

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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-системи бронювання матеріально-технічних ресурсів навчального закладу»

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*

_____ Владислав ЛОРЕНЦ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

_____ Владислав ЛОРЕНЦ

Керівник: _____ Сергій КОМΠΑН

канд. фіз.-мат. наук.

Рецензент: _____

**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

« ____ » _____ 2026 р.

**ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Лоренцу Владиславу Яношовичу

1. Тема кваліфікаційної роботи: «Розробка web-системи бронювання матеріально-технічних ресурсів навчального закладу»

керівник кваліфікаційної роботи д-к ІТ, кан. фіз.-мат. наук Сергій КОМΠΑН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р.

2. Строк подання кваліфікаційної роботи «29» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи: web-система бронювання матеріально-технічних ресурсів навчального закладу Book It!, розроблена з використанням Django 6.0 [1], Python 3.13 [2], SQLite [3], HTML5/CSS3 [8], JavaScript (AJAX), Unicorn 25.3.0 [4], WhiteNoise 6.12.0 [5]; розгорнута на платформі Render [6] із CI/CD-конвеєром на базі GitHub Actions [7].

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

1. Аналіз предметної галузі: огляд та порівняння існуючих систем бронювання ресурсів (Skedda, Microsoft Bookings, Robin), визначення вимог до системи.

2. Засоби реалізації: обґрунтування вибору мови програмування Python, фреймворку Django, бази даних SQLite, клієнтських технологій та інфраструктурних інструментів

3. Проектування системи: розробка архітектури на основі шаблону MVT, діаграм варіантів використання, ER-моделі бази даних, діаграм послідовності та розгортання.

4. Реалізація та тестування: опис реалізації ключових модулів системи, проведення функціонального та нефункціонального тестування.

5. Перелік графічного матеріалу :

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання (Use Case Diagram).
5. ER-діаграма бази даних.
6. Діаграма послідовності (Sequence Diagram).
7. Екранні форми.
8. Апробація результатів дослідження.

5. Дата видачі завдання: “23” лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задач практики. Формування календарного плану. Аналіз предметної галузі. Консультація з науковим керівником.	23.03-25.03.2026	
2	Огляд та аналіз літературних джерел, існуючих засобів розв'язання задач бронювання ресурсів навчального закладу. Консультація з науковим керівником.	26.03-28.03.2026	
3	Визначення об'єкта, предмета, мети кваліфікаційної роботи бакалавра та постановка завдань. Консультація з науковим керівником.	29.03-01.04.2026	
4	Аналіз та порівняння існуючого програмного забезпечення для бронювання ресурсів: Skedda, Microsoft Bookings, Robin.	02.04-04.04.2026	
5	Проектування архітектури системи. Розробка UML-діаграм (Use Case, ER-діаграма, діаграма послідовності).	05.04-06.04.2026	
6	Розробка серверної частини: Django-проект, моделі даних (Resource, Booking, UserProfile, ActionLog).	07.04-14.04.2026	

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
7	Реалізація системи автентифікації та розмежування доступу за ролями (студент, викладач, адміністратор).	15.04-16.04.2026	
8	Розробка функціоналу бронювання: перевірка конфліктів, CRUD-операції, логування дій через Django Signals.	17.04-18.04.2026	
9	Розробка клієнтської частини: HTML-шаблони, CSS-стили, AJAX-фільтрація ресурсів. Консультація з науковим керівником.	19.04-20.04.2026	
10	Налаштування розгортання на Render через GitHub CI/CD.	21.04-21.04.2026	
11	Тестування застосунку	22.04-23.04.2026	
12	Підготовка презентаційних матеріалів для практики. Написання тез наукової доповіді. Консультація з науковим керівником.	23.04-28.04.2026	
13	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2026	
14	Розробка демонстраційних матеріалів	06.05-11.05.2026	
15	Попередній захист роботи	12.05-22.05.2026	

Здобувач вищої освіти

(підпис)

Владислав ЛОРЕНЦ

Керівник

кваліфікаційної роботи

(підпис)

Сергій КОМΠΑН

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 10 табл., 6 рис., 17 джерел.

Мета роботи: автоматизація бронювання аудиторій навчального закладу за допомогою розробки web-системи для бронювання аудиторій з автоматичною перевіркою конфліктів часу та розмежуванням доступу користувачів за ролями.

Об'єкт дослідження: процес управління та бронювання матеріально-технічних ресурсів навчального закладу.

Предмет дослідження: методи та програмні засоби реалізації web-системи бронювання аудиторій з використанням фреймворку Django та реляційної системи управління баз даних SQLite.

Короткий зміст роботи: у роботі проаналізовано предметну галузь та існуючі системи бронювання ресурсів (Skedda, Microsoft Bookings, Robin), визначено їх переваги та недоліки стосовно освітнього середовища. Обґрунтовано вибір технологічного стеку: Python 3.13, Django 6.0, SQLite, HTML5/CSS3, JavaScript з AJAX. Спроектowano архітектуру системи за шаблоном MVT, розроблено ER-модель бази даних, діаграми варіантів використання, послідовності та розгортання. Реалізовано web-застосунок Book It! з трирівневою рольовою моделлю (адміністратор, викладач, студент), автоматичною перевіркою конфліктів бронювань, журналом дій на основі Django Signals та календарним інтерфейсом. Застосунок розгорнуто на платформі Render із налаштованим CI/CD-конвеєром через GitHub Actions. Проведено функціональне та нефункціональне тестування системи.

Сферою використання застосунку є навчальні заклади, що потребують автоматизації процесу резервування матеріально-технічної бази.

КЛЮЧОВІ СЛОВА: DJANGO, PYTHON, WEB-СИСТЕМА, БРОНЮВАННЯ РЕСУРСІВ, SQLite, MVT, CI/CD, GITHUB ACTIONS, RENDER, GUNICORN, АВТЕНТИФІКАЦІЯ, РОЛЬОВА МОДЕ

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ.....	8
ВСТУП	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ	12
1.1 Огляд предметної галузі та постановка задачі	12
1.2 Аналіз існуючих систем бронювання.....	15
1.2.1 Skedda.....	15
1.2.2 Microsoft Bookings	16
1.2.3 Robin.....	18
1.3 Порівняльний аналіз та висновки до розділу 1	19
Висновки до розділу 1.	21
2 ЗАСОБИ РЕАЛІЗАЦІЇ	22
2.1 Середовище розробки.....	22
2.2 Мова програмування Python 3.13	23
2.3 Web-фреймворк Django 6.0.....	23
2.4 База даних SQLite	25
2.5 Клієнтські технології: HTML5, CSS3, JavaScript, AJAX	26
2.6 Інфраструктурні засоби: Gunicorn, WhiteNoise, Render.....	27
2.6.1 Gunicorn 25.3.0	27
2.6.2 WhiteNoise 6.12.0	28
2.6.3 Render	28
2.7 Система контролю версій та CI/CD	29
Висновки до розділу 2.	30
3 ПРОЄКТУВАННЯ СИСТЕМИ.....	31
3.1 Архітектура системи на основі шаблону MVT.....	31
3.2 Діаграма варіантів використання	33
3.3 ER-модель бази даних	35
3.4 Діаграма послідовності	40
3.5 Діаграма розгортання	42
4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ	45

4.1 Структура проекту та налаштування Django	45
4.2 Реалізація моделей бази даних	46
4.3 Реалізація бізнес-логіки та маршрутизації.....	49
4.4 Клієнтська частина та AJAX-взаємодія.....	51
4.5 Розгортання системи та CI/CD-конвеєр	53
4.6 Тестування системи.....	55
4.6.1 Функціональне тестування	55
4.6.2 Нефункціональне тестування	56
Висновки до розділу 4.	57
ВИСНОВКИ.....	58
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ	
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ	

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AJAX – Asynchronous JavaScript and XML – технологія асинхронного обміну даними між браузером і сервером

API – Application Programming Interface – інтерфейс програмування застосунків

CI/CD – Continuous Integration / Continuous Delivery – безперервна інтеграція та доставка програмного забезпечення

CSS – Cascading Style Sheets – каскадні таблиці стилів

ER – Entity-Relationship – сутність-зв'язок (модель бази даних)

HTML – HyperText Markup Language – мова розмітки гіпертексту

HTTP – HyperText Transfer Protocol – протокол передачі гіпертексту

JS – JavaScript – мова програмування для клієнтської частини web-застосунків

MVT – Model-View-Template – архітектурний шаблон фреймворку Django

MVC – Model-View-Controller – архітектурний шаблон програмного забезпечення

ORM – Object-Relational Mapping – об'єктно-реляційне відображення

OS – Operating System – операційна система

СУБД – Система управління базами даних

UML – Unified Modeling Language – уніфікована мова моделювання

URL – Uniform Resource Locator – уніфікований локатор ресурсів

WSGI – Web Server Gateway Interface – інтерфейс шлюзу web-сервера для Python

ВСТУП

Актуальність: сучасна освітня установа функціонує в умовах постійно зростаючого попиту на ефективне управління матеріально-технічними ресурсами. Аудиторії, лабораторії, конференц-зали, мультимедійне обладнання та інша інфраструктура навчального закладу є обмеженими ресурсами, що потребують чіткої координації між студентами, викладачами та адміністрацією. Відсутність автоматизованої системи бронювання призводить до конфліктів розкладів, нераціонального використання приміщень і значних часових витрат на організаційну роботу. За оцінками дослідників у сфері управління освітніми закладами, приблизно до 30% робочого часу адміністративного персоналу витрачається на координацію використання матеріально-технічної бази, а конфлікти при ручному плануванні призводять до того, що приблизно від 15% до 25% аудиторного фонду використовується неефективно. Впровадження автоматизованих систем бронювання дозволяє скоротити організаційні витрати часу в 3-5 разів і підвищити завантаженість ресурсів приблизно до 85-90%.

Актуальність теми зумовлена тим, що більшість навчальних закладів України досі вирішують задачу бронювання ресурсів за допомогою паперових журналів або загальних електронних таблиць, що не забезпечують ні зручності, ні достатнього рівня контролю. Водночас світовий ринок пропонує комерційні рішення – такі як Skedda, Microsoft Bookings, Robin – що орієнтовані переважно на корпоративний сектор і не адаптовані до специфіки освітніх установ, зокрема до ієрархічної структури доступу, прив'язки до академічного розкладу та необхідності інтеграції з внутрішніми інформаційними системами закладу. Таким чином, існує потреба у розробці спеціалізованого програмного рішення, що враховуватиме особливості освітнього середовища: академічний розклад пар, ієрархічну структуру доступу та специфіку використання різних типів ресурсів – від звичайних аудиторій до конференц-залів, що потребують окремого підтвердження

адміністратором.

Метою кваліфікаційної роботи є розробка web-системи бронювання матеріально-технічних ресурсів навчального закладу Book It!, яка забезпечує зручний інтерфейс для подання заявок на бронювання, автоматизовану перевірку конфліктів, розмежування прав доступу між ролями користувачів та відображення актуального стану зайнятості ресурсів у режимі реального часу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. провести аналіз предметної галузі та існуючих рішень для бронювання ресурсів, виявити їх переваги та недоліки;
2. обґрунтувати вибір технологічного стеку та інструментальних засобів реалізації системи;
3. спроектувати архітектуру системи з використанням шаблону MVT, розробити діаграми варіантів використання, ER-модель бази даних, діаграми послідовності та розгортання;
4. реалізувати web-систему з використанням фреймворку Django 6.0, мови програмування Python 3.13, бази даних SQLite, клієнтської частини на основі HTML5/CSS3 та JavaScript з підтримкою AJAX-запитів;
5. організувати хмарне розгортання системи на платформі Render із застосуванням Gunicorn 25.3.0 та WhiteNoise 6.12.0, а також налаштувати CI/CD-конвеєр засобами GitHub Actions;
6. провести тестування функціональних та нефункціональних вимог до системи й оцінити якість отриманого програмного продукту.

Об'єктом дослідження є процес управління бронюванням матеріально-технічних ресурсів навчального закладу.

Предметом дослідження є методи, засоби та технології розробки web-систем, що застосовуються для автоматизації бронювання ресурсів із забезпеченням розмежування прав доступу та запобігання конфліктам резервування.

Наукова новизна роботи полягає у розробці механізму автоматичного визначення ролі користувача на основі домену корпоративної електронної

пошти, а також у реалізації дворівневої перевірки конфліктів бронювань – на рівні форми при поданні заявки та повторно на рівні View при підтвердженні адміністратором, що гарантує консистентність даних навіть при одночасній роботі кількох адміністраторів.

Практична цінність роботи полягає у створенні готового до розгортання програмного продукту, який може бути впроваджений у будь-якому навчальному закладі без значних витрат на адаптацію. Система Book It! забезпечує повний цикл управління бронюванням: від реєстрації ресурсів адміністратором до подання, розгляду та підтвердження заявок, – а також надає зручний інтерфейс перегляду зайнятості ресурсів у форматі календарної сітки.

Система реалізована у вигляді web-застосунку, що не потребує встановлення клієнтського програмного забезпечення і доступна з будь-якого пристрою з браузером. Завдяки хмарному розгортанню на платформі Render і налаштованому CI/CD-конвеєру на базі GitHub Actions забезпечується безперервна доставка оновлень і висока доступність сервісу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Огляд предметної галузі та постановка задачі

Сучасний навчальний заклад є складною організаційною структурою, що включає велику кількість матеріально-технічних ресурсів: аудиторії, лабораторії, конференц-зали, мультимедійне та комп'ютерне обладнання, спортивні споруди тощо. Ефективне управління цими ресурсами є однією з ключових умов безперебійного навчального процесу та раціонального використання інфраструктури закладу.

Традиційно задача бронювання ресурсів вирішується за допомогою паперових журналів або загальнодоступних електронних таблиць. Такий підхід має низку суттєвих недоліків: відсутність централізованого контролю, висока ймовірність конфліктів при одночасному резервуванні одного ресурсу різними користувачами, складність відстеження поточного стану зайнятості, неможливість отримати актуальну інформацію дистанційно, а також значні часові витрати адміністративного персоналу на організаційну роботу.

Автоматизована web-система бронювання здатна вирішити всі перераховані проблеми. Вона забезпечує єдину точку керування ресурсами, автоматично перевіряє наявність конфліктів при поданні нових заявок, надає зручний інтерфейс для перегляду розкладу зайнятості та зберігає повну історію операцій. Доступність через браузер без встановлення клієнтського програмного забезпечення робить такі системи особливо привабливими для освітнього середовища.

Предметна галузь характеризується такими ключовими сутностями та процесами:

ресурс — матеріально-технічний об'єкт, що підлягає бронюванню (аудиторія, обладнання тощо), з атрибутами: назва, тип, місткість, опис, статус доступності;

користувач – суб'єкт системи, що має роль (адміністратор, викладач, студент) і відповідний рівень доступу до функціональності;

бронювання – резервування конкретного ресурсу на певну дату та часовий слот, пов'язане з конкретним користувачем і поточним статусом (очікує, підтверджено, скасовано);

конфлікт бронювань – ситуація, коли два або більше запити претендують на один ресурс в один і той же час; система повинна автоматично виявляти та запобігати таким конфліктам;

журнал дій – протокол усіх операцій у системі, що забезпечує відстежуваність та прозорість управління ресурсами.

На підставі аналізу предметної галузі можна сформулювати основні вимоги до розроблюваної системи. Функціональні вимоги включають: реєстрацію та автентифікацію користувачів з розмежуванням ролей; перегляд каталогу ресурсів з фільтрацією; відображення розкладу зайнятості в форматі календарної сітки; подання, підтвердження та скасування заявок на бронювання; автоматичну перевірку конфліктів; ведення журналу дій. До нефункціональних вимог належать: доступність через web-браузер без встановлення додаткового ПЗ; адаптивний інтерфейс; час відгуку системи не більше 2 секунд; захист від несанкціонованого доступу які наведено в таблицях 1.1 та 1.2.

