метод .CalculateOccupancyPrecentage має повернути 11,11%

2. Перевірка доступності години для бронювання:

```
[Fact]
0 references
public void AddBooking_WhenTimeIsFree_ShouldSucceed()
{
    var bookings = CreateTestBookings();
    var newStart = new DateTime(2025, 5, 25, 11, 0, 0);
    var newEnd = new DateTime(2025, 5, 25, 14, 0, 0);

    bool overlaps = bookings.Any(b => b.WorkplaceId == 1 && newStart < b.EndTime && newEnd > b.StartTime);

    Assert.False(overlaps);
}
```

Мета: визначити, чи система правильно реагує на спробу бронювання вже зайнятого слоту.

Очікування: тест повертає False.

3. Перевірка логіки хешування пароля:

```
[Fact]
0 references
public void VerifyPassword_ShouldReturnTrue_WhenPasswordIsCorrect()
{
    var password = "mySecret";
    var salt = "test_salt";
    var hash = PasswordHelper.HashPassword(password, salt);

    var result = PasswordHelper.VerifyPassword(password, hash, salt);

    Assert.True(result);
}
```

Мета: перевірити, чи повертає система true при правильному паролі. Очікування: тест повинен пройти успішно.

Усі тести виконуються за допомогою середовища Visual Studio або командного рядка. Приклад команди для запуску тестів:

```
dotnet test
```

Результат: тести проходять успішно, що підтверджує коректність логіки роботи модулів, пов’язаних із бронюванням, обліком часу, перевіркою доступності та безпечним зберіганням паролів.

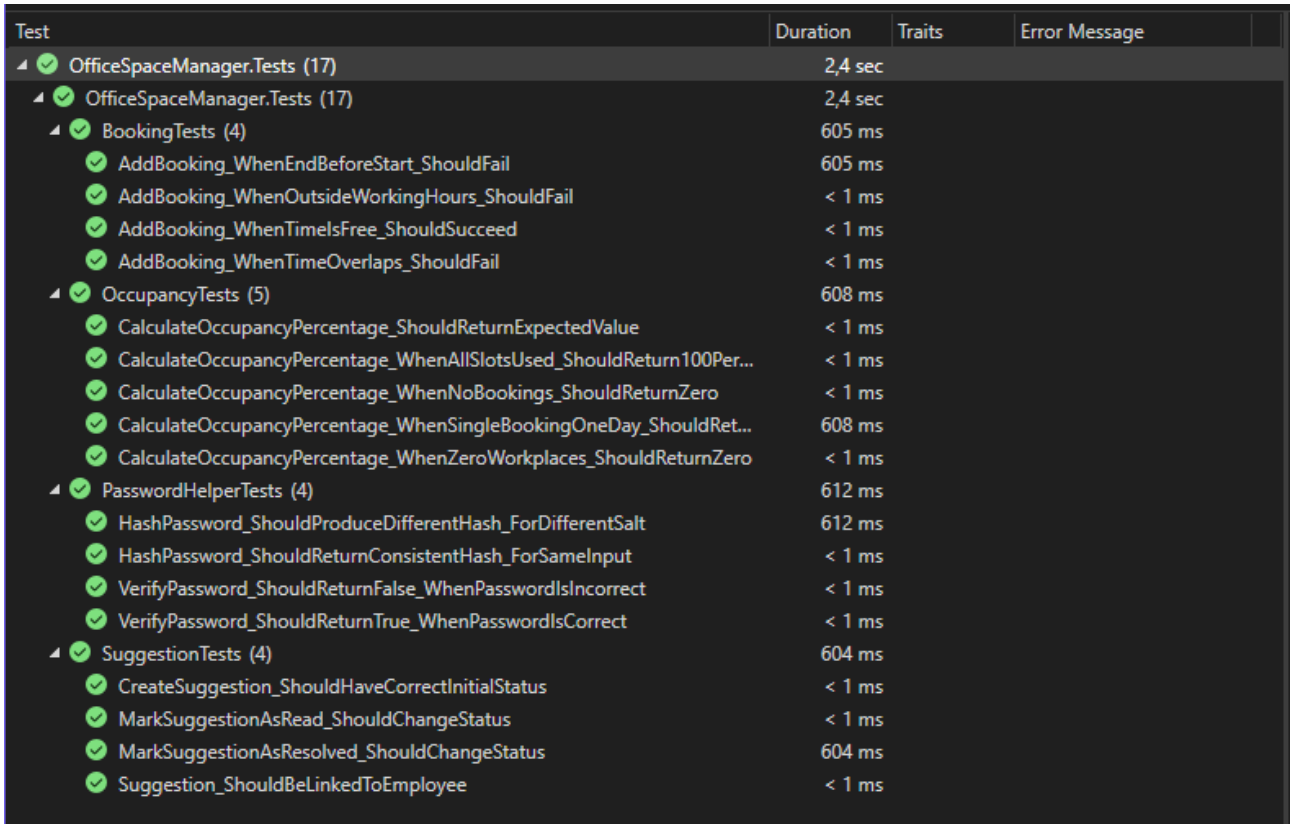


Рис. 4.10 Результати проходження модульного тестування

ВИСНОВКИ

Результатом виконаної роботи стало створення функціонального застосунку для управління офісним простором, який дозволяє оптимізувати процеси бронювання робочих місць, обліку приміщень та генерації звітів. У межах роботи було реалізовано низку функціональних і нефункціональних вимог, зосереджених на зручності користування, безпеці та масштабованості системи.