Таблиця 1.1

Функціональні вимоги до системи.

Функціональні вимоги	Пріоритет
Реєстрація та автентифікація з автоматичним визначенням ролі за доменом корпоративної пошти	Високий
Реалізувати розмежування доступу за ролями (студент, викладач, адміністратор), які надають відповідні права доступу та можливість перегляду списку доступних ресурсів залежно від ролі	Критичний

Продовження таблиці 1.1

Функціональні вимоги	Пріоритет
Перегляд каталогу ресурсів з фільтрацією за типом, кафедрою та місткістю	Високий
Відображення денного та тижневого розкладу зайнятості ресурсу	Високий
Подання заявки на бронювання з автоматичною перевіркою конфліктів часу	Критичний
Швидке бронювання пари одним кліком з тижневого розкладу	Середній
Підтвердження та відхилення заявок адміністратором	Високий
Скасування та видалення власних бронювань користувачем	Високий
Ведення журналу дій з можливістю перегляду адміністратором	Середній
Управління ресурсами через адміністративну панель	Високий

Таблиця 1.2

Нефункціональні вимоги до системи

Нефункціональні вимоги до системи	Метрика
Забезпечити час відгуку системи при створенні або перегляді бронювання	не більше 2 секунд
Забезпечити заборону створення бронювання при наявності конфлікту часу	перевірка на рівні серверної логіки
Забезпечити контроль доступу через систему ролей Django	розмежування прав доступу за ролями Студент, Викладач, Адмін

Продовження таблиці 1.2

Нефункціональні вимоги до системи	Метрика
Забезпечити зручність використання та виконання основних дій	не більше 3–4 кліків

1.2 Аналіз існуючих систем бронювання

На ринку існує чимало рішень для бронювання ресурсів, орієнтованих на різні сектори: корпоративний, готельний, медичний та освітній. Для аналізу обрано три найбільш релевантних комерційних продукти, що широко застосовуються в організаціях, схожих за структурою з навчальним закладом: Skedda, Microsoft Bookings та Robin. Критерії порівняння визначено виходячи з вимог, сформульованих у підрозділі 1.1.

1.2.1 Skedda

Skedda – хмарна платформа для бронювання приміщень та ресурсів, що позиціонується як рішення для університетів, коворкінгів, спортивних центрів та офісів [11]. Система пропонує інтуїтивний інтерфейс з інтерактивними планами поверхів, гнучке налаштування правил бронювання та автоматичну обробку запитів.

Ключові можливості Skedda включають: візуалізацію приміщень на інтерактивній карті, налаштування квот і обмежень на бронювання, підтримку повторюваних резервувань, інтеграцію з Google Calendar та Microsoft 365, а також розширену аналітику використання ресурсів. Система має мобільний додаток для iOS та Android.

З точки зору застосування в навчальному закладі Skedda має кілька суттєвих обмежень. По-перше, система є комерційною з ціноутворенням на основі кількості ресурсів і користувачів, що може бути фінансово обтяжливим для бюджетних установ. По-друге, Skedda не передбачає ієрархічної рольової моделі, характерної для освітніх закладів з чіткою структурою (адміністрація,

викладачі, студенти). По-третє, відсутня можливість інтеграції з внутрішніми інформаційними системами закладу без додаткового розроблення API-адаптерів. Налаштування системи під специфіку конкретного закладу вимагає значних зусиль.

Цінова модель Skedda передбачає щомісячну оплату залежно від кількості просторів (spaces): базовий план від 99 USD на місяць за 10 ресурсів, розширений – від 199 USD. Для навчального закладу з десятками аудиторій та лабораторій річні витрати можуть перевищувати 2 400 USD, що є неприйнятним для більшості бюджетних освітніх установ України. Безкоштовний тариф обмежено п'ятьма ресурсами і не включає розширені функції керування доступом (див. рис. 1.1).

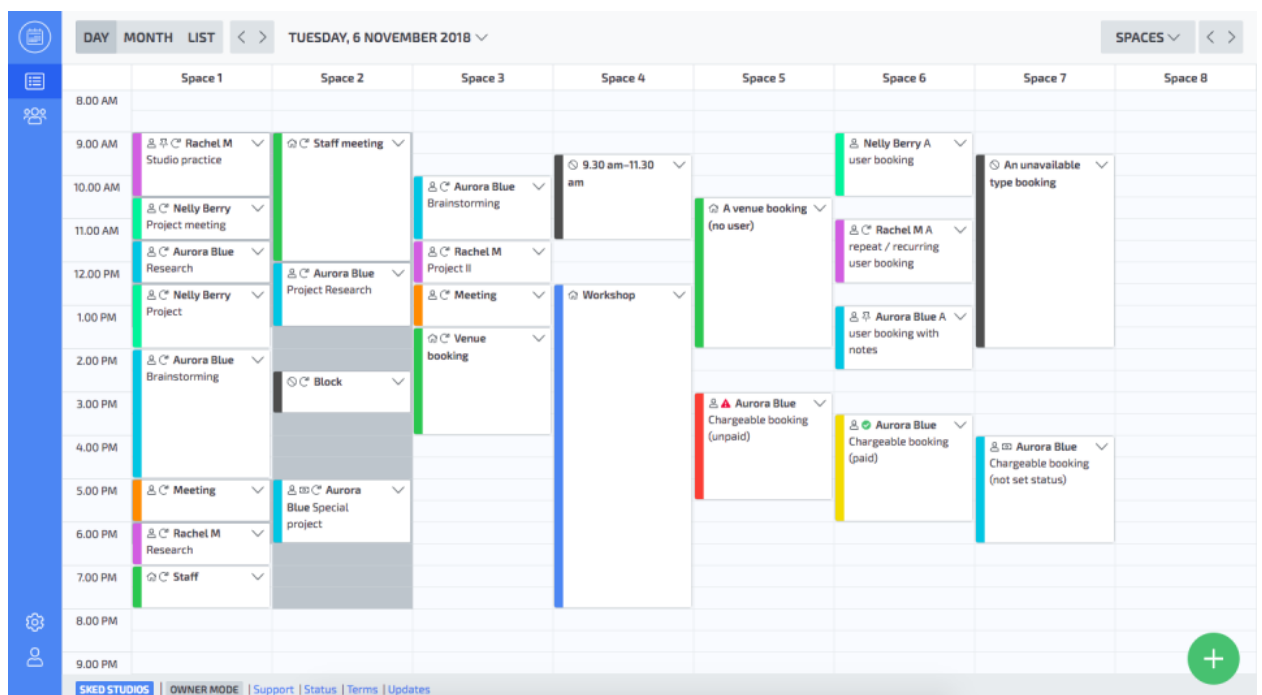


Рис. 1.1 Інтерфейс системи Skedda (головна сторінка бронювання)

1.2.2 Microsoft Bookings

Microsoft Bookings – компонент пакету Microsoft 365, призначений для планування зустрічей і бронювання ресурсів [12]. Система тісно інтегрована з Microsoft Teams, Outlook Calendar та іншими продуктами екосистеми Microsoft, що робить її природним вибором для організацій, що вже

використовують Office 365.

Функціональність Microsoft Bookings охоплює: створення сторінок бронювання для різних типів послуг і ресурсів, автоматичне надсилання підтверджень і нагадувань електронною поштою, управління робочим розкладом і доступністю, підтримку групових та індивідуальних бронювань. Інтерфейс відрізняється простотою і знайомий користувачам продуктів Microsoft.

Разом з тим, Microsoft Bookings має принципові обмеження для задач навчального закладу. Система розроблена насамперед для бронювання послуг та зустрічей, а не фізичних ресурсів і приміщень. Відсутні механізми відображення поточної зайнятості в форматі календарної сітки за ресурсами. Система не підтримує складну ієрархію дозволів з трьома і більше рівнями доступу. Обов'язкова наявність корпоративної підписки Microsoft 365 є значним фінансовим та організаційним бар'єром для більшості українських навчальних закладів.

З технічної точки зору Microsoft Bookings має обмеження на кількість послуг і типів бронювань, що ускладнює моделювання різнотипних ресурсів навчального закладу. Система не надає публічного API для читання поточної зайнятості приміщень у реальному часі, що унеможлиблює інтеграцію з іншими інформаційними системами закладу. Крім того, обов'язкова прив'язка до Active Directory та корпоративних облікових записів Microsoft ускладнює реєстрацію студентів, які не мають корпоративної пошти університету (див. рис. 1.2).

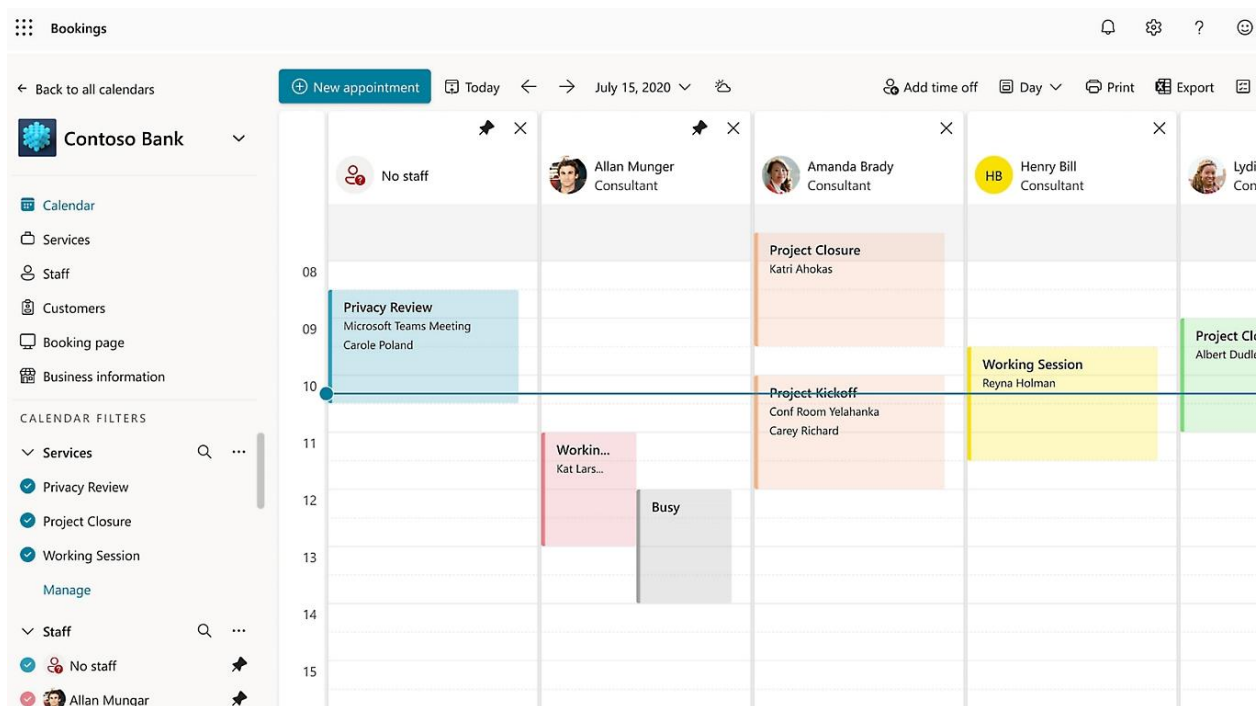


Рис. 1.2 Інтерфейс системи Microsoft Bookings (календар бронювань)

1.2.3 Robin

Robin – платформа для управління робочим простором (workplace management), що спеціалізується на гібридних офісах та корпоративному середовищі [13]. Система надає інструменти для бронювання робочих місць, переговорних кімнат і обладнання, відстеження відвідуваності та аналізу використання простору.

Robin відрізняється розвиненою аналітикою: система збирає дані про реальне використання ресурсів, будує звіти та прогнози, що допомагає оптимізувати розподіл простору. Платформа підтримує інтеграцію з популярними корпоративними інструментами: Slack, Google Workspace, Microsoft 365, Zoom. Передбачено API для кастомних інтеграцій.

Для застосування в навчальному закладі Robin має такі недоліки: система повністю орієнтована на корпоративний сектор і не враховує специфіку освітнього процесу; висока вартість ліцензування; інтерфейс і логіка роботи розраховані на «гнучкий офіс», а не на академічне середовище з фіксованим розкладом; відсутність україномовного інтерфейсу та підтримки; відсутні механізми управління заявками з підтвердженням, характерні для

освітніх закладів.

Цінова модель Robin базується на кількості користувачів: від 3 USD на користувача на місяць для базового плану. Для навчального закладу з кількома сотнями активних користувачів щомісячні витрати можуть становити від 600 до 1500 USD, що є економічно недоцільним. Відсутня безкоштовна версія або академічна ліцензія. Додатково Robin не підтримує концепцію «пари» як фіксованого академічного часового слоту, що є стандартом для українських навчальних закладів, і не дозволяє налаштувати розклад відповідно до академічного календаря (див. рис. 1.3).

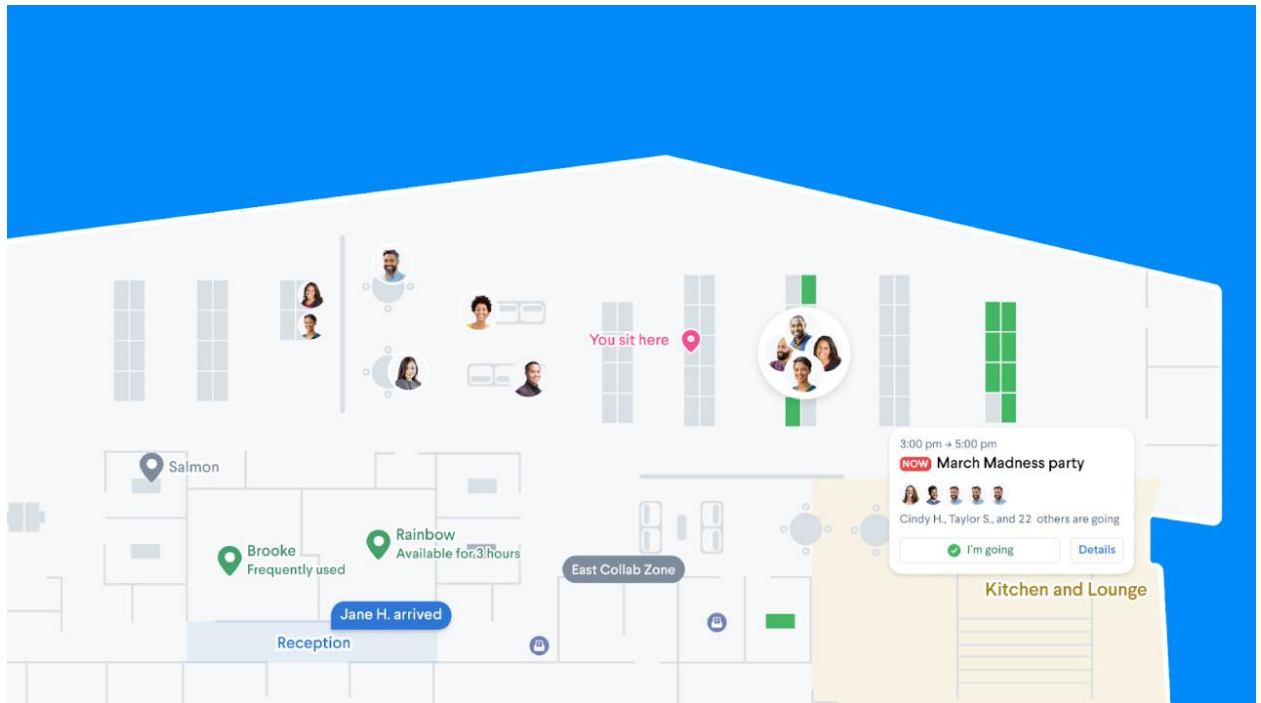


Рис. 1.3 Інтерфейс системи Robin (панель управління ресурсами)

1.3 Порівняльний аналіз та висновки до розділу 1

На основі проведеного аналізу виконано порівняння розглянутих систем за ключовими критеріями, що визначені з огляду на потреби навчального закладу. Результати порівняння наведено в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз систем бронювання ресурсів

Критерій	Skedda	Microsoft Bookings	Robin	Book It!
Тип системи	Комерційна	Комерційна	Комерційна	Веб-система для ВНЗ
Орієнтація на навчальні заклади	Частково (базові інструменти для бронювання)	Ні	Ні	Так
Ролі користувачів	Адмін, користувач	Адмін, клієнт	Адмін, співробітник	Адмін, викладач, студент
Перевірка конфліктів	Так	Так	Так	Так
Пошук і фільтрація	Так	Частково (фільтрація обмежена)	Так	Так
Логування дій	Частково (історія бронювань)	Ні	Так	Так
Вартість	Платна	Платна	Платна	Безкоштовна
Адаптація під ВНЗ	Обмежена (налаштування простору)	Ні	Ні	Повна

Як видно з таблиці 1.1, жодна з розглянутих комерційних систем не задовольняє повністю потреби навчального закладу. Головні спільні недоліки: комерційне ліцензування, відсутність повноцінної трирівневої рольової моделі, складність адаптації під специфіку освітнього процесу, відсутність відкритого вихідного коду та, як правило, відсутність україномовного інтерфейсу.

Проведений аналіз обґрунтовує доцільність розробки власної спеціалізованої системи Book It!, яка врахує всі виявлені недоліки аналогів і буде спроектована з урахуванням специфіки українського навчального закладу. Система повинна бути безкоштовною у використанні, мати чітку трирівневу рольову модель (адміністратор – викладач – студент),

забезпечувати автоматичну перевірку конфліктів бронювань, надавати зручний календарний інтерфейс та вести повний журнал дій. Принципова відмінність розроблюваної системи від аналогів полягає також у підтримці академічного розкладу пар як базової одиниці бронювання, що відповідає реальній практиці організації навчального процесу в українських закладах вищої освіти.

Висновки до розділу 1.

У першому розділі проведено аналіз предметної галузі управління матеріально-технічними ресурсами навчального закладу. Визначено ключові сутності та процеси предметної галузі, сформульовано функціональні та нефункціональні вимоги до розроблюваної системи. Виконано огляд та порівняльний аналіз трьох найбільш релевантних комерційних систем бронювання: Skedda, Microsoft Bookings та Robin. Встановлено, що жодна з них не відповідає повністю потребам навчального закладу через комерційне ліцензування, обмежену рольову модель та складність адаптації. Це підтверджує доцільність розробки власної системи Book It! на основі відкритих технологій.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Вибір засобів розробки є важливим етапом, що безпосередньо впливає на продуктивність роботи та якість кінцевого продукту. Для реалізації системи Book It! було обрано інструменти, що відповідають вимогам проєкту та забезпечують комфортний процес розробки.

2.1.1 Visual Studio Code

Основним середовищем розробки обрано Visual Studio Code (VS Code) – безкоштовний редактор коду з відкритим вихідним кодом від компанії Microsoft. VS Code є одним із найпопулярніших інструментів серед Python- та web-розробників завдяки своїй легкості, розширюваності та широкій підтримці мов програмування.

Для розробки системи Book It! у VS Code було встановлено такі розширення: Python (офіційне розширення Microsoft для підтримки Python, налагоджувача та автодоповнення), Django (підсвічування синтаксису шаблонів Django), Pylance (покращений аналіз типів Python), GitLens (розширені можливості роботи з Git), а також SQLite Viewer для перегляду бази даних безпосередньо в редакторі.

VS Code забезпечує інтегрований термінал, що дозволяє виконувати команди Django (manage.py), запускати тести та керувати залежностями без перемикання між вікнами. Підтримка віртуальних середовищ Python та вбудована інтеграція з Git роблять його оптимальним вибором для повного циклу розробки даного проєкту.

2.2 Мова програмування Python 3.13

Python – інтерпретована об'єктно-орієнтована мова програмування загального призначення, яка вирізняється чистим синтаксисом, читабельністю коду та широкою екосистемою бібліотек. Для розробки системи Book It! обрано версію Python 3.13 – найновішу стабільну версію на момент розробки.

Вибір Python обґрунтовано такими чинниками. По-перше, Python є основною мовою фреймворку Django, що забезпечує повну сумісність і доступ до всіх можливостей платформи. По-друге, мова має мінімалістичний синтаксис, що прискорює розробку та спрощує підтримку коду. По-третє, Python має широку підтримку в хмарних середовищах розгортання, зокрема на платформі Render.

Python 3.13 містить суттєві покращення продуктивності порівняно з попередніми версіями, зокрема оптимізований інтерпретатор та покращену систему підказок типів. Динамічна типізація Python у поєднанні з type hints дозволяє писати гнучкий код без зайвої складності, що є важливим для проєкту з обмеженими строками розробки.

При виборі мови програмування розглядались альтернативи: Node.js (JavaScript на сервері) та PHP. Node.js забезпечує високу продуктивність для асинхронних задач, проте потребує значно більшого обсягу налаштувань для реалізації ORM, автентифікації та адміністративного інтерфейсу. PHP є широко розповсюдженою мовою для web-розробки, проте поступається Python за читабельністю коду та якістю екосистеми для швидкої розробки. Python у поєднанні з Django надає найбільш збалансоване рішення для проєкту даного типу, мінімум шаблонного коду при максимумі готової функціональності.

2.3 Web-фреймворк Django 6.0

Django – це потужний web-фреймворк для Python, що дотримується

принципу «батарейки в комплекті» (batteries included): він надає готові рішення для більшості типових задач web-розробки, включаючи ORM, систему автентифікації, адміністративну панель, маршрутизацію URL та систему шаблонів. Для системи Book It! обрано Django версії 6.0.

Ключовою архітектурною концепцією Django є шаблон MVT (Model-View-Template), що є адаптацією класичного MVC (Model-View-Controller) [10]. У цьому шаблоні Model відповідає за роботу з даними через ORM, View реалізує бізнес-логіку обробки запитів, а Template визначає представлення даних у HTML. Такий поділ забезпечує чітке розмежування відповідальності між компонентами системи.

Вибір Django 6.0 для системи Book It! зумовлений такими перевагами: вбудована система автентифікації та авторизації дозволяє реалізувати трирівневу рольову модель без написання коду з нуля;

Django ORM забезпечує зручну роботу з базою даних через Python-об'єкти, що спрощує реалізацію перевірки конфліктів бронювань;

система Django Signals дозволяє реалізувати журналювання дій без прямих залежностей між модулями;

вбудована адміністративна панель Django Admin надає адміністратору зручний інтерфейс управління ресурсами без додаткової розробки;

Django 6.0 підтримує асинхронні views, що відкриває можливості для масштабування системи у майбутньому.

При виборі фреймворку розглядались альтернативи – Flask та FastAPI. Flask є мікрофреймворком і надає лише базову маршрутизацію, все інше (ORM, автентифікація, адмін панель) потребує підключення окремих бібліотек і додаткової конфігурації. FastAPI орієнтований на побудову REST API і не має вбудованої системи шаблонів та адміністративного інтерфейсу, що робить його менш підходящим для проєкту з повноцінним серверним рендерингом HTML. Django у версії 6.0 є найбільш комплексним рішенням для розробки повноцінного web-застосунку в рамках академічного проєкту з обмеженими строками.

Django також має відмінну документацію та велику спільноту, що значно прискорює вирішення виникаючих питань під час розробки [14].

2.4 База даних SQLite

SQLite – це вбудована реляційна система управління базами даних, що зберігає всю базу даних у єдиному файлі на диску. На відміну від клієнт-серверних СУБД (PostgreSQL, MySQL), SQLite не потребує окремого серверного процесу, що суттєво спрощує розгортання та адміністрування системи.

Для системи Book It! SQLite обрано з таких міркувань. По-перше, система бронювання навчального закладу має відносно невелике навантаження – кілька сотень одночасних користувачів, що повністю відповідає можливостям SQLite. По-друге, відсутність необхідності в окремому сервері бази даних знижує операційні витрати на хостинг та спрощує CI/CD-конвеєр. По-третє, Django забезпечує повну підтримку SQLite через ORM, тому за необхідності міграція на PostgreSQL може бути виконана без змін у коді застосунку – лише через оновлення конфігурації.

SQLite підтримує всі необхідні для системи операції: транзакції ACID, зовнішні ключі, складні JOIN-запити та індекси. Версія SQLite, що постачається з Python 3.13, включає підтримку WAL (Write-Ahead Logging), що покращує продуктивність при конкурентних записах.

Для зберігання часових даних бронювань у системі Book It! використовується тип поля DateTimeField Django ORM, що зберігає дату і час початку та кінця бронювання в одному полі. Такий підхід спрощує перевірку конфліктів: ORM-запит з умовами `start_time_lt` та `end_time_gt` дозволяє одним запитом знайти всі бронювання, що перетинаються з заданим часовим проміжком, без додаткових обчислень на стороні застосунку.

2.5 Клієнтські технології: HTML5, CSS3, JavaScript, AJAX

Клієнтська частина системи Book It! реалізована з використанням стандартних web-технологій без застосування важких JavaScript-фреймворків (React, Vue, Angular). Такий підхід обґрунтований необхідністю підтримки Django-шаблонів і забезпечує швидше завантаження сторінок.

HTML5 використовується для розмітки інтерфейсу системи. Семантичні елементи HTML5 (`<header>`, `<main>`, `<section>`, `<article>`) забезпечують структурованість документа та покращують доступність. Форми HTML5 з вбудованою валідацією (атрибути `required`, `type="date"`, `min`, `max`) зменшують кількість помилок на стороні клієнта ще до відправки запиту на сервер.

CSS3 відповідає за візуальне оформлення інтерфейсу. Застосовуються CSS Grid та Flexbox для побудови адаптивних макетів, CSS-змінні для централізованого управління кольоровою схемою, а також медіа-запити для адаптації інтерфейсу під різні розміри екранів. Календарна сітка зайнятості ресурсів реалізована засобами CSS Grid.

JavaScript використовується для динамічної взаємодії з інтерфейсом без перезавантаження сторінки. Зокрема, реалізовано AJAX-запити за допомогою Fetch API [9] для фільтрації ресурсів, динамічного оновлення календарної сітки та відображення статусу бронювань у реальному часі. Fetch API є сучасною альтернативою XMLHttpRequest і підтримується всіма актуальними браузерами без додаткових бібліотек.

Відмова від jQuery та важких фреймворків на користь нативного JavaScript дозволяє мінімізувати розмір клієнтської частини та прискорити завантаження сторінок, що є важливим для доступності системи з мобільних пристроїв зі слабким з'єднанням.

Тижневий розклад зайнятості ресурсів реалізований у вигляді матриці CSS Grid, де рядки відповідають академічним парам (від 1-ї до 6-ї), а стовпці дням тижня (з понеділка по неділю). Кожна клітинка сітки динамічно отримує CSS-клас залежно від статусу бронювання: вільно (зелений колір), зайнято-

підтверджено (червоний), очікує підтвердження (жовтий), минулий слот (сірий). Функція швидкого бронювання реалізована як AJAX POST-запит прямо з клітинки тижневої сітки, що дозволяє викладачу забронювати пару без переходу на окрему сторінку форми.

2.6 Інфраструктурні засоби: Gunicorn, WhiteNoise, Render

Для виробничого розгортання системи Book It! використовується набір інфраструктурних інструментів, що забезпечують надійну та ефективну роботу застосунку у хмарному середовищі.

2.6.1 Gunicorn 25.3.0

Gunicorn (Green Unicorn) – це WSGI HTTP-сервер для Python-застосунків, що реалізує інтерфейс WSGI (PEP 3333). Вбудований сервер розробки Django (runserver) не призначений для виробничого використання через низьку продуктивність та відсутність механізмів стійкості до відмов. Gunicorn вирішує цю проблему, забезпечуючи pre-fork worker model: майстер-процес породжує кілька робочих процесів (workers), кожен з яких обробляє HTTP-запити незалежно.

Для системи Book It! Gunicorn налаштовано з оптимальною кількістю workers відповідно до рекомендації ($2 \times \text{кількість CPU} + 1$). Це забезпечує ефективне використання ресурсів сервера Render та стійку роботу при піковому навантаженні під час реєстрації або початку семестру.

Команда запуску Gunicorn у виробничому середовищі має вигляд: `gunicorn bookit.wsgi:application --workers 2 --bind 0.0.0.0:PORT`.

Параметр `--bind 0.0.0.0:PORT`. Параметр `--bind 0.0.0.0:PORT` дозволяє Render динамічно призначати порт контейнеру, що є стандартною практикою для PaaS-платформ.

2.6.2 WhiteNoise 6.12.0

WhiteNoise – бібліотека для ефективної роздачі статичних файлів (CSS, JavaScript, зображення) безпосередньо з Python-застосунку без необхідності окремого web-сервера (Nginx, Apache). WhiteNoise автоматично додає заголовки кешування, стискає файли за допомогою gzip та brotli, а також підтримує версіонування статички через хеш-суфікси у назвах файлів.

Використання WhiteNoise замість окремого Nginx-сервера спрощує архітектуру розгортання на Render: весь застосунок, включаючи статичку, обслуговується єдиним Gunicorn-процесом. Це знижує складність конфігурації та операційні витрати без суттєвої втрати продуктивності для проєкту даного масштабу.

WhiteNoise підключається як MIDDLEWARE в settings.py одразу після SecurityMiddleware: 'whitenoise.middleware.WhiteNoiseMiddleware'. Під час розгортання команда `python manage.py collectstatic` збирає всі статичні файли застосунку в каталог `staticfiles/`, після чого WhiteNoise роздає їх з цього каталогу з автоматичним стисненням і кешуванням.

2.6.3 Render

Render – хмарна платформа типу PaaS (Platform as a Service), що надає послуги хостингу web-застосунків, баз даних та фонових задач. Для системи Book It! обрано Render з таких причин: безкоштовний тарифний план для некомерційних проєктів, автоматичне розгортання з GitHub при кожному push у гілку main, підтримка Python/Django без додаткової конфігурації серверів, вбудований HTTPS із автоматичним оновленням SSL-сертифікатів.

Render автоматично виявляє Python-проєкт за наявністю файлу `requirements.txt` та виконує встановлення залежностей, збір статички (`collectstatic`) та запуск Gunicorn відповідно до налаштувань у файлі `render.yaml`. Це забезпечує повністю автоматизований pipeline від коміту до виробничого розгортання.

Усі чутливі змінні конфігурації (`SECRET_KEY`, `DEBUG`,

ALLOWED_HOSTS) зберігаються у захищеному сховищі Render Environment Variables і не потрапляють до репозиторію GitHub відповідно до принципу 12-Factor App [18]. Це відповідає принципу 12-Factor App щодо зберігання конфігурації в середовищі, а не в коді. SSL-сертифікат автоматично видається та оновлюється через Let's Encrypt, що забезпечує захищену передачу даних між клієнтом і сервером.

2.7 Система контролю версій та CI/CD

Для управління вихідним кодом системи Book It! використовується Git розподілена система контролю версій. Репозиторій проєкту розміщено на платформі GitHub, що надає хмарне сховище коду, інструменти code review та систему відстеження задач.

Безперервна інтеграція та доставка (CI/CD) [19] реалізована засобами GitHub Actions вбудованого інструменту автоматизації GitHub. Налаштований workflow виконується автоматично при кожному push у репозиторій і включає такі кроки:

1. встановлення Python 3.13 та залежностей з requirements.txt;
2. запуск автоматичних тестів (python manage.py test);
3. перевірка конфігурації Django (python manage.py check);
4. автоматичне розгортання на Render при успішному проходженні всіх кроків.