У процесі виконання кваліфікаційної роботи були розв’язані такі основні завдання:

- Проаналізовано предметну галузь: досліджено особливості організації офісного простору, розглянуто цільову аудиторію, проаналізовано переваги та недоліки існуючих систем, зокрема таких як Condeco, Skedda, Robin Powered та OfficeRnD Hybrid. Визначено перелік функціональних і нефункціональних вимог до системи.
- Обґрунтовано вибір засобів реалізації: середовище розробки Visual Studio, мова програмування C#, WPF для реалізації графічного інтерфейсу, SQLite як вбудована СУБД, GitHub для контролю версій, Entity Framework для роботи з базою даних, ClosedXML для експорту звітів, xUnit для модульного тестування.
- Розроблено програмну архітектуру системи: здійснено проєктування структури даних, взаємодії між компонентами, виконано побудову UML-діаграм (прецедентів, предметної області, діяльності, станів, пакетів, послідовності та класів), що дозволило формалізувати логіку функціонування застосунку.
- Реалізовано ключові компоненти: розроблено графічний інтерфейс користувача з підтримкою ролей, мов і тем; впроваджено функціонал бронювання, фільтрації, перегляду доступності, ведення пропозицій, генерації звітів. Застосунок забезпечує локалізацію інтерфейсу, зберігання даних в SQLite, та захист авторизації через хешування паролів.

- Проведено модульне тестування: розроблено і виконано тести для основних компонентів системи, що підтвердило коректну роботу функціоналу відповідно до вимог.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Ткачищен В.М., Гаманюк І.М. Використання діаграми моделі предметної галузі для моделювання предметної галузі під час розробки застосунку з управління офісним простором: Матеріали VI Міжнародної науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». 15.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій (подано до друку).

2. Ткачищен В.М., Гаманюк І.М. Проектування архітектури застосунку для управління офісним простором за допомогою діаграми пакетів: Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.2025, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез К.: ДУІКТ, 2025. С.467-469.

3. Ткачищен В.М., Шевченко С.М. Захист інформації у застосунку для управління офісним простором: Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025». 15.05.2025, Київ, Київський столичний університет імені Бориса Грінченка. Збірник тез К.: КУБГ, 2025. С.335.

ПЕРЕЛІК ПОСИЛАНЬ

1. Bennett S. Integrated Workplace Management Systems Statistics 2025. LLCBuddy. 23.03.2025. URL: https://llcbuddy.com/data/integrated-workplace-management-systems-statistics/

2. Defining the digital workplace: a systematic literature review / L. Mićić та ін. Strategic Management. 2022. URL: https://www.researchgate.net/publication/358954088_Defining_the_digital_workplace_A_systematic_literature_review.

3. Marsh E., Perez Vallejos E., Spence A. Digital workplace technology intensity: qualitative insights on employee wellbeing impacts of digital workplace job demands. Frontiers in Organizational Psychology. 2024. URL: https://doi.org/10.3389/forgp.2024.1392997.

4. Oladiran, O., Hallam, P., & Elliott, L. (2023). The COVID-19 pandemic and office space demand dynamics. International Journal of Strategic Property Management, 27(1), 35–49. URL: https://doi.org/10.3846/ijspm.2023.18003

5. Al-Saudi N.H., Flayyih H.H. The Impact of Digital Transformation on Office Management Efficiency. Journal of Economic and Administrative Sciences, 2024. URL: https://www.researchgate.net/publication/383696669_The_Impact_of_Digital_Transformation_on_Office_Management_Efficiency

6. Condeco. URL: https://www.condecosoftware.com/

7. Skedda. URL: https://www.skedda.com/

8. Robin Powered. URL: https://robinpowered.com/

9. OfficeRnD Hybrid. URL: https://www.officernd.com/

10. Visual Studio product family documentation. URL: https://learn.microsoft.com/en-us/visualstudio/?view=vs-2022

11. C# language documentation. URL: https://learn.microsoft.com/en-us/dotnet/csharp/

12. Microsoft Docs. .NET Documentation. URL: https://learn.microsoft.com/en-us/dotnet/

13. Microsoft Docs. Windows Presentation Foundation (WPF). URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/

14. SQLite Documentation. URL: https://www.sqlite.org/docs.html

15. Git Documentation. URL: https://git-scm.com/doc

16. GitHub Docs. URL: https://docs.github.com/en

17. Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/

18. ClosedXML. URL: https://github.com/ClosedXML/ClosedXML

19. xUnit.net. URL: https://xunit.net/

20. Koç, H., Erdoğan, A. M., Barjakly, Y., & Peker, S. (2021). UML Diagrams in Software Engineering Research: A Systematic Literature Review. Proceedings, 74(1), 13. DOI: 10.3390/proceedings2021074013

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку для управління офісним простором

Виконав студент 4 курсу
Групи ПД-42
Ткачищен Валентин Михайлович
Керівник роботи
Старший викладач кафедри ІПЗ Гаманюк Ігор Михайлович

Київ – 2025

МЕТА, ОБ’ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращити діяльність компанії шляхом автоматизації процесу управління офісним простором за рахунок створення застосунку з управління офісним простором

Об’єкт дослідження: процес управління офісним простором

Предмет дослідження: програмне забезпечення для управління офісним простором

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати особливості управління офісним простором та існуючі програмні рішення.
2. Визначити функціональні та нефункціональні вимоги до майбутнього застосунку.
3. Розробити архітектуру та структуру даних застосунку.
4. Спроєктувати та реалізувати графічний інтерфейс користувача.
5. Реалізувати функціонал для ведення обліку приміщень, робочих місць і бронювань.
6. Забезпечити підтримку ролей користувачів та локалізації інтерфейсу.
7. Перевірити реалізовану систему

3

АНАЛІЗ АНАЛОГІВ

	Condeco	Skedda	Robin Powered	OfficeRnD Hybrid	OfficePoint
Бронювання робочих місць	+	+	+	+	+
Аналітика завантаженості	+	±	+	+	+
Підтримка доступу за ролями	+	±	+	+	+
Перегляд графіку зайнятості за датою	±	-	+	±	+
Автопідставлення даних при бронюванні зі слота	-	-	±	-	+
Обмеження доступу до історії бронювань для працівників	-	-	-	-	+
Локалізація (українська мова)	-	-	-	-	+
Пропозиції працівників щодо приміщень	-	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги

1. Додавання, редагування та видалення інформації про працівників.
2. Додавання, зміна та видалення даних про робочі місця.
3. Функція бронювання робочих місць (для працівників).
4. Ведення реєстру приміщень з можливістю пошуку вільних.
5. Генерація звітів про зайнятість та активність.
6. Можливість залишати та переглядати пропозиції щодо приміщень.