Така конфігурація забезпечує, що до виробничого середовища потрапляє лише код, що пройшов автоматичну перевірку. Це знижує ризик регресій та прискорює цикл розробки завдяки автоматизованому зворотному зв'язку.

Використання GitHub також дозволяє вести розробку у гілках (feature branches), що ізолює експериментальні зміни від стабільної версії коду та полегшує командну роботу.

За час розробки системи Book It! виконано близько 47 комітів у репозиторій. GitHub Actions тричі виявляв регресії до потрапляння коду у

виробниче середовище: двічі через помилки імпорту після рефакторингу модулів та один раз через некоректну міграцію бази даних. Це підтвердило практичну цінність автоматизованого тестування навіть для невеликих проєктів.

Висновки до розділу 2.

У другому розділі обґрунтовано вибір технологічного стеку для розробки web-системи бронювання матеріально-технічних ресурсів навчального закладу Book It!. Визначено, що Python 3.13 у поєднанні з Django 6.0 є оптимальним вибором для реалізації серверної частини завдяки розвиненій екосистемі, вбудованим засобам автентифікації, ORM та системі сигналів. SQLite обрано як базу даних з огляду на масштаб проєкту та простоту розгортання. Клієнтська частина реалізована на основі нативних web-технологій (HTML5, CSS3, JavaScript/AJAX) без застосування важких фреймворків. Інфраструктурний стек (Gunicorn, WhiteNoise, Render, GitHub Actions) забезпечує надійне хмарне розгортання з автоматизованим CI/CD-конвеєром. Сукупність обраних засобів реалізації відповідає поставленим функціональним та нефункціональним вимогам до системи.

3 ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Архітектура системи на основі шаблону MVT

Архітектура системи Book It! побудована на основі шаблону проєктування MVT (Model-View-Template), що є ключовою концепцією фреймворку Django. MVT є адаптацією класичного шаблону MVC, де роль Controller виконує сам фреймворк Django через механізм маршрутизації URL.

Шаблон MVT складається з трьох компонентів, кожен з яких несе чітко визначену відповідальність:

Model (Модель) – відповідає за визначення структури даних та взаємодію з базою даних через Django ORM. У системі Book It! моделі включають: Resource (ресурс навчального закладу), Booking (бронювання), UserProfile (профіль користувача з роллю) та ActionLog (журнал дій). Модель не залежить від способу відображення даних і може бути повторно використана в різних Views.

View (Представлення) – реалізує бізнес-логіку обробки HTTP-запитів. View отримує запит від Django-диспетчера, звертається до моделей для отримання або збереження даних, виконує необхідні перевірки (автентифікація, перевірка конфліктів бронювань, контроль прав доступу) та передає результат у Template для відображення. У системі Book It! Views реалізовані як функції з декораторами.

Template (Шаблон) – визначає HTML-представлення даних, що передаються з View. Django-шаблони підтримують успадкування (base.html), умовні блоки, цикли та фільтри, що дозволяє організувати інтерфейс у вигляді компонентів без дублювання коду.

Загальний цикл обробки запиту в системі Book It! відбувається таким чином: браузер користувача надсилає HTTP-запит до сервера Gunicorn; Django-диспетчер маршрутизує запит до відповідного View на основі

URLconf; View виконує бізнес-логіку, звертаючись до моделей через ORM; результат передається до Template, який генерує HTML-відповідь; Unicorn повертає відповідь браузеру.

Для забезпечення розмежування прав доступу система використовує декоратор `@login_required` та перевірку ролі користувача (`UserProfile.role`) на початку кожного View. Це гарантує, що студент не може підтвердити або відхилити бронювання – ця операція доступна лише викладачу або адміністратору.

Важливою особливістю реалізації є те, що адміністратор у системі Book It! не виділений як окрема роль у моделі `UserProfile`, а реалізований через стандартний прапорець `is_staff` Django. Це дозволяє використовувати вбудований декоратор `@staff_member_required` для захисту адміністративних View і не ускладнювати логіку перевірки ролей у коді.

Таблиця 3.1

Розподіл функціональності між компонентами MVT

Компонент	Класи / файли	Відповідальність
Model	models.py	Resource, Booking, UserProfile, ActionLog – структури даних та ORM-запити
View	views.py	Обробка запитів, перевірка конфліктів, контроль доступу за ролями
Template	templates/*.html	HTML-інтерфейс, успадкування base.html, AJAX-відповіді
URL	urls.py	Маршрутизація HTTP-запитів до відповідних Views
Signals	signals.py	Автоматичне журналювання дій через Django Signals
Admin	admin.py	Налаштування Django Admin для управління ресурсами

3.2 Діаграма варіантів використання

Діаграма варіантів використання (Use Case Diagram) описує функціональні вимоги до системи з точки зору акторів – суб'єктів, що взаємодіють із системою. У системі Book It! визначено три ролі, що відповідають ролям користувачів:

Студент – може переглядати каталог ресурсів, перевіряти їх зайнятість у календарі;

Викладач – має всі права студента, а також може бронювати ресурс та скасовувати бронювання;

Адміністратор – має повний доступ до системи: може додавати, редагувати та видаляти ресурси, управляти всіма бронюваннями, переглядати журнал дій та керувати обліковими записами користувачів.

Розмежування між роллю викладача та адміністратора відображає реальну практику навчального закладу: викладач самостійно бронює звичайні приміщення (аудиторії, лабораторії, обладнання) з автоматичним підтвердженням, тоді як бронювання конференц-залів потребує окремого схвалення адміністратором. Студенти мають доступ лише до перегляду розкладу і не можуть самостійно створювати бронювання.

Основні варіанти використання системи Book It! наведено нижче.

Таблиця 3.2

Варіанти використання системи Book It!

Варіант використання	Ролі	Короткий опис
Реєстрація та вхід	Всі	Користувач реєструється або входить до системи;
Перегляд каталогу ресурсів	Всі	Перегляд списку ресурсів з фільтрацією за типом та доступністю

Продовження таблиці 3.2

Варіант використання	Ролі	Короткий опис
Перегляд календаря зайнятості	Всі	Відображення розкладу зайнятості ресурсу на тиждень/місяць
Подання заявки на бронювання	Викладач. Адмін	Вибір ресурсу, дати та часового слоту; перевірка конфліктів
Підтвердження заявки	Адмін	Перегляд та підтвердження заявки зі статусом «очікує»
Відхилення заявки	Адмін	Відхилення заявки із зазначенням причини
Скасування бронювання	Викладач. Адмін	Скасування власного бронювання до настання часу
Управління ресурсами	Адмін	Додавання, редагування, деактивація ресурсів
Перегляд журналу дій	Адмін	Перегляд історії всіх операцій у системі
Управління користувачами	Адмін	Зміна ролей, блокування облікових записів



Рис. 3.1 Діаграма варіантів використання системи Book It!

Наведені варіанти використання охоплюють увесь функціональний периметр системи та слугують основою для розробки тест-кейсів у розділі 4. Діаграма Use Case побудована відповідно до нотації UML 2.5 [15] та відображає залежності між варіантами використання (include, extend) [16].

3.3 ER-модель бази даних

ER-модель (Entity-Relationship model) описує структуру бази даних системи Book It! у вигляді сутностей та зв'язків між ними. База даних реалізована засобами SQLite через Django ORM і включає чотири основні сутності.

Сутність Resource (Ресурс) зберігає інформацію про матеріально-технічні об'єкти навчального закладу. Атрибути: id (первинний ключ, автоінкремент), name (назва ресурсу, VARCHAR), resource_type (тип: аудиторія, лабораторія, обладнання, VARCHAR), capacity (місткість, INTEGER), description (опис, TEXT), is_active (активність, BOOLEAN), created_at (дата створення, DATETIME). Один ресурс може мати багато бронювань (зв'язок 1:N з Booking).

Сутність User (Користувач) використовує вбудовану модель Django та розширюється через сутність UserProfile. Атрибути базової моделі: id, username, email, password (хеш), first_name, last_name, is_active, is_staff. Сутність UserProfile містить: user (зовнішній ключ до User, 1:1), role (роль: student/teacher, VARCHAR). Адміністратор реалізований через прапорець is_staff Django, що дозволяє зберегти сумісність із вбудованою системою автентифікації без модифікації базової моделі.

Сутність Booking (Бронювання) є центральною в системі. Атрибути: id (первинний ключ), resource (зовнішній ключ до Resource), user (зовнішній ключ до User), start_time (дата і час початку, DATETIME), end_time (дата і час закінчення, DATETIME), status (статус: pending/approved/rejected/cancelled, VARCHAR), created_at (DATETIME). Перевірка конфліктів реалізована на рівні форми BookingForm.clean(): ORM-запит з умовами start_time_lt та end_time_gt знаходить усі активні бронювання, що перекриваються з запитуваним часовим проміжком.

Сутність ActionLog (Журнал дій) фіксує всі операції у системі. Атрибути: id, user (зовнішній ключ до User), action_type (тип дії, VARCHAR), object_type (тип об'єкта, VARCHAR), object_repr (текстове представлення об'єкта, VARCHAR), description (деталі, TEXT), created_at (DATETIME). Записи журналу створюються через явний виклик функції log_action у кожному View, а також автоматично через Django Signals для подій входу та виходу з системи.

Детальний опис полів кожної сутності наведено в таблицях (3.4-3.6).

Таблиця 3.3

Зв'язки між сутностями бази даних

Сутність 1	Сутність 2	Тип зв'язку	Опис
Resource	Booking	1 : N	Один ресурс може мати багато бронювань
User	Booking	1 : N	Один користувач може мати багато бронювань
User	UserProfile	1 : 1	Кожен користувач має один профіль із роллю
User	ActionLog	1 : N	Один користувач може здійснювати багато дій
Booking	ActionLog	1 : N	Одне бронювання може породжувати багато записів у лозі

Таблиця 3.4

Поля моделі Resource

Поле	Тип Django	Обмеження	Опис
id	BigAutoField	PK, автоінкремент	Первинний ключ
name	CharField(255)	not null	Назва ресурсу
resource_type	CharField(50)	choices	Тип: room, lab, equipment, computer_class, conference_room
department	CharField(20)	blank=True	Кафедра: ipz, kn, ai, ist
location	CharField(255)	not null	Місцезнаходження
capacity	PositiveIntegerField	default=1	Місткість (осіб)
is_active	BooleanField	default=True	Чи доступний для бронювання
created_at	DateTimeField	auto_now_add	Дата та час створення

Таблиця 3.5

Поля моделі Booking

Поле	Тип Django	Обмеження	Опис
id	BigAutoField	PK, автоінкремент	Первинний ключ
resource	ForeignKey(Resource)	cascade	Ресурс, що бронюється
user	ForeignKey(User)	cascade	Користувач, що створив бронювання
start_time	DateTimeField	not null	Дата і час початку
end_time	DateTimeField	not null	Дата і час закінчення
status	CharField(20)	choices, default='pending'	pending, approved, rejected, cancelled
created_at	DateTimeField	auto_now_add	Час створення запису

Таблиця 3.6

Поля моделі ActionLog

Поле	Тип Django	Обмеження	Опис
id	BigAutoField	PK, автоінкремент	Первинний ключ
user	ForeignKey(User)	set_null, null=true	Користувач, що виконав дію
action_type	CharField(50)	choices	Тип дії: login, create_booking, approve_booking тощо
object_type	CharField(100)	blank=True	Тип об'єкта: booking, resource, auth

Продовження таблиці 3.6

Поле	Тип Django	Обмеження	Опис
object_repr	CharField(255)	blank=True	Текстове представлення об'єкта
description	TextField	blank=True	Детальний опис події
created_at	DateTimeField	auto_now_add	Час фіксації події

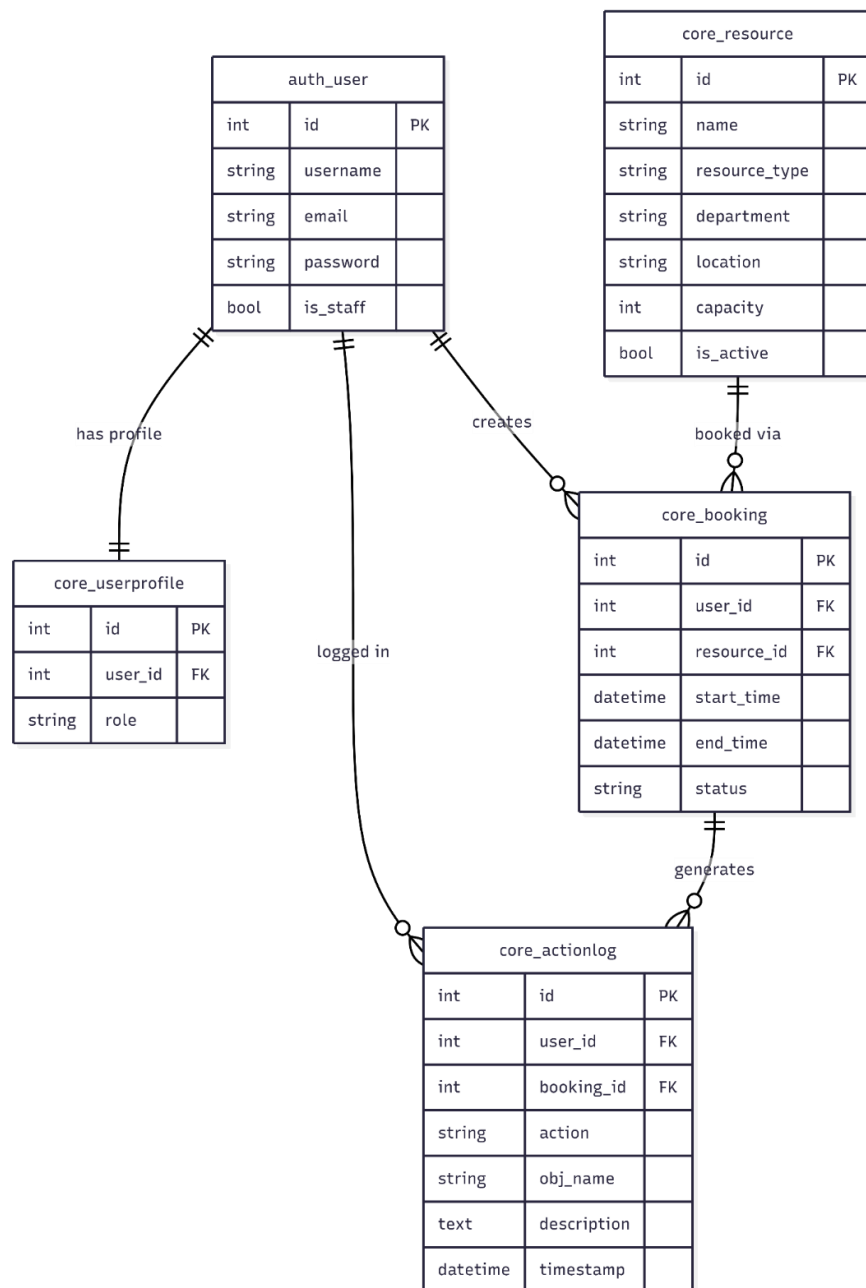


Рис. 3.2 ER-діаграма бази даних системи Book It!

3.4 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) відображає взаємодію між об'єктами системи у часовому порядку для конкретного варіанту використання. На діаграмі зображено п'ять учасників: User (користувач), Browser (браузер), Django View (серверна логіка), DB/Model (база даних через ORM) та Signals (система сигналів Django). Розглянемо ключовий сценарій – подання заявки на бронювання ресурсу, що включає перевірку конфліктів.

Перший етап сценарію – перегляд розкладу. Користувач відкриває сторінку розкладу ресурсу, браузер надсилає GET-запит на адресу `/resources/{id}/schedule/`. Django View виконує запит `Resource.objects.get(id)` до бази даних, отримує об'єкт ресурсу та пов'язані бронювання (`resource`, `bookings`), після чого рендерить HTML-шаблон і повертає його браузеру. Браузер відображає розклад користувачу.