Нефункціональні вимоги

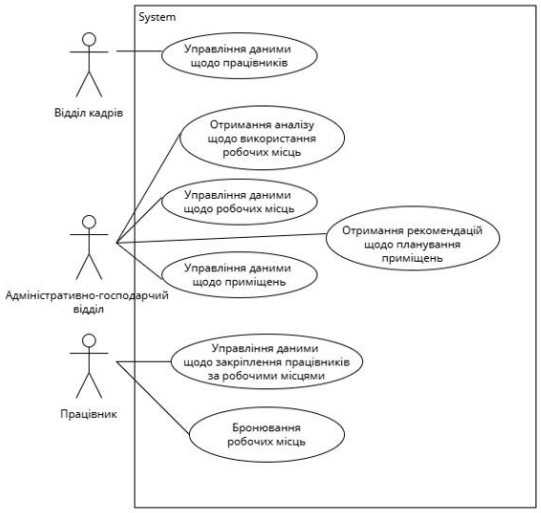
1. Безпека - захист даних користувачів шляхом реалізації авторизації з розподілом ролей та збереженням паролів у вигляді хешу.
2. Локалізація - підтримка української та англійської мов інтерфейсу для зручності користувачів з різними мовними вподобаннями.
3. Масштабованість - можливість підтримки великої кількості користувачів (до 1000+) без втрати продуктивності системи.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