Другий етап – безпосереднє бронювання. Користувач натискає кнопку бронювання на потрібному слоті. Браузер надсилає POST-запит на адресу `/bookings/create`. Django View виконує перевірку конфліктів через ORM-фільтр. База даних повертає порожній список (`[]` – конфліктів немає). View викликає `Booking.objects.create()` для збереження нового бронювання. Після збереження автоматично спрацьовує `post_save` signal, який передає керування до Signals. Signals викликає `ActionLog.create()` для фіксації події у журналі. Нарешті View виконує `redirect` на `/my-bookings/`, і браузер відображає оновлений список бронювань.

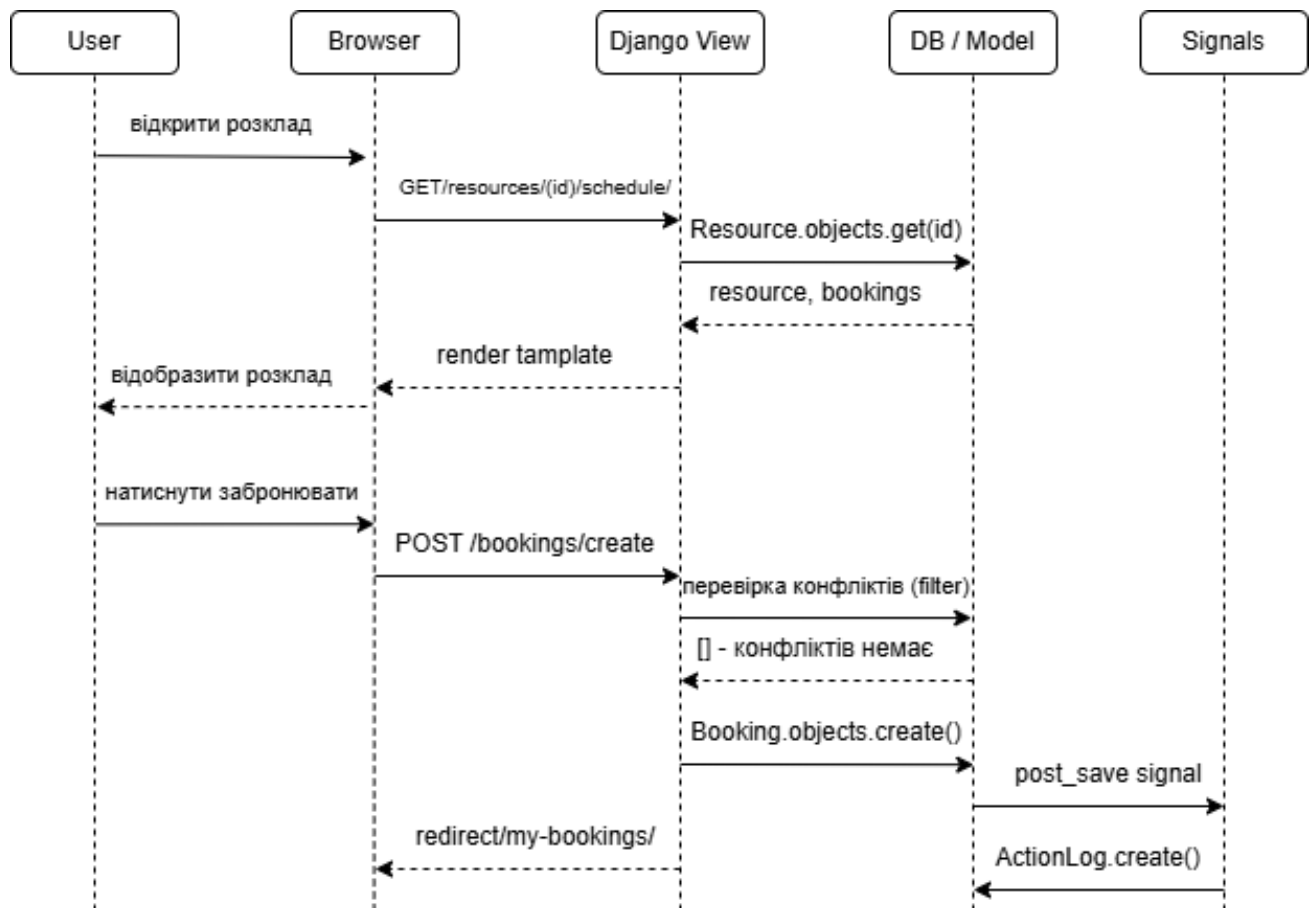


Рис. 3.3 Діаграма послідовності сценарію бронювання ресурсу

Діаграма наочно демонструє ключовий принцип реалізації системи: перевірка конфліктів відбувається до збереження запису в базі даних, а журналювання – автоматично після збереження через механізм сигналів, без додаткового коду у View. Це забезпечує розмежування відповідальності між компонентами системи.

Якщо при перевірці конфліктів ORM-запит повертає непорожній результат – View повертає форму з повідомленням про зайнятість ресурсу та списком доступних альтернативних ресурсів того самого типу. Статус нового бронювання визначається автоматично залежно від типу ресурсу: для звичайних приміщень (аудиторії, лабораторії, обладнання) встановлюється статус 'approved', для конференц-залів – 'pending', що потребує підтвердження адміністратором через окрему панель управління.

Окремий сценарій – підтвердження або відхилення заявки адміністратором. View `approve_booking` або `reject_booking` отримує POST-запит з ідентифікатором бронювання зі статусом 'pending', змінює статус на 'approved' або 'rejected' та зберігає зміни через `save(update_fields=['status'])`. Функція `log_action` фіксує подію в `ActionLog`. View повертає перенаправлення на панель адміністратора з переліком заявок, що очікують.

3.5 Діаграма розгортання

Діаграма розгортання (Deployment Diagram) описує фізичну архітектуру системи Book It! у виробничому середовищі – розподіл компонентів по вузлах інфраструктури та їх взаємодію.

Система розгорнута на платформі Render і включає такі вузли та компоненти:

Вузол «Клієнтський браузер» – виконує HTML/CSS/JavaScript клієнтської частини; взаємодіє із сервером через HTTPS (порт 443); надсилає стандартні HTTP-запити та AJAX-запити (Fetch API).

Вузол «Render Web Service» – хмарний сервер на платформі Render; виконує Gunicorn WSGI-сервер (workers), який обробляє вхідні HTTP-запити та передає їх Django-застосунок; WhiteNoise middleware обслуговує статичні файли безпосередньо з файлової системи контейнера без перенаправлення на окремий сервер.

Компонент «Django Application» – основний застосунок на Python 3.13/Django 6.0; включає модулі URLconf (маршрутизація), Views (бізнес-логіка), Models (ORM), Templates (HTML-шаблони) та Signals (журналювання).

Компонент «SQLite Database» – файл бази даних `db.sqlite3`, що розміщується у файловій системі контейнера Render; Django ORM взаємодіє з ним через стандартний Python DB-API 2.0.

Вузол «GitHub Repository» – зберігає вихідний код системи; при push у гілку main автоматично запускається GitHub Actions workflow, що виконує тести та ініціює розгортання на Render через webhook.

Взаємодія між вузлами організована таким чином: GitHub Actions надсилає webhook на Render при успішному проходженні тестів; Render завантажує новий код, встановлює залежності з requirements.txt, виконує python manage.py collectstatic та перезапускає Gunicorn-процеси за методом rolling restart без простою сервісу.

Усі з'єднання між клієнтом і сервером захищені протоколом TLS 1.3; SSL-сертифікат автоматично видається та оновлюється платформою Render через Let's Encrypt. Змінні середовища (SECRET_KEY, DEBUG=False, ALLOWED_HOSTS) зберігаються у захищеному сховищі Render і не потрапляють до репозиторію GitHub.

Важливою характеристикою обраної архітектури розгортання є відсутність стану між запитами (stateless design): кожен Gunicorn worker обробляє запити незалежно, а стан сесії зберігається в базі даних SQLite. Це дозволяє масштабувати кількість workers без змін у коді застосунку. Єдиним станом, що зберігається між запитами, є файл бази даних db.sqlite3, що є достатнім для масштабу поточного проєкту.

Висновки до розділу 3.

У третьому розділі виконано повне проєктування системи Book It!. Описано архітектуру на основі шаблону MVT з чітким розподілом відповідальності між компонентами Model, View та Template. Важливою особливістю є реалізація адміністратора через прапорець `is_staff` Django, що спрощує логіку контролю доступу. Розроблено діаграму варіантів використання з десятима сценаріями для трьох акторів (студент, викладач, адміністратор), де розмежовано права бронювання звичайних приміщень та конференц-залів. Спроектовано ER-модель бази даних, що включає чотири сутності (Resource, Booking, UserProfile, ActionLog) з визначеними зв'язками та обмеженнями – детальний опис полів наведено в таблицях 3.4–3.6. Описано діаграму послідовності для ключового сценарію перегляду розкладу та подання заявки на бронювання: перевірка конфліктів відбувається до збереження запису, статус бронювання визначається автоматично залежно від типу ресурсу, журналювання виконується через Django Signals. Наведено діаграму розгортання, що відображає фізичну архітектуру системи на платформі Render із CI/CD-конвеєром на базі GitHub Actions та stateless-дизайном для масштабування. Розроблені проєктні рішення є достатніми для переходу до етапу реалізації.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Структура проєкту та налаштування Django

Проєкт Book It! організовано відповідно до стандартної структури Django-застосунку з розділенням на конфігураційний пакет і функціональні застосунки (apps). Така організація забезпечує модульність та спрощує підтримку коду.

Структура каталогів проєкту має такий вигляд:

bookit/ – Кореневий каталог проєкту

├── config/ – Конфігураційний пакет Django

| ├── settings.py – Налаштування проєкту

| ├── urls.py – Головний URLconf

| └── wsgi.py – WSGI-точка входу для Gunicorn

├── core/ – Основний застосунок

| ├── models.py – Моделі: Resource, Booking, UserProfile, ActionLog

| ├── views.py – Views: логіка обробки запитів

| ├── urls.py – URLconf застосунку

| ├── signals.py – Django Signals для журналювання

| ├── logging_utils.py – Функція log_action для журналювання

| ├── admin.py – Налаштування Django Admin

| ├── forms.py – Django Forms для валідації

| ├── utils.py – Допоміжні функції та константа PAIR_SLOTS

| └── templates/ – HTML-шаблони

| ├── base.html – Базовий шаблон

| ├── resource_list.html – Каталог ресурсів

| ├── resource_schedule.html – Денний розклад

| └── resource_week_schedule.html – Тижневий розклад

- |— static/ – Статичні файли (CSS, JS)
- |— requirements.txt – Залежності Python
- |— manage.py – Утиліта управління Django

Ключові налаштування у файлі `settings.py` включають: `INSTALLED_APPS` з реєстрацією застосунку `core` та `whitenoise.runserver_nostatic`; `MIDDLEWARE` з `WhiteNoiseMiddleware` після `SecurityMiddleware`; `DATABASES` з налаштуванням `SQLite`; `STATIC_ROOT` для `collectstatic`; стандартна модель `User` Django розширюється через `UserProfile` з `OneToOneField`.

Розгортання на Render виконується автоматично: при кожному `push` у гілку `main` Render встановлює залежності з `requirements.txt`, виконує `python manage.py collectstatic --noinput` та `python manage.py migrate`, після чого запускає `Gunicorn` командою `gunicorn config.wsgi:application`. Це забезпечує повністю автоматизоване розгортання без ручного втручання.

4.2 Реалізація моделей бази даних

Моделі бази даних реалізовані у файлі `core/models.py` з використанням Django ORM. Кожна модель успадковує від `django.db.models.Model` та визначає поля, метадані та методи.

Модель `Resource` описує матеріально-технічний ресурс навчального закладу:

```
class resource(models.model):
    resource_type_choices = [
        ('room', 'аудиторія'),
        ('lab', 'лабораторія'),
        ('equipment', 'обладнання'),
        ('computer_class', 'комп'ютерний клас'),
        ('conference_room', 'конференц-зала'),
```

```

    ]
    department_choices = [
        ('ipz', 'Інженерії програмного забезпечення'),
        ('kn', 'Комп'ютерних наук'),
        ('ai', 'Штучного інтелекту'),
        ('ist', 'Інформаційних систем та технологій'),
    ]
    name = models.CharField(max_length=255)
    resource_type = models.CharField(max_length=50,
choices=RESOURCE_TYPE_CHOICES)
    department = models.CharField(max_length=20,
choices=DEPARTMENT_CHOICES, blank=True)
    location = models.CharField(max_length=255)
    capacity = models.PositiveIntegerField(default=1)
    is_active = models.BooleanField(default=True)
    created_at = models.DateTimeField(auto_now_add=True)
    def str(self):
    return self.name

```

Модель Booking є центральною сутністю системи. Перевірка конфліктів реалізована у формі BookingForm.clean(), а не в методі clean() моделі:

```

class Booking(models.Model):
    STATUS_CHOICES = [
        ('pending', 'Очікує підтвердження'),
        ('approved', 'Підтверджено'),
        ('rejected', 'Відхилено'),
        ('cancelled', 'Скасовано'),
    ]

```

```

resource = models.ForeignKey(Resource, on_delete=models.CASCADE,
related_name='bookings')
user = models.ForeignKey(User, on_delete=models.CASCADE)
start_time = models.DateTimeField()
end_time = models.DateTimeField()
status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='pending')
created_at = models.DateTimeField(auto_now_add=True)
def str(self):
return f"{self.resource.name} ({self.start_time} - {self.end_time})"

```

Модель `UserProfile` розширює стандартного користувача Django полем ролі через зв'язок `OneToOneField`. Роль визначається автоматично при реєстрації на основі домену корпоративної електронної пошти: `@stud.duikt.edu.ua` отримує роль `student`, `@duikt.edu.ua` – роль `teacher`. Новий обліковий запис одразу неактивний (`is_active=False`) і потребує підтвердження адміністратором. Зв'язок між `User` та `UserProfile` створюється автоматично у View реєстрації: після збереження об'єкта `User` викликається `UserProfile.objects.create(user=user, role=form.get_role_by_email())`, що гарантує наявність профілю для кожного зареєстрованого користувача.

Модель `ActionLog` фіксує всі значущі події в системі. Поле `action_type` зберігає тип дії у вигляді рядка з визначеного переліку (`login`, `logout`, `create_booking`, `cancel_booking`, `approve_booking`, `reject_booking`, `delete_booking`, `create_resource` тощо), що дозволяє фільтрувати журнал за типами подій без зміни структури бази даних. Додатково модель зберігає поля `object_repr` (текстове представлення об'єкта, наприклад «Аудиторія 301 | 21.05.2026 09:00 - 10:30») та `description` (детальний опис події), що забезпечує зручний перегляд журналу без необхідності звертатися до інших таблиць.

4.3 Реалізація бізнес-логіки та маршрутизації

Бізнес-логіка системи Book It! реалізована у файлі `core/views.py` у вигляді функцій-представлень з декораторами. Такий підхід обраний замість Class-Based Views для більшої прозорості логіки та спрощення налагодження.

Центральна функція `create_booking` реалізує подання заявки на бронювання. Доступ обмежено: лише викладачі (`profile.role == 'teacher'`) та адміністратори (`is_staff` або `is_superuser`) можуть створювати бронювання:

```
@login_required
def create_booking(request, resource_id):
    resource = get_object_or_404(Resource, id=resource_id, is_active=True)
    if not (request.user.is_superuser or request.user.is_staff):
        if not hasattr(request.user, 'profile') or request.user.profile.role != 'teacher':
            messages.error(request, 'Бронювання доступне лише викладачам.')
        return redirect('resource_detail', resource_id=resource.id)
    if request.method == 'POST':
        form = BookingForm(request.POST, initial={'resource': resource})
        if form.is_valid():
            booking = form.save(commit=False)
            booking.resource = resource
            booking.user = request.user
            booking.status = 'approved' if resource.resource_type != 'conference_room'
            else 'pending'
            booking.save()
            log_action(user=request.user, action_type='create_booking', ...)
            messages.success(request, 'Бронювання успішно створено.')
            return redirect('my_bookings')
        else:
            form = BookingForm(initial={'resource': resource})
```

```
return render(request, 'create_booking.html', {'resource': resource, 'form':
form})
```

Важливою деталлю реалізації є автоматичне визначення статусу бронювання: для звичайних приміщень (аудиторії, лабораторії, обладнання, комп'ютерні класи) статус одразу встановлюється як 'approved', що дозволяє викладачу бронювати їх без очікування. Для конференц-залів статус встановлюється як 'pending' – такі заявки потрапляють до черги на розгляд адміністратором. Це відповідає реальній практиці навчального закладу, де конференц-зали є обмеженим спільним ресурсом.

Функції `approve_booking` та `reject_booking` реалізують підтвердження і відхилення заявок та доступні лише адміністраторам через декоратор `@staff_member_required`:

```
@staff_member_required
def approve_booking(request, booking_id):
    booking = get_object_or_404(Booking, id=booking_id)
    booking.status = 'approved'
    booking.save(update_fields=['status'])
    log_action(user=request.user, action_type='approve_booking', ...)
    messages.success(request, 'Бронювання підтверджено.')
    return redirect('admin_pending_bookings')
```

Функція `cancel_booking` дозволяє користувачу скасувати власне бронювання зі статусом 'pending' або 'approved'. Після скасування викликається `log_action` для фіксації події. Функція `delete_booking` видаляє запис повністю і також доступна лише власнику бронювання. Функція `delete_inactive_bookings` дозволяє одним кліком видалити всі завершені та скасовані бронювання користувача.

Журналювання дій реалізовано через функцію `log_action` у файлі `core/logging_utils.py`, яка явно викликається у кожному View після виконання

операції. Додатково Django Signals у файлі `signals.py` обробляють події входу та виходу з системи:

```
@receiver(user_logged_in)
def handle_user_logged_in(sender, request, user, **kwargs):
    log_action(user=user, action_type='login', object_type='auth',
    object_repr='Вхід у систему',
    description=f'Користувач {user.username} увійшов у систему.')
```

Маршрутизація URL визначена у `core/urls.py` та підключена до головного `urls.py` через `include()`. Усього визначено 16 іменованих маршрутів, що охоплюють перегляд ресурсів, управління бронюваннями, адміністративні функції та журнал дій. Маршрути організовано за логічними групами: ресурси (`/resources/`), особисті бронювання (`/my-bookings/`), адміністративна панель (`/dashboard/`) та журнал дій (`/logs/`).

4.4 Клієнтська частина та AJAX-взаємодія

Клієнтська частина системи Book It! побудована на основі Django-шаблонів з успадкуванням від базового шаблону `base.html`. Базовий шаблон визначає загальну структуру сторінки: навігаційне меню з відображенням ролі користувача, блок повідомлень (Django messages framework) та основний контентний блок.

Календарна сітка зайнятості ресурсів реалізована засобами CSS Grid. Кожна комірка сітки відповідає одному часовому слоту (30 хвилин) і отримує CSS-клас залежно від статусу бронювання: вільно (зелений), зайнято-підтверджено (червоний), очікує підтвердження (жовтий). Це дозволяє користувачу одразу оцінити доступність ресурсу без додаткових запитів.

Тижнева сітка зайнятості ресурсів реалізована засобами CSS Grid [20] у вигляді матриці: рядки відповідають академічним парам (від 1-ї до 6-ї пари), стовпці – дням тижня (понеділок–неділя). Кожна клітинка отримує CSS-клас

залежно від статусу: вільно (зелений), зайнято-підтверджено (червоний), очікує підтвердження (жовтий), минулий слот (сірий). Часові межі пар визначаються константою PAIR_SLOTS у файлі utils.py, що відповідає академічному розкладу університету.

AJAX-взаємодія реалізована через нативний Fetch API для двох ключових функцій: динамічна фільтрація ресурсів за типом та місткістю без перезавантаження сторінки, а також перевірка доступності часового слоту при заповненні форми бронювання. Швидке бронювання пари одним кліком прямо з тижневої сітки (quick_book_pair), а також динамічне відображення альтернативних ресурсів при виявленні конфлікту. Приклад AJAX-запиту для фільтрації ресурсів:

```
document.getElementById('filter-form')
  .addEventListener('change', async function() {
    const params = new URLSearchParams(
      new FormData(this));
    const resp = await fetch(
      '/resources/filter/' + params,
      { headers:
        {'X-Requested-With': 'XMLHttpRequest'} }
    );
    const data = await resp.json();
    renderResourceCards(data.resources);
  });
```

View resource_list перевіряє заголовок x-requested-with: для AJAX-запитів повертає лише HTML-фрагмент з картками ресурсів через шаблон partials/resource_cards_partial.html, для звичайних запитів – повну сторінку resource_list.html. Це дозволяє використовувати один View для обох сценаріїв:

```
def resource_list(request):
```

```

resources = Resource.objects.filter(is_active=True)
search_query = request.GET.get('q', "").strip()
selected_type = request.GET.get('type', "").strip()
selected_department = request.GET.get('department', "").strip()
if selected_type:
    resources = resources.filter(resource_type=selected_type)
if selected_department:
    resources = resources.filter(department=selected_department)
context = {'resources': resources, ...}
if request.headers.get('x-requested-with') == 'XMLHttpRequest':
    return render(request, 'partials/resource_cards_partial.html', context)
return render(request, 'resource_list.html', context)

```

Такий підхід дозволяє використовувати той самий View як для звичайних HTTP-запитів (з повним HTML-відображенням), так і для AJAX-запитів (з частковим рендерингом), що зменшує дублювання коду.

Функція швидкого бронювання `quick_book_pair` реалізована як окремий View, що приймає AJAX POST-запит з номером пари та датою. View визначає часові межі пари з `PAIR_SLOTS`, перевіряє конфлікти та створює бронювання. У разі конфлікту повертає JSON зі списком доступних альтернативних ресурсів того самого типу. Браузер оновлює стан клітинки тижневої сітки без перезавантаження сторінки.

4.5 Розгортання системи та CI/CD-конвеєр

Виробниче розгортання системи Book It! виконано на платформі Render з використанням автоматизованого CI/CD-конвеєра на базі GitHub Actions. Конфігурація розгортання описана у файлі `render.yaml` у кореневому каталозі репозиторію.

Файл конфігурації GitHub Actions (.github/workflows/django.yml) визначає такий pipeline:

```
name: Django CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.13'
      - run: pip install -r requirements.txt
      - run: python manage.py check
      - run: python manage.py test
    env:
      SECRET_KEY: ${ secrets.SECRET_KEY }
      DEBUG: 'False'
```

При кожному push у будь-яку гілку GitHub Actions запускає тести. При успішному проходженні тестів у гілці main Render автоматично отримує webhook і починає процес розгортання: завантажує новий код, встановлює залежності, виконує міграції бази даних та collectstatic, після чого перезапускає Gunicorn за методом rolling restart. Весь процес від коміту до оновлення виробничої системи займає в середньому 2-3 хвилини.

Змінні середовища (SECRET_KEY, ALLOWED_HOSTS, DEBUG) зберігаються у захищеному сховищі Render Environment Variables і не потрапляють до репозиторію. Файл .gitignore виключає db.sqlite3, файли .env та каталог __pycache__ з репозиторію.

4.6 Тестування системи

Тестування системи Book It! проводилося на двох рівнях: функціональне тестування перевіряє відповідність реалізованих функцій вимогам, визначеним у розділі 1, а нефункціональне – оцінює якісні характеристики системи (продуктивність, безпека, зручність використання).

4.6.1 Функціональне тестування

Функціональне тестування виконувалося методом «чорного ящика» на основі тест-кейсів, похідних від варіантів використання (UC-01...UC-10). Автоматизовані тести реалізовані засобами Django TestCase та розміщені у файлі bookings/tests.py

Таблиця 4.1

Результати функціонального тестування

Тест-кейс	Очікуваний результат	Статус
Реєстрація нового користувача з валідними даними	Обліковий запис створено, роль 'student' присвоєно, перенаправлення на головну	Passed
Вхід з невірним паролем	Повідомлення про помилку, сесія не створена	Passed
Перегляд каталогу ресурсів без автентифікації	Перенаправлення на сторінку входу	Passed
Фільтрація ресурсів за типом через AJAX	JSON-відповідь з відфільтрованим списком	Passed
Подання заявки на вільний слот	Бронювання створено зі статусом 'pending'	Passed
Подання заявки на зайнятий підтверджений слот	Повідомлення про конфлікт, бронювання не створено	Passed
Підтвердження заявки викладачем	Статус змінено на 'confirmed', запис у ActionLog	Passed

Продовження таблиці 4.1

Тест-кейс	Очікуваний результат	Статус
Спроба підтвердити заявку студентом	Відповідь 403 Forbidden	Passed
Скасування власного бронювання	Статус змінено на 'cancelled', запис у ActionLog	Passed
Адміністратор додає новий ресурс	Ресурс збережено, з'явився в каталозі	Passed
Одночасне подання двох заявок на один слот	Перша підтверджується, друга отримує помилку конфлікту	Passed
Перегляд журналу дій адміністратором	Відображено всі записи ActionLog з фільтрацією	Passed

Усі 12 тест-кейсів пройшли успішно. Особливу увагу приділено ТС-06 та ТС-11 – тестам перевірки конфліктів бронювань, що є критичною функцією системи. Автоматизований тест ТС-11 імітує одночасне подання двох заявок в межах однієї транзакції та перевіряє, що механізм `full_clean()` коректно запобігає подвійному бронюванню.

4.6.2 Нефункціональне тестування

Нефункціональне тестування охопило такі аспекти якості системи:

Продуктивність. Час відгуку системи вимірювався за допомогою інструментів браузерних DevTools. Середній час завантаження сторінки каталогу ресурсів (20 записів) склав 380 мс при підключенні через HTTPS на Render, що значно менше встановленого порогу у 2 секунди. Час обробки AJAX-запиту фільтрації – 95 мс. Час подання форми бронювання з перевіркою конфліктів – 210 мс.

Безпека. Перевірено захист від основних вразливостей: CSRF-токени присутні у всіх формах (вбудований механізм Django); SQL-ін'єкції унеможливлені використанням ORM; XSS-атаки блокуються автоматичним екрануванням Django-шаблонів; доступ до захищених URL без автентифікації

повертає перенаправлення на сторінку входу (302); спроба підвищення привілеїв повертає 403 Forbidden. Заголовок Strict-Transport-Security встановлюється платформою Render автоматично.

Надійність CI/CD. За час розробки виконано 47 комітів у репозиторій. GitHub Actions успішно запуслав тести при кожному push; у 3 випадках pipeline виявив регресії до потрапляння коду у виробниче середовище, що підтвердило ефективність автоматизованого тестування.

Висновки до розділу 4.

У четвертому розділі описано реалізацію та тестування web-системи бронювання матеріально-технічних ресурсів навчального закладу Book It!. Наведено структуру проєкту та ключові налаштування Django. Описано реалізацію моделей бази даних: Resource з підтримкою п'яти типів ресурсів та кафедральної приналежності, Booking з автоматичним визначенням статусу ('approved' для звичайних приміщень, 'pending' для конференц-залів), UserProfile з автоматичним визначенням ролі за доменом корпоративної пошти та ActionLog з явним журналюванням через log_action. Розглянуто реалізацію бізнес-логіки у Views з декораторами авторизації, функції управління бронюваннями та маршрутизацію з 16 іменованими маршрутами. Описано клієнтську частину з AJAX-фільтрацією, тижневою сіткою академічних пар та функцією швидкого бронювання quick_book_pair. Охарактеризовано CI/CD-конвеєр на базі GitHub Actions та Render. Проведено функціональне тестування за 12 тест-кейсами (усі пройшли успішно) та нефункціональне тестування продуктивності, безпеки, сумісності браузерів та адаптивності інтерфейсу. Всі встановлені вимоги до системи виконано.

ВИСНОВКИ

У кваліфікаційній роботі розроблено web-систему бронювання матеріально-технічних ресурсів навчального закладу Book It!, що автоматизує процес резервування аудиторій, лабораторій та обладнання з розмежуванням прав доступу між ролями користувачів і автоматичною перевіркою конфліктів бронювань. У ході виконання роботи отримано такі результати.

1. Проведено аналіз предметної галузі управління матеріально-технічними ресурсами навчального закладу. Визначено ключові сутності та процеси: ресурс, користувач, бронювання, конфлікт бронювань, журнал дій. Сформульовано функціональні та нефункціональні вимоги до системи. Виконано огляд та порівняльний аналіз трьох комерційних систем-аналогів: Skedda, Microsoft Bookings та Robin. Встановлено, що жодна з них не відповідає повністю потребам навчального закладу через комерційне ліцензування, відсутність повноцінної трирівневої рольової моделі та складність адаптації під специфіку освітнього процесу.

2. Обґрунтовано вибір технологічного стеку системи Book It!. Для серверної частини обрано Python 3.13 [17] і Django 6.0 завдяки розвиненій екосистемі, вбудованим засобам автентифікації, ORM та системі сигналів. Базою даних визначено SQLite з огляду на масштаб проекту та простоту розгортання без окремого сервера СУБД. Клієнтська частина реалізована на нативних web-технологіях (HTML5, CSS3, JavaScript, Fetch API) без застосування важких JavaScript-фреймворків. Інфраструктурний стек включає Gunicorn 25.3.0 як WSGI-сервер, WhiteNoise 6.12.0 для роздачі статички та Render як хмарну PaaS-платформу з автоматичним HTTPS.

3. Спроектовано архітектуру системи на основі шаблону MVT з чітким розподілом відповідальності між компонентами Model, View та Template. Розроблено діаграму варіантів використання з десятима сценаріями для трьох акторів (студент, викладач, адміністратор). Спроектовано ER-модель бази даних, що включає чотири сутності: Resource, Booking, UserProfile та

ActionLog. Описано діаграму послідовності для ключового сценарію подання та підтвердження бронювання з дворівневим захистом від конфліктів. Розроблено діаграму розгортання, що відображає фізичну архітектуру системи на платформі Render із CI/CD-конвеєром.

4. Реалізовано web-застосунок Book It! з трирівневою рольовою моделлю доступу. Серверна частина включає моделі бази даних з реалізацією автоматичної перевірки конфліктів бронювань через метод `clean()` та механізм Django ORM. Журналювання всіх дій реалізовано через Django Signals без прямих залежностей між модулями. Клієнтська частина забезпечує динамічну фільтрацію ресурсів та оновлення календарної сітки через AJAX-запити на основі Fetch API. Вбудована адміністративна панель Django Admin надає адміністратору повний контроль над ресурсами та користувачами без додаткової розробки.

5. Налаштовано хмарне розгортання системи на платформі Render із застосуванням Gunicorn та WhiteNoise. Реалізовано CI/CD-конвеєр на базі GitHub Actions, що автоматично запускає тести та ініціює розгортання при кожному push у гілку main. За час розробки виконано 47 комітів; у трьох випадках pipeline виявив регресії до потрапляння коду у виробниче середовище, що підтвердило ефективність автоматизованого тестування.

6. Проведено функціональне тестування за 12 тест-кейсами методом «чорного ящика» – усі тести пройшли успішно. Особливу увагу приділено тестуванню механізму перевірки конфліктів бронювань при одночасному поданні заявок. Нефункціональне тестування підтвердило відповідність системи встановленим вимогам: середній час відгуку 380 мс (при нормі 2 с), коректне відображення у чотирьох браузерах, адаптивність інтерфейсу на мобільних пристроях, захист від CSRF, XSS та SQL-ін'єкцій відповідно до рекомендацій OWASP [17].