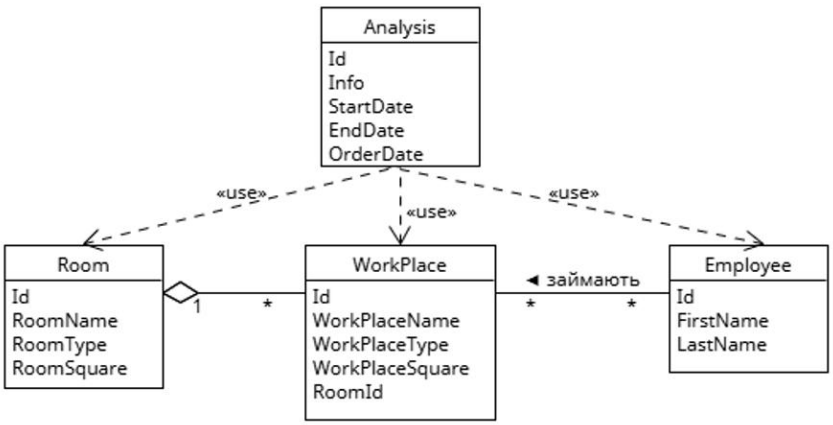
6

ДІАГРАМА ПРЕЦЕДЕНТІВ



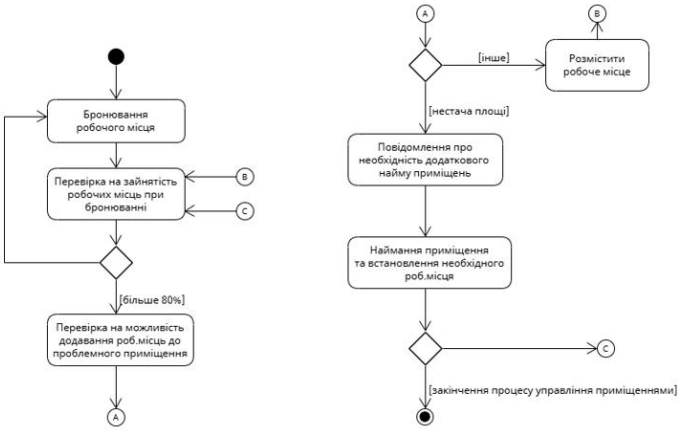
7

БІЗНЕС МОДЕЛЬ



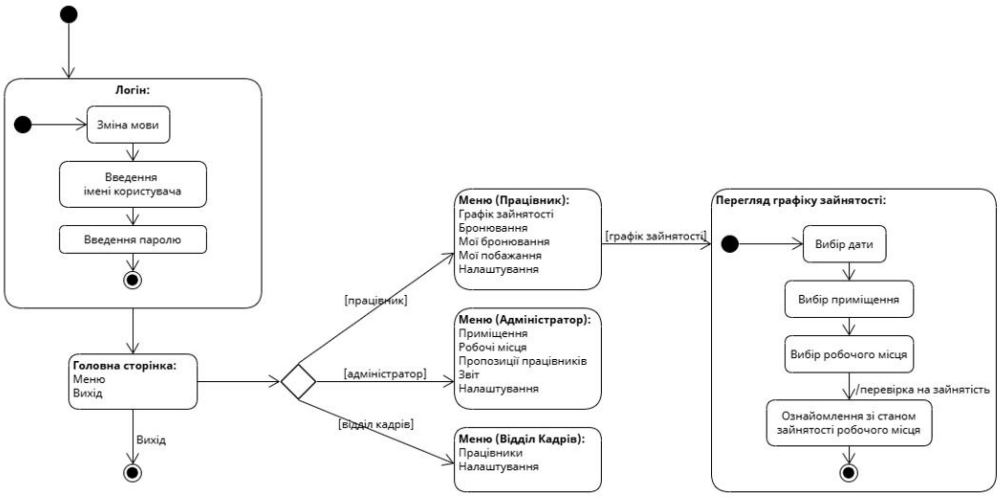
8

ДІАГРАМА ДІЯЛЬНОСТІ



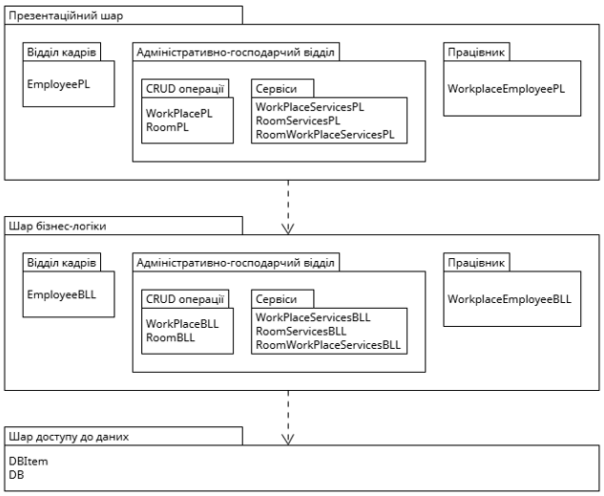
9

ДІАГРАМА СТАНІВ



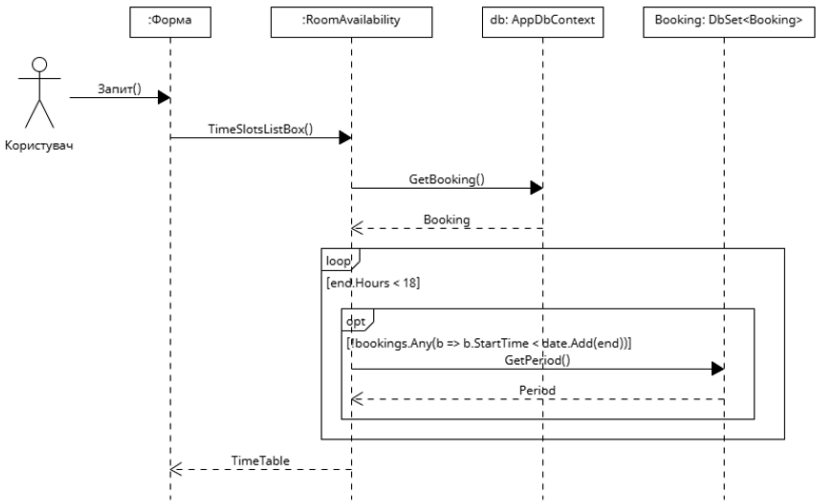
10

ДІАГРАМА ПАКЕТІВ



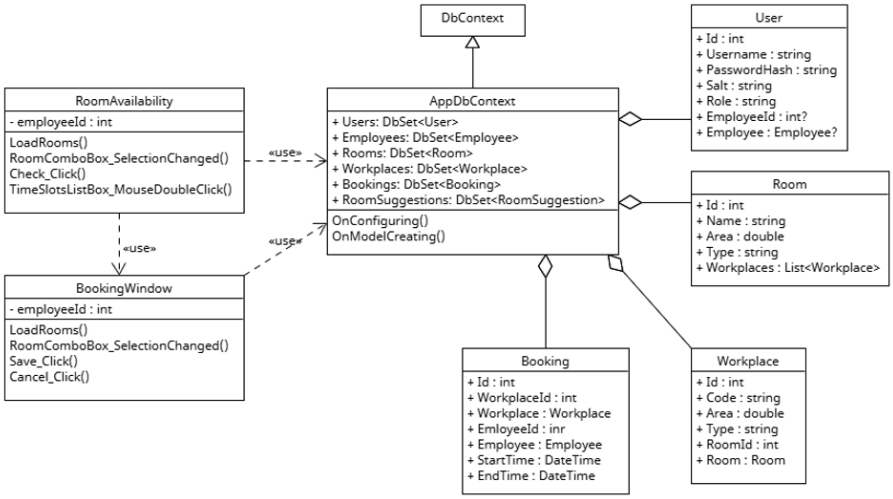
11

ДІАГРАМА ПОСЛІДОВНОСТІ



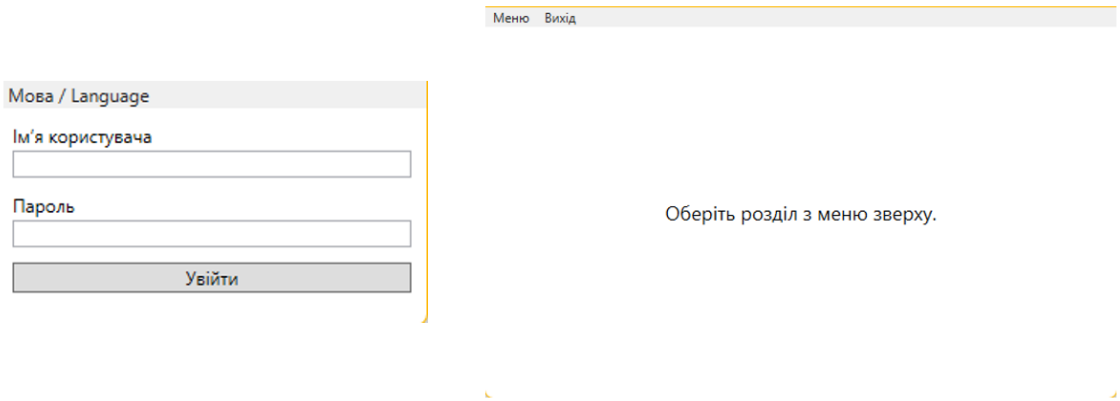
12

ДІАГРАМА КЛАСІВ



13

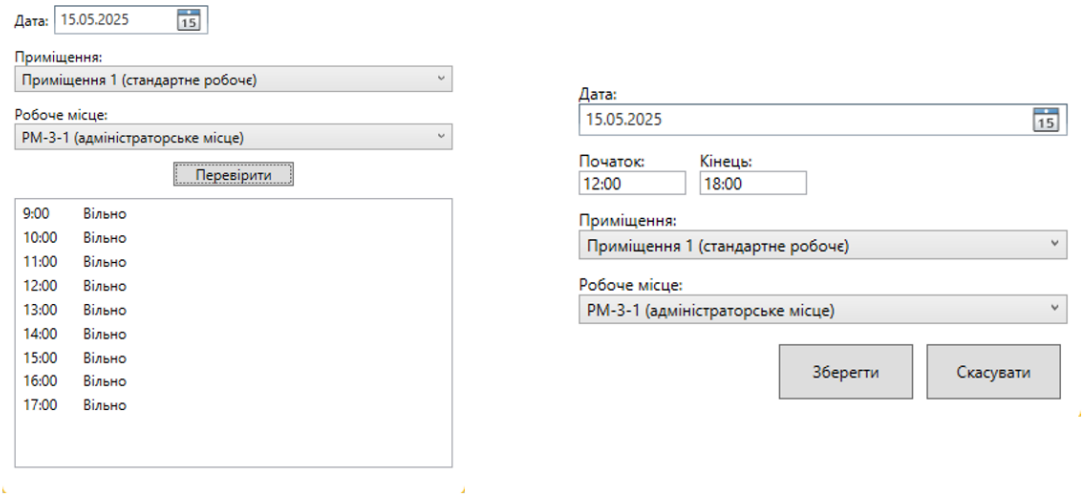
ЕКРАННІ ФОРМИ



Вікно авторизації та головне вікно

14

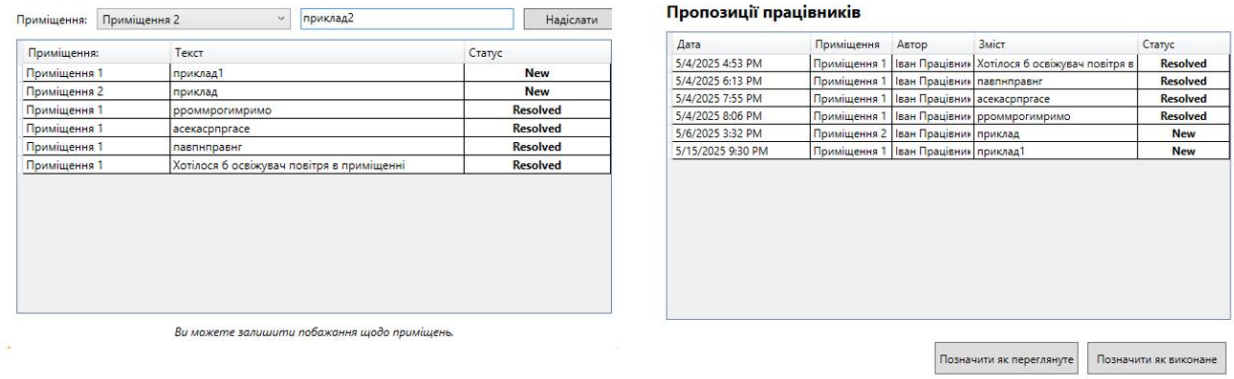
ЕКРАННІ ФОРМИ



Перегляд та бронювання робочих місць

15

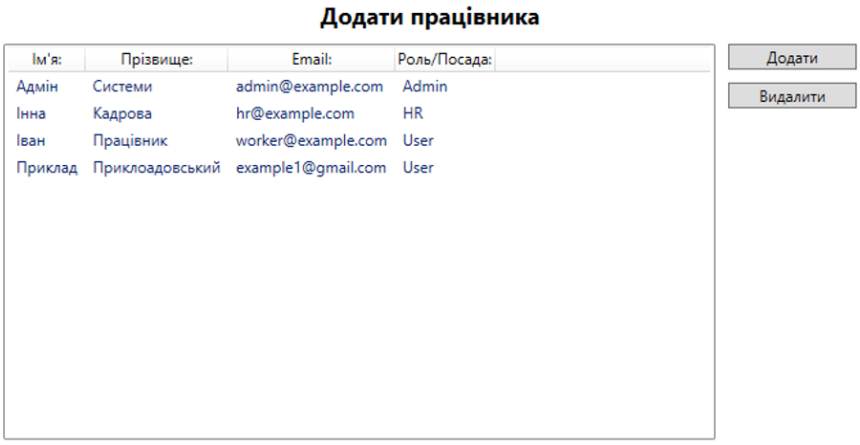
ЕКРАННІ ФОРМИ



Вікно пропозицій (з боку працівника та адміністратора)

16

ЕКРАННІ ФОРМИ



Список працівників

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Ткачищен В.М., Гаманюк І.М. Використання діаграми моделі предметної галузі для моделювання предметної галузі під час розробки застосунку з управління офісним простором: Матеріали VI Міжнародної науково-технічної конференції «Сучасний стан та перспективи розвитку IoT». 15.04.25. Державний університет інформаційно-комунікаційних технологій, Київ, Україна, 2025 (подано до друку).
2. Ткачищен В.М., Гаманюк І.М. Проектування архітектури застосунку для управління офісним простором за допомогою діаграми пакетів: Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 24.04.25. Державний університет інформаційно-комунікаційних технологій, Київ, Україна, 2025. Збірник тез К.: ДУІКТ, С.467-469.
3. Ткачищен В.М., Шевченко С.М. Захист інформації у застосунку для управління офісним простором: Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025». 15.05.25. КСУ імені Бориса Грінченка, Київ, Україна, 2025. Збірник тез К.: КУБГ, С.335.

ВИСНОВКИ

1. Проведено аналіз предметної галузі та програмних аналогів, що дозволило виявити ключові проблеми в управлінні офісним простором і визначити функціональні переваги, реалізовані у власному рішенні.
2. Сформульовано вимоги до системи, на основі яких розроблено архітектуру застосунку, структуру даних і логіку взаємодії між основними компонентами.
3. Реалізовано зручний інтерфейс користувача з підтримкою ролей (працівник, адміністрація, HR) та локалізації (українська/англійська), що забезпечує адаптивність до різних сценаріїв використання.
4. Створено повний набір функцій: бронювання, перегляд зайнятості, ведення обліку приміщень і місць, формування звітів і збір пропозицій, що охоплює потреби всіх користувацьких ролей.
5. Проведене тестування підтвердило стабільну роботу системи, відповідність поставленим вимогам і готовність програмного продукту до подальшого впровадження.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Код з LoginWindow.xaml.cs

```
using System.Windows;
using System.Windows.Controls;
using OfficeSpaceManager.Data;
using OfficeSpaceManager.Helpers;

namespace OfficeSpaceManager
{
   public partial class LoginWindow : Window
   {
      public LoginWindow()
      {
         InitializeComponent();
      }

      private void Login_Click(object sender,
RoutedEventArgs e)
      {
         var username = UsernameBox.Text;
         var password = PasswordBox.Password;

         using var db = new AppDbContext();
         var user = db.Users.FirstOrDefault(u => u.Username
== username);

         if (user != null &&
PasswordHelper.VerifyPassword(password,
user.PasswordHash, user.Salt))
         {

MessageBox.Show((string)Application.Current.Resources["L
oginSuccess"]);

            var mainWindow = new MainWindow(user);
            mainWindow.Show();
            this.Close();
         }
         else
         {

MessageBox.Show((string)Application.Current.Resources["L
oginFail"]);
         }
      }

      private void LanguageMenuItem_Click(object sender,
RoutedEventArgs e)
      {
         if (sender is MenuItem menuItem &&  menuItem.Tag
is string languageCode)
         {

LocalizationManager.ApplyLanguage(languageCode);
         }
      }

   }
}
```