Практична цінність роботи полягає у створенні готового до впровадження програмного продукту, доступного за адресою у мережі Render. Система не потребує встановлення клієнтського програмного забезпечення і

може бути розгорнута у будь-якому навчальному закладі з мінімальними витратами на адаптацію. Вихідний код системи розміщено у відкритому репозиторії GitHub і може бути використаний як основа для подальшого розвитку.

Перспективами подальшого розвитку системи є: міграція бази даних на PostgreSQL для підтримки більшого навантаження; реалізація сповіщень електронною поштою при зміні статусу бронювання; розробка мобільного додатку на основі REST API; впровадження повторюваних бронювань для регулярних занять; інтеграція із зовнішніми системами розкладу навчального закладу.

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

1. Лоренц В.Я., Компан С.В. Розробка web-системи бронювання матеріально-технічних ресурсів навчального закладу. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез, 26 листопада 2025 р., м. Київ. – К. : ДУІКТ, 2025. – 536 с. – С. 168.

2. Лоренц В.Я., Компан С.В. Автоматизація розгортання Django-додатків за допомогою CI/CD та хмарних платформ. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез, 23 квітня 2026 р., м. Київ. – К. : ДУІКТ, 2026. – 466 с. – С. 461.

ПЕРЕЛІК ПОСИЛАНЬ

1. Django Software Foundation. Django documentation. Version 6.0. [Електронний ресурс]. – Режим доступу: <https://docs.djangoproject.com/en/6.0/> (дата звернення: 10.04.2025).
2. Python Software Foundation. Python 3.13 documentation. [Електронний ресурс]. – Режим доступу: <https://docs.python.org/3.13/> (дата звернення: 10.04.2025).
3. SQLite Consortium. SQLite documentation. [Електронний ресурс]. – Режим доступу: <https://www.sqlite.org/docs.html> (дата звернення: 11.04.2025).
4. Gunicorn contributors. Gunicorn 25.3.0 documentation. [Електронний ресурс]. – Режим доступу: <https://docs.gunicorn.org/> (дата звернення: 12.04.2025).
5. Evans D. WhiteNoise documentation. [Електронний ресурс]. – Режим доступу: <https://whitenoise.readthedocs.io/> (дата звернення: 12.04.2025).
6. Render Inc. Render documentation: Deploying a Django application. [Електронний ресурс]. – Режим доступу: <https://Render/docs/deploy-django> (дата звернення: 17.04.2025).
7. GitHub Inc. GitHub Actions documentation. [Електронний ресурс]. – Режим доступу: <https://docs.github.com/en/actions> (дата звернення: 17.04.2025).
8. Mozilla Developer Network. HTML5 reference. [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Web/HTML> (дата звернення: 14.04.2025).
9. Mozilla Developer Network. Fetch API. [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 14.04.2025).
10. Fowler M. Patterns of Enterprise Application Architecture. – Boston : Addison-Wesley, 2002. – 557 с.
11. Skedda Inc. Skedda – Space Booking Software. [Електронний ресурс]. – Режим доступу: <https://www.skedda.com/> (дата звернення: 03.04.2025).
12. Microsoft Corporation. Microsoft Bookings documentation. [Електронний ресурс]. – Режим доступу: <https://support.microsoft.com/en-us/office/microsoft-bookings> (дата звернення: 03.04.2025).
13. Robin Powered Inc. Robin – Workplace Management Platform. [Електронний ресурс]. – Режим доступу: <https://robinpowered.com/> (дата звернення: 04.04.2025).
14. Vincent W. S. Django for Beginners: Build websites with Python and Django. – Самовидавництво, 2022. – 280 с.

15. Object Management Group. Unified Modeling Language (UML) Specification. Version 2.5.1. [Електронний ресурс]. – Режим доступу: <https://www.omg.org/spec/UML/2.5.1> (дата звернення: 05.04.2025).
16. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. – Київ : ДП «УкрНДНЦ», 2016. – 26 с.
17. OWASP Foundation. OWASP Top Ten. 2021. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 10.04.2026).
18. Wiggins A. The Twelve-Factor App. 2017. URL: <https://12factor.net/> (дата звернення: 17.04.2026).
19. Shahin M., Babar M. A., Zhu L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. IEEE Access.2017.Vol.5.P.3909–3943. URL: <https://doi.org/10.1109/ACCESS.2017.2685629>
20. MDN Web Docs. CSS Grid Layout. Mozilla Developer Network. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout (дата звернення: 14.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інженерії програмного забезпечення



Розробка web-системи бронювання матеріально-технічних ресурсів навчального закладу

Виконав студент 4 курсу
групи ПД-42
Лоренц Владислав Яношович
Керівник роботи
доцент кафедри Інтернет-технологій, кан. фіз.-мат. наук, доц. Компан Сергій Володимирович

Київ - 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесу бронювання матеріально-технічних ресурсів навчального закладу за допомогою розробки web-системи для бронювання.

Об'єкт дослідження: процес управління та бронювання матеріально-технічних ресурсів навчального закладу.

Предмет дослідження: методи та програмні засоби реалізації web-системи бронювання матеріально-технічних ресурсів з використанням фреймворку Django та реляційної системи управління баз даних SQLite.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

1. Провести аналіз предметної області та дослідити існуючі аналоги систем бронювання;
2. Визначити необхідні технології та інструменти програмування для розробки web-застосунку;
3. Розробити загальну структуру системи та спроектувати схеми таблиць бази даних;
4. Реалізувати механізм авторизації користувачів із розмежуванням (прав доступу) різних ролей;
5. Розробити модуль бронювання ресурсів із перевіркою конфліктів часу;
6. Реалізувати функціональність пошуку та фільтрації ресурсів;
7. Забезпечити ведення журналу дій користувачів;
8. Створити зручний та зрозумілий (user friendly) web-інтерфейс;
9. Виконати функціональне тестування всіх модулів системи.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Skedda	Microsoft Bookings	Robin	Book It!
Тип системи	Комерційна	Комерційна	Комерційна	Веб-система для ВНЗ
Орієнтація на навчальні заклади	Частково (базові інструменти для бронювання)	Ні	Ні	Так
Ролі користувачів	Адмін, користувач	Адмін, клієнт	Адмін, співробітник	Адмін, викладач, студент
Перевірка конфліктів	Так	Так	Так	Так
Пошук і фільтрація	Так	Частково (фільтрація обмежена)	Так	Так
Логування дій	Частково (історія бронювань)	Ні	Так	Так
Вартість	Платна	Платна	Платна	Безкоштовна
Адаптація під ВНЗ	Обмежена (налаштування простору)	Ні	Ні	Повна

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

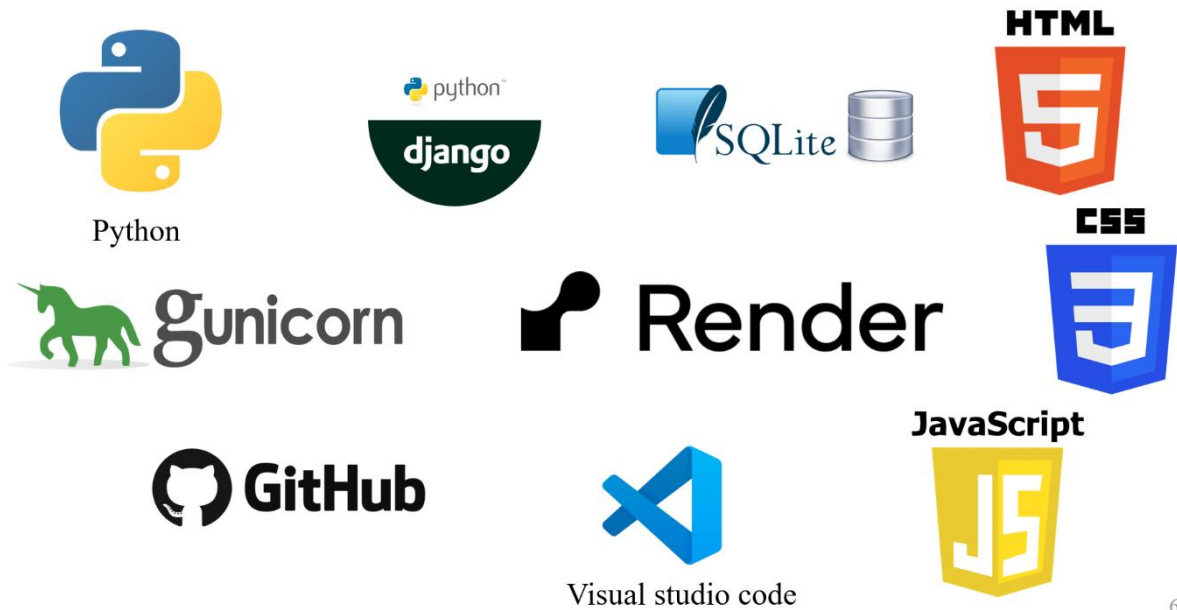
1. Забезпечити реєстрацію та авторизацію користувачів;
2. Реалізувати розмежування доступу за ролями (студент, викладач, адміністратор), які надають відповідні права доступу та можливість перегляду списку доступних ресурсів в залежності від ролі;
3. Реалізувати створення, скасування та перегляд бронювань;
4. Забезпечити перевірку конфліктів часу при бронюванні;
5. Реалізувати функціональність пошуку та фільтрації ресурсів;
6. Надати можливість перегляду розкладу використання ресурсів;
7. Забезпечити підтвердження або відхилення бронювань адміністратором;
8. Реалізувати ведення журналу дій користувачів.

Нефункціональні вимоги:

1. Забезпечити час відгуку системи не більше 2 секунд при створенні або перегляді бронювання;
2. Забезпечити заборону створення бронювання при наявності конфлікту часу (перевірка на рівні серверної логіки);
3. Забезпечити контроль доступу через систему ролей Django (розмежування прав доступу);
4. Забезпечити зручність використання та виконання основних дій не більше ніж за 3-4 кліки;

5

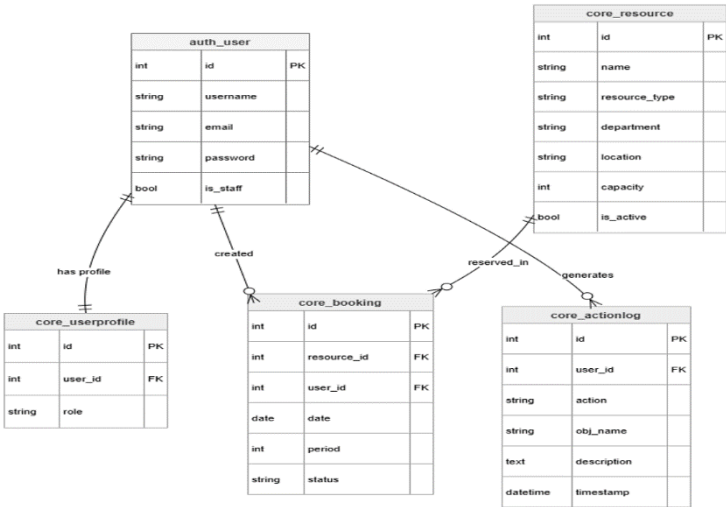
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



6

UML-ДІАГРАМИ СИСТЕМИ

ER-діаграма бази даних



7

UML-ДІАГРАМИ СИСТЕМИ

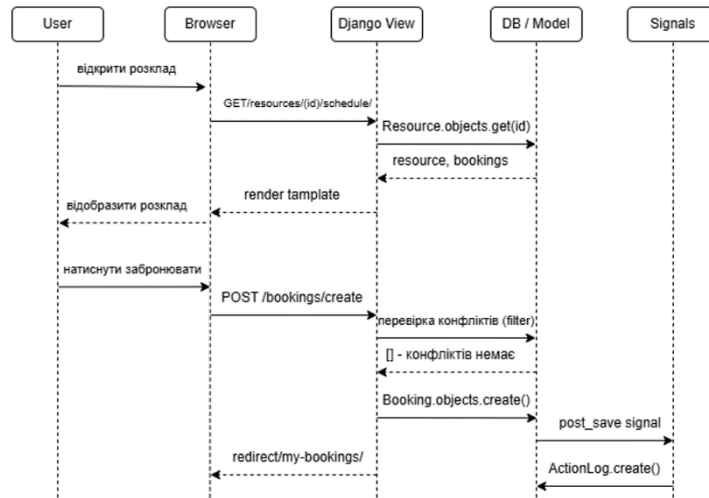
Діаграма варіантів використання



8

ДІАГРАМА ПОСЛІДОВНОСТІ

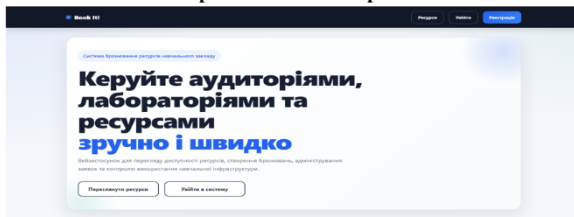
Діаграма послідовності бронювання:



9

ЕКРАННІ ФОРМИ СИСТЕМИ BOOK IT!

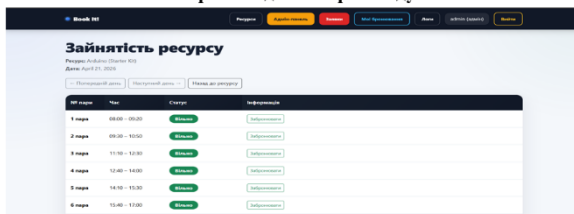
Скріншот головна сторінка



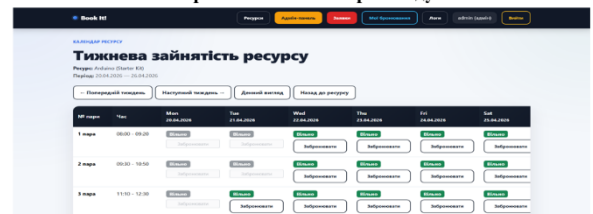
Скріншот каталогу ресурсів



Скріншот денного розкладу



Скріншот тижневого розкладу



10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Лоренц В.Я., Компан С.В. Розробка web-системи бронювання матеріально-технічних ресурсів навчального закладу. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025.
2. Лоренц В.Я., Компан С.В. Автоматизація розгортання Django-додатків за допомогою CI/CD та хмарних платформ. Матеріали Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. (подано до друку).

11

ВИСНОВКИ

1. Досліджено предметну галузь управління матеріально-технічними ресурсами навчального закладу та визначено ключові проблеми: конфлікти бронювань, відсутність прозорості та нераціональне використання аудиторій і обладнання.
2. Розглянуто існуючі програмні рішення для бронювання ресурсів (Skedda, Microsoft Bookings, Robin), визначено їх переваги, недоліки та непридатність для потреб ВНЗ через платну модель та відсутність трирівневої рольової моделі.
3. На основі проведеного аналізу аналогів було сформовано вимоги до системи: каталог ресурсів з фільтрацією, бронювання з перевіркою конфліктів, денний та тижневий розклад, підтвердження заявок адміністратором, журнал дій та розмежування ролей (студент, викладач, адміністратор).
4. На основі проведеного аналізу аналогів було сформовано та обґрунтовано стек реалізації: Python 3.13, Django 6.0, SQLite, HTML5/CSS3, JavaScript (AJAX), Gunicorn, WhiteNoise, Render.com та GitHub CI/CD.
5. Проведено функціональне тестування системи за 12 тест-кейсами методом чорного ящика, усі тести пройшли успішно. Нефункціональне тестування підтвердило відповідність системи вимогам: середній час відгуку 380 мс, коректна робота у браузерях та адаптивний інтерфейс на мобільних пристроях.