Код з BookingWindow.xaml.cs

```
using System;
using System.Linq;
using System.Windows;
using OfficeSpaceManager.Data;
using OfficeSpaceManager.Models;
using Microsoft.EntityFrameworkCore;

namespace OfficeSpaceManager
{
   public partial class BookingWindow : Window
   {
      private readonly int employeeId;

      public BookingWindow(int employeeId,
                           Workplace? preselectedWorkplace = null,
                           DateTime? date = null,
                           TimeSpan? startTime = null,
                           TimeSpan? endTime = null)
      {
         InitializeComponent();
         this.employeeId = employeeId;
         LoadRooms();

         if (date.HasValue)
            BookingDatePicker.SelectedDate = date.Value;

         if (startTime.HasValue)
            StartTimeBox.Text =
startTime.Value.ToString(@"hh\:mm");

         if (endTime.HasValue)
            EndTimeBox.Text =
endTime.Value.ToString(@"hh\:mm");

         if (preselectedWorkplace != null)
         {
            using var db = new AppDbContext();
            var workplace = db.Workplaces.Include(w =>
w.Room).FirstOrDefault(w => w.Id ==
preselectedWorkplace.Id);

            if (workplace != null)
            {
               var room = db.Rooms.Find(workplace.RoomId);
               RoomComboBox.SelectedValue = room;
               RoomComboBox_SelectionChanged(null!,
null!);


WorkplaceComboBox.Dispatcher.InvokeAsync(() =>
               {
                  foreach (var item in
WorkplaceComboBox.Items)
                  {
                     var prop =
item.GetType().GetProperty("Workplace");
                     var value = prop?.GetValue(item) as
Workplace;
                     if (value?.Id == workplace.Id)
                     {
                        WorkplaceComboBox.SelectedItem =
item;
```

```
                    break;
                }
            }
        });
    }
  }
}

private void LoadRooms()
{
  using var db = new AppDbContext();
  RoomComboBox.ItemsSource = db.Rooms
    .Select(r => new
    {
      Room = r,
      Display = $"{r.Name} ({r.Type})"
    })
    .ToList();
  RoomComboBox.DisplayMemberPath = "Display";
  RoomComboBox.SelectedValuePath = "Room";
}

private void RoomComboBox_SelectionChanged(object
sender,
System.Windows.Controls.SelectionChangedEventArgs e)
{
  if (RoomComboBox.SelectedValue is Room room)
  {
    using var db = new AppDbContext();
    WorkplaceComboBox.ItemsSource =
db.Workplaces
      .Where(w => w.RoomId == room.Id)
      .Select(w => new
      {
        Workplace = w,
        Display = $"{w.Code} ({w.Type})"
      })
      .ToList();
    WorkplaceComboBox.DisplayMemberPath =
"Display";
    WorkplaceComboBox.SelectedValuePath =
"Workplace";
  }
}

private void Save_Click(object sender,
RoutedEventArgs e)
{
  if (WorkplaceComboBox.SelectedValue is not
Workplace workplace ||
    BookingDatePicker.SelectedDate is not DateTime
date ||
    !TimeSpan.TryParse(StartTimeBox.Text, out var
start) ||
    !TimeSpan.TryParse(EndTimeBox.Text, out var
end) ||
    start >= end)
  {

MessageBox.Show((string)Application.Current.Resources["F
illAllFieldsMessage"]);
    return;
  }

  if (date.Date < DateTime.Today)
  {

MessageBox.Show((string)Application.Current.Resources["I
nvalidTimeRangeMessage"]);
    return;
  }

  var startTime = date.Date + start;
  var endTime = date.Date + end;

  using var db = new AppDbContext();

  bool hasConflict = db.Bookings.Any(b =>
    b.WorkplaceId == workplace.Id &&
    b.StartTime < endTime &&
    b.EndTime > startTime);

  if (hasConflict)
  {

MessageBox.Show((string)Application.Current.Resources["
OccupiedWorkplaceMessage"]);
    return;
  }

  var booking = new Booking
  {
    EmployeeId = employeeId,
    WorkplaceId = workplace.Id,
    StartTime = startTime,
    EndTime = endTime
  };

  db.Bookings.Add(booking);
  db.SaveChanges();

MessageBox.Show((string)Application.Current.Resources["B
ookingSuccessMessage"]);
  DialogResult = true;
  Close();
}

private void Cancel_Click(object sender,
RoutedEventArgs e)
{
  DialogResult = false;
  Close();
}
  }
}
```

Код з PasswordHelper.cs

```
using System;
using System.Security.Cryptography;
using System.Text;

namespace OfficeSpaceManager.Helpers
{
  public static class PasswordHelper
  {
    public static string GenerateSalt()
    {
      byte[] saltBytes = new byte[16];
      using var rng = RandomNumberGenerator.Create();
      rng.GetBytes(saltBytes);
      return Convert.ToBase64String(saltBytes);
    }

    public static string HashPassword(string password,
string salt)
```

```
        {
            using var sha256 = SHA256.Create();
            var combined = Encoding.UTF8.GetBytes(salt +
password);
            var hash = sha256.ComputeHash(combined);
            return Convert.ToBase64String(hash);
        }

        public static bool VerifyPassword(string
enteredPassword, string storedHash, string salt)
        {
            var enteredHash = HashPassword(enteredPassword,
salt);
            return storedHash == enteredHash;
        }
    }
}
```

Код з MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using OfficeSpaceManager.Models;

namespace OfficeSpaceManager
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        private readonly User currentUser;
        public User? CurrentUser => currentUser;

        public MainWindow(User user)
        {
            InitializeComponent();
            currentUser = user;
            SetupAccess();
        }

        private void SetupAccess()
        {
            switch (currentUser.Role)
            {
                case "User":
                    EmployeesMenu.Visibility =
Visibility.Collapsed;
                    RoomsMenu.Visibility = Visibility.Collapsed;
                    WorkplacesMenu.Visibility =
Visibility.Collapsed;
                    ReportMenu.Visibility = Visibility.Collapsed;
                    AdminSuggestionMenu.Visibility =
Visibility.Collapsed;
                    break;

                case "HR":
                    RoomsMenu.Visibility = Visibility.Collapsed;
                    WorkplacesMenu.Visibility =
Visibility.Collapsed;
                    BookingMenu.Visibility = Visibility.Collapsed;
                    MyBookingsMenu.Visibility =
Visibility.Collapsed;
                    ReportMenu.Visibility = Visibility.Collapsed;
                    AvailabilityMenu.Visibility =
Visibility.Collapsed;
                    MySuggestionsMenu.Visibility =
Visibility.Collapsed;
                    AdminSuggestionMenu.Visibility =
Visibility.Collapsed;
                    break;

                case "Admin":
                    EmployeesMenu.Visibility =
Visibility.Collapsed;
                    BookingMenu.Visibility = Visibility.Collapsed;
                    MyBookingsMenu.Visibility =
Visibility.Collapsed;
                    AvailabilityMenu.Visibility =
Visibility.Collapsed;
                    MySuggestionsMenu.Visibility =
Visibility.Collapsed;
                    break;

                default:

MessageBox.Show((string)Application.Current.Resources["
UserRoleUnknownMessage"]);
                    Close();
                    break;
            }
        }

        private void EmployeesMenu_Click(object sender,
RoutedEventArgs e)
        {
            new EmployeeListWindow().ShowDialog();
        }

        private void RoomsMenu_Click(object sender,
RoutedEventArgs e)
        {
            new RoomListWindow().ShowDialog();
        }

        private void WorkplacesMenu_Click(object sender,
RoutedEventArgs e)
        {
            new WorkplaceListWindow().ShowDialog();
        }
        private void Availability_Click(object sender,
RoutedEventArgs e)
        {
            new
RoomAvailabilityWindow(currentUser.EmployeeId.Value).S
howDialog();
        }


        private void BookingMenu_Click(object sender,
RoutedEventArgs e)
        {
            if (currentUser.EmployeeId == null)
            {
```

```
MessageBox.Show((string)Application.Current.Resources["
UserNotLinkedMessage"]);
            return;
        }

        var bookingWindow = new
BookingWindow(currentUser.EmployeeId.Value);
        bookingWindow.ShowDialog();
    }

    private void MyBookingsMenu_Click(object sender,
RoutedEventArgs e)
    {
        if (currentUser.EmployeeId != null)
        {
            new
MyBookingsWindow(currentUser.EmployeeId.Value).Show
Dialog();
        }
    }
    private void OpenReport_Click(object sender,
RoutedEventArgs e)
    {
        new ReportWindow().ShowDialog();
    }



    private void ExitMenu_Click(object sender,
RoutedEventArgs e)
    {
        this.Close();
    }
    private void LogoutMenu_Click(object sender,
RoutedEventArgs e)
    {
        var loginWindow = new LoginWindow();
        loginWindow.Show();
        this.Close();
    }
    private void OpenSettings_Click(object sender,
RoutedEventArgs e)
    {
        var settingsWindow = new SettingsWindow();
        settingsWindow.ShowDialog();
    }
    private void OpenMySuggestions_Click(object sender,
RoutedEventArgs e)
    {
        var window = new
MySuggestionsWindow(CurrentUser.EmployeeId.Value);
        window.ShowDialog();
    }

    private void AdminSuggestionMenu_Click(object
sender, RoutedEventArgs e)
    {
        var window = new AdminSuggestionWindow();
        window.ShowDialog();
    }


  }
}
```

Код з RoomAvailabilityWindow.xaml.cs

```
using System;
using System.Linq;
using System.Windows;
using OfficeSpaceManager.Data;
using OfficeSpaceManager.Models;

namespace OfficeSpaceManager
{
  public partial class RoomAvailabilityWindow : Window
  {
    private readonly int employeeId;

    public RoomAvailabilityWindow(int employeeId)
    {
      InitializeComponent();
      this.employeeId = employeeId;
      LoadRooms();
      DatePicker.DisplayDateStart = DateTime.Today;
      DatePicker.SelectedDate = DateTime.Today;
    }

    private void LoadRooms()
    {
      using var db = new AppDbContext();
      RoomComboBox.ItemsSource = db.Rooms
        .Select(r => new
        {
          Room = r,
          Display = $"{r.Name} ({r.Type})"
        })
        .ToList();
      RoomComboBox.DisplayMemberPath = "Display";
      RoomComboBox.SelectedValuePath = "Room";
    }
    private void RoomComboBox_SelectionChanged(object
sender,
System.Windows.Controls.SelectionChangedEventArgs e)
    {
      if (RoomComboBox.SelectedValue is Room room)
      {
        using var db = new AppDbContext();
        WorkplaceComboBox.ItemsSource =
db.Workplaces
          .Where(w => w.RoomId == room.Id)
          .Select(w => new
          {
            Workplace = w,
            Display = $"{w.Code} ({w.Type})"
          })
          .ToList();
        WorkplaceComboBox.DisplayMemberPath =
"Display";
        WorkplaceComboBox.SelectedValuePath =
"Workplace";
      }
    }

    private void Check_Click(object sender,
RoutedEventArgs e)
    {
      if (WorkplaceComboBox.SelectedValue is not
Workplace workplace ||
        DatePicker.SelectedDate is not DateTime date)
      {

MessageBox.Show((string)Application.Current.Resources["R
oomAvailabilityMissingSelection"]);
        return;
      }
```

```
        if (date.Date < DateTime.Today)
        {
            MessageBox.Show((string)Application.Current.Resources["RoomAvailabilityPastDateError"]);
            return;
        }

        using var db = new AppDbContext();
        var bookings = db.Bookings
            .Where(b => b.WorkplaceId == workplace.Id && b.StartTime.Date == date.Date)
            .ToList();

        var availability = Enumerable.Range(9, 9)
            .Select(h =>
            {
                var time = new DateTime(date.Year, date.Month, date.Day, h, 0, 0);
                bool isBooked = bookings.Any(b => time >= b.StartTime && time < b.EndTime);
                string status = isBooked
                    ? (string)Application.Current.Resources["StatusOccupied"]
                    : (string)Application.Current.Resources["StatusFree"];
                return new { Hour = h, Status = status };
            })
            .ToList();


        TimeSlotsListBox.ItemsSource = availability;
    }

    private void TimeSlotsListBox_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)
    {
        var selectedItem = TimeSlotsListBox.SelectedItem;
        if (selectedItem == null)
            return;

        dynamic slot = selectedItem;

        var occupiedLabel = Application.Current.Resources["StatusOccupied"] as string;
        if (slot.Status == occupiedLabel)
            return;

        if (WorkplaceComboBox.SelectedValue is not Workplace workplace ||
            DatePicker.SelectedDate is not DateTime date)
            return;

        int hour = slot.Hour;
        TimeSpan start = TimeSpan.FromHours(hour);
        TimeSpan end = start.Add(TimeSpan.FromHours(1));

        using var db = new AppDbContext();
        var bookings = db.Bookings
            .Where(b => b.WorkplaceId == workplace.Id && b.StartTime.Date == date.Date)
            .OrderBy(b => b.StartTime)
            .ToList();

        while (end.Hours < 18 && !bookings.Any(b => b.StartTime < date.Add(end) && b.EndTime > date.Add(end - TimeSpan.FromHours(1))))
        {
            end = end.Add(TimeSpan.FromHours(1));
        }

        var bookingWindow = new BookingWindow(employeeId, workplace, date, start, end);
        bookingWindow.ShowDialog();
    }

  }
}
```

Код з WorkplaceFormWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using OfficeSpaceManager.Data;
using OfficeSpaceManager.Models;

namespace OfficeSpaceManager
{
  /// <summary>
  /// Interaction logic for WorkplaceFormWindow.xaml
  /// </summary>
  public partial class WorkplaceFormWindow : Window
  {
    private Workplace? existingWorkplace;

    public WorkplaceFormWindow(Workplace? workplace = null)
    {
      InitializeComponent();
      LoadRooms();
      LoadTypes();

      if (workplace != null)
      {
        Title = (string)Application.Current.Resources["EditWorkplaceTitle"];
        existingWorkplace = workplace;
        CodeBox.Text = workplace.Code;
        AreaBox.Text = workplace.Area.ToString();
        RoomComboBox.SelectedValue = workplace.RoomId;
        TypeBox.Text = workplace.Type;
      }
    }

    private void LoadRooms()
    {
      using var db = new AppDbContext();
      RoomComboBox.ItemsSource = db.Rooms.ToList();
    }

    private void LoadTypes()
    {
```

```
        using var db = new AppDbContext();
        var types = db.Workplaces
            .Select(w => w.Type)
            .Where(t => !string.IsNullOrWhiteSpace(t))
            .Distinct()
            .ToList();

        TypeBox.ItemsSource = types;
    }

    private void Save_Click(object sender,
RoutedEventArgs e)
    {
        var code = CodeBox.Text.Trim();
        var type = TypeBox.Text.Trim();
        var selectedRoomId =
(int?)RoomComboBox.SelectedValue;

        if (string.IsNullOrWhiteSpace(code) ||
string.IsNullOrWhiteSpace(type) || selectedRoomId == null)
        {

MessageBox.Show((string)Application.Current.Resources["
WorkplaceFormMissingFields"]);
            return;
        }

        if (!double.TryParse(AreaBox.Text, out var wpArea)
|| wpArea <= 0)
        {

MessageBox.Show((string)Application.Current.Resources["
AreaInvalidMessage"]);
            return;
        }

        using var db = new AppDbContext();
        var room = db.Rooms.Find(selectedRoomId.Value);
        if (room == null)
        {

MessageBox.Show((string)Application.Current.Resources["R
oomNotFoundMessage"]);
            return;
        }

        var totalUsedArea = db.Workplaces
            .Where(w => w.RoomId == room.Id &&
(existingWorkplace == null || w.Id != existingWorkplace.Id))
            .Sum(w => w.Area);

        if (totalUsedArea + wpArea > room.Area)
        {
            var remaining = room.Area - totalUsedArea;
            var msg =
string.Format((string)Application.Current.Resources["NotEn
oughRoomAreaMessage"], remaining);
            MessageBox.Show(msg);
            return;
        }

        if (existingWorkplace == null)
        {
            var newWorkplace = new Workplace
            {
                Code = code,
                Area = wpArea,
                RoomId = room.Id,
                Type = type
            };
            db.Workplaces.Add(newWorkplace);
        }
        else
        {
            var toUpdate =
db.Workplaces.Find(existingWorkplace.Id);
            if (toUpdate != null)
            {
                toUpdate.Code = code;
                toUpdate.Area = wpArea;
                toUpdate.RoomId = room.Id;
                toUpdate.Type = type;
            }
        }

        db.SaveChanges();
        DialogResult = true;
        Close();
    }

    private void Cancel_Click(object sender,
RoutedEventArgs e)
    {
        DialogResult = false;
        Close();
    }
  }
}
```