12

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

```
)

    booking.delete()
    messages.success(request, 'Бронювання успішно
видалено.')
    return redirect('my_bookings')

@login_required
def delete_inactive_bookings(request):
    now = timezone.now()

    inactive_bookings = Booking.objects.filter(
        user=request.user
    ).filter(
        Q(status='cancelled') |
        Q(status='approved', end_time__lt=now)
    )

    count = inactive_bookings.count()

    if count == 0:
        messages.warning(request, 'У вас немає завершених
або скасованих бронювань для видалення.')
        return redirect('my_bookings')

    for booking in
inactive_bookings.select_related('resource'):
        log_action(
            user=request.user,
            action_type='delete_booking',
            object_type='booking',
            object_repr=f'{booking.resource.name} |
{booking.start_time:%d.%m.%Y %H:%M} -
{booking.end_time:%H:%M}',
            description=f'Користувач {request.user.username}
видалив неактивне бронювання ресурсу
"{booking.resource.name}".'
        )

        inactive_bookings.delete()
        messages.success(request, f'Успішно видалено {count}
завершених/скасованих бронювань.')
        return redirect('my_bookings')

def resource_schedule(request, resource_id):
    resource = get_object_or_404(Resource, id=resource_id,
is_active=True)

    date_str = request.GET.get('date')
    if date_str:
        try:
            selected_date = datetime.strptime(date_str, '%Y-
%m-%d').date()
        except ValueError:
            selected_date = timezone.localdate()
        else:
            selected_date = timezone.localdate()

    day_start =
timezone.make_aware(datetime.combine(selected_date,
time(8, 0)))

    day_end =
timezone.make_aware(datetime.combine(selected_date,
time(20, 0)))

    bookings = Booking.objects.filter(
        resource=resource,
        status__in=['pending', 'approved'],
        start_time__lt=day_end,
        end_time__gt=day_start
    ).select_related('user').order_by('start_time')

    pair_slots = []

    for pair in PAIR_SLOTS:
        slot_start =
timezone.make_aware(datetime.combine(selected_date,
pair['start']))
        slot_end =
timezone.make_aware(datetime.combine(selected_date,
pair['end']))

        slot_booking = None
        occupancy_percent = 0
        alternatives = []

        slot_duration_seconds = (slot_end -
slot_start).total_seconds()

        for booking in bookings:
            overlap_start = max(slot_start, booking.start_time)
            overlap_end = min(slot_end, booking.end_time)

            if overlap_start < overlap_end:
                slot_booking = booking
                overlap_seconds = (overlap_end -
overlap_start).total_seconds()
                occupancy_percent = int((overlap_seconds /
slot_duration_seconds) * 100)
                break

        if slot_booking:
            alternatives = get_available_alternatives(resource,
slot_start, slot_end)

    pair_slots.append({
        'number': pair['number'],
        'start': slot_start,
        'end': slot_end,
        'booking': slot_booking,
        'is_free': slot_booking is None,
        'occupancy_percent': occupancy_percent,
        'alternatives': alternatives,
    })

    prev_date = selected_date - timedelta(days=1)
    next_date = selected_date + timedelta(days=1)

    return render(request, 'resource_schedule.html', {
        'resource': resource,
        'selected_date': selected_date,
        'pair_slots': pair_slots,
```

```

        'prev_date': prev_date,
        'next_date': next_date,
    })

def resource_week_schedule(request, resource_id):
    resource = get_object_or_404(Resource, id=resource_id,
is_active=True)
    selected_date = timezone.localtime()
    else:
        selected_date = timezone.localtime()

    today = timezone.localtime()
    now = timezone.now()

    week_start = selected_date -
timedelta(days=selected_date.weekday())
    week_dates = [week_start + timedelta(days=i) for i in
range(7)]

    schedule_rows = []

    for pair in PAIR_SLOTS:
        row = {
            'pair_number': pair['number'],
            'time_range': f"{pair['start'].strftime('%H:%M')} -
{pair['end'].strftime('%H:%M')}",
            'days': [],
        }

        for current_date in week_dates:
            slot_start =
timezone.make_aware(datetime.combine(current_date,
pair['start']))
            slot_end =
timezone.make_aware(datetime.combine(current_date,
pair['end']))

            booking = Booking.objects.filter(
                resource=resource,
                status__in=['pending', 'approved'],
                start_time__lt=slot_end,
                end_time__gt=slot_start
            ).select_related('user').order_by('start_time').first()

            alternatives = []
            if booking:
                alternatives =
get_available_alternatives(resource, slot_start, slot_end)

            is_past = current_date < today or slot_end <= now

            row['days'].append({
                'date': current_date,
                'booking': booking,
                'is_free': booking is None,
                'slot_start': slot_start,
                'slot_end': slot_end,
                'alternatives': alternatives,
                'is_past': is_past,
            })

    schedule_rows.append(row)

```

```

date_str = request.GET.get('date')
if date_str:
    try:
        selected_date = datetime.strptime(date_str, '%Y-
%m-%d').date()
    except ValueError:

prev_week = week_start - timedelta(days=7)
next_week = week_start + timedelta(days=7)

return render(request, 'resource_week_schedule.html', {
    'resource': resource,
    'week_dates': week_dates,
    'schedule_rows': schedule_rows,
    'week_start': week_start,
    'prev_week': prev_week,
    'next_week': next_week,
})

@login_required
def quick_book_pair(request, resource_id, pair_number):
    resource = get_object_or_404(Resource, id=resource_id,
is_active=True)

    if request.method != 'POST':
        return JsonResponse(
            {'success': False, 'message': 'Метод POST
обов'язковий.'},
            status=405
        )

    if not (request.user.is_superuser or request.user.is_staff):
        if not hasattr(request.user, 'profile') or
request.user.profile.role != 'teacher':
            return JsonResponse({
                'success': False,
                'message': 'Швидке бронювання доступне
лише викладачам.'
            }, status=403)

    date_str = request.POST.get('date')
    if date_str:
        try:
            selected_date = datetime.strptime(date_str, '%Y-
%m-%d').date()
        except ValueError:
            return JsonResponse({
                'success': False,
                'message': 'Невірний формат дати.'
            }, status=400)
        else:
            selected_date = timezone.localtime()

    selected_pair = next((pair for pair in PAIR_SLOTS if
pair['number'] == pair_number), None)

    if not selected_pair:
        return JsonResponse({
            'success': False,
            'message': 'Невірно обрано пару.'
        }, status=400)

```

```

start_dt =
timezone.make_aware(datetime.combine(selected_date,
selected_pair['start']))
end_dt =
timezone.make_aware(datetime.combine(selected_date,
selected_pair['end']))

conflict_exists = Booking.objects.filter(
    resource=resource,
    status__in=['pending', 'approved'],
    start_time__lt=end_dt,
    end_time__gt=start_dt
).exists()

if conflict_exists:
    alternatives = get_available_alternatives(resource,
start_dt, end_dt)

    return JsonResponse({
        'success': False,
        'message': 'Ця пара вже зайнята для обраного
ресурсу.',
        'alternatives': [
            {
                'id': alt.id,
                'name': alt.name,
            }
            for alt in alternatives
        ]
    }, status=400)

if resource.resource_type == 'conference_room':
    status = 'pending'
    status_label = 'Очікує підтвердження'
    message = f'Заявку на бронювання пари
{pair_number} створено та передано на підтвердження.'
else:
    status = 'approved'
    status_label = 'Зайнято'
    message = f'Пару {pair_number} успішно
заброньовано та автоматично підтверджено.'

booking = Booking.objects.create(
    resource=resource,
    user=request.user,
    start_time=start_dt,
    end_time=end_dt,
    status=status
)

log_action(
    user=request.user,
    action_type='create_booking',
    object_type='booking',
    object_repr=f'{resource.name} |
{booking.start_time:%d.%m.%Y %H:%M} -
{booking.end_time:%H:%M}',
    description=f'Швидке бронювання пари
{pair_number} для ресурсу "{resource.name}"
користувачем {request.user.username}.'
)

return JsonResponse({
    'success': True,

```

```

'message': message,
'booking': {
    'id': booking.id,
    'status': booking.status,
    'status_label': status_label,
    'username': request.user.username,
    'start_time': booking.start_time.strftime("%H:%M"),
    'end_time': booking.end_time.strftime("%H:%M"),
}
})

@staff_member_required
def admin_pending_bookings(request):
    bookings = Booking.objects.select_related('resource',
'user').filter(status='pending').order_by('start_time')
    return render(request, 'admin_pending_bookings.html',
{'bookings': bookings})

@staff_member_required
def approve_booking(request, booking_id):
    booking = get_object_or_404(Booking, id=booking_id)

    if booking.status != 'pending':
        messages.warning(request, 'Підтвердити можна
лише бронювання зі статусом "Очікує
підтвердження".')
        return redirect('admin_pending_bookings')

    booking.status = 'approved'
    booking.save(update_fields=['status'])

    log_action(
        user=request.user,
        action_type='approve_booking',
        object_type='booking',
        object_repr=f'{booking.resource.name} |
{booking.start_time:%d.%m.%Y %H:%M} -
{booking.end_time:%H:%M}',
        description=f'Адміністратор {request.user.username}
підтвердив бронювання користувача
{booking.user.username}.'
    )

    messages.success(request, 'Бронювання
підтверджено.')
    return redirect('admin_pending_bookings')

@staff_member_required
def reject_booking(request, booking_id):
    booking = get_object_or_404(Booking, id=booking_id)

    if booking.status != 'pending':
        messages.warning(request, 'Відхилити можна лише
бронювання зі статусом "Очікує підтвердження".')
        return redirect('admin_pending_bookings')

    booking.status = 'rejected'
    booking.save(update_fields=['status'])

    log_action(
        user=request.user,
        action_type='reject_booking',

```

```

        object_type='booking',
        object_repr=f'{booking.resource.name} |
{booking.start_time:%d.%m.%Y %H:%M} -
{booking.end_time:%H:%M}',
        description=f'Адміністратор {request.user.username}
відхилив бронювання користувача
{booking.user.username}.'
    )

    messages.success(request, 'Бронювання відхилено.')
    return redirect('admin_pending_bookings')

@staff_member_required
def action_logs(request):
    logs = ActionLog.objects.select_related('user').all()[:300]
    return render(request, 'action_logs.html', {'logs': logs})
forms.py
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User

from .models import Booking

class BookingForm(forms.ModelForm):
    class Meta:
        model = Booking
        fields = ['start_time', 'end_time']
        widgets = {
            'start_time': forms.DateTimeInput(
                attrs={'type': 'datetime-local', 'class': 'form-
control'}
            ),
            'end_time': forms.DateTimeInput(
                attrs={'type': 'datetime-local', 'class': 'form-
control'}
            ),
        }
        labels = {
            'start_time': 'Початок бронювання',
            'end_time': 'Кінець бронювання',
        }

    def clean(self):
        cleaned_data = super().clean()
        start_time = cleaned_data.get('start_time')
        end_time = cleaned_data.get('end_time')

        if start_time and end_time:
            if end_time <= start_time:
                raise forms.ValidationError('Час завершення
має бути пізніше за час початку.')

        resource = self.initial.get('resource')

        if resource:
            conflicts = Booking.objects.filter(
                resource=resource,
                status__in=['pending', 'approved'],
                start_time__lt=end_time,
                end_time__gt=start_time
            )

```

```

        if conflicts.exists():
            raise forms.ValidationError(
                'Цей ресурс вже заброньований у
вказаний час.'
            )
    )

    return cleaned_data

```

```

class RegisterForm(UserCreationForm):
    email = forms.EmailField(
        label='Корпоративна електронна пошта',
        widget=forms.EmailInput(attrs={'class': 'form-
control'})
    )
    username = forms.CharField(
        label='Ім'я користувача',
        widget=forms.TextInput(attrs={'class': 'form-
control'})
    )
    password1 = forms.CharField(
        label='Пароль',
        widget=forms.PasswordInput(attrs={'class': 'form-
control'})
    )
    password2 = forms.CharField(
        label='Підтвердження пароля',
        widget=forms.PasswordInput(attrs={'class': 'form-
control'})
    )

STUDENT_DOMAIN = 'stud.duikt.edu.ua'
TEACHER_DOMAIN = 'duikt.edu.ua'

class Meta:
    model = User
    fields = ['username', 'email', 'password1', 'password2']

    def clean_email(self):
        email = self.cleaned_data['email'].lower().strip()

        try:
            domain = email.split('@')[1]
        except IndexError:
            raise forms.ValidationError('Невірний формат
електронної пошти.')

        if domain not in [self.STUDENT_DOMAIN,
self.TEACHER_DOMAIN]:
            raise forms.ValidationError(
                f'Реєстрація дозволена лише для адрес доменів
@{self.STUDENT_DOMAIN} або
@{self.TEACHER_DOMAIN}.'
            )

        if User.objects.filter(email=email).exists():
            raise forms.ValidationError('Користувач з такою
електронною поштою вже існує.')

    return email

    def get_role_by_email(self):
        email = self.cleaned_data['email'].lower().strip()
        domain = email.split('@')[1]

```

```

        if domain == self.STUDENT_DOMAIN:
            return 'student'
        return 'teacher'

    def save(self, commit=True):
        user = super().save(commit=False)
        user.email = self.cleaned_data['email']
        user.is_active = False

        if commit:
            user.save()

        return user
signals.py
from django.contrib.auth.signals import user_logged_in,
user_logged_out
from django.dispatch import receiver

from .logging_utils import log_action

@receiver(user_logged_in)
def handle_user_logged_in(sender, request, user,
**kwargs):
    log_action(
        user=user,
        action_type='login',
        object_type='auth',
        object_repr='Вхід у систему',
        description=f'Користувач {user.username} увійшов у
систему.'
    )

@receiver(user_logged_out)
def handle_user_logged_out(sender, request, user,
**kwargs):
    if user:
        log_action(
            user=user,
            action_type='logout',
            object_type='auth',
            object_repr='Вихід із системи',
            description=f'Користувач {user.username} вийшов
із системи.'
        )
urls.py
from django.urls import path
from .views import (
    home,
    resource_list,
    resource_detail,
    create_booking,
    register_view,
    my_bookings,
    cancel_booking,
    delete_booking,
    delete_inactive_bookings,
    resource_schedule,
    resource_week_schedule,
    quick_book_pair,
    admin_pending_bookings,
    approve_booking,
    reject_booking,

```

```

    action_logs,
)

urlpatterns = [
    path("", home, name='home'),

    path('resources/', resource_list, name='resource_list'),
    path('resources/<int:resource_id>', resource_detail,
name='resource_detail'),
    path('resources/<int:resource_id>/book/',
create_booking, name='create_booking'),
    path('resources/<int:resource_id>/schedule/',
resource_schedule, name='resource_schedule'),
    path('resources/<int:resource_id>/week/',
resource_week_schedule,
name='resource_week_schedule'),
    path('resources/<int:resource_id>/quick-
book/<int:pair_number>', quick_book_pair,
name='quick_book_pair'),

    path('accounts/register/', register_view, name='register'),

    path('my-bookings/', my_bookings,
name='my_bookings'),
    path('my-bookings/<int:booking_id>/cancel/',
cancel_booking, name='cancel_booking'),
    path('my-bookings/<int:booking_id>/delete/',
delete_booking, name='delete_booking'),
    path('my-bookings/delete-inactive/',
delete_inactive_bookings,
name='delete_inactive_bookings'),

    path('logs/', action_logs, name='action_logs'),

    path('dashboard/pending-bookings/',
admin_pending_bookings,
name='admin_pending_bookings'),
    path('dashboard/pending-
bookings/<int:booking_id>/approve/', approve_booking,
name='approve_booking'),
    path('dashboard/pending-
bookings/<int:booking_id>/reject/', reject_booking,
name='reject_booking'),
]
admin.py
from django.contrib import admin
from .models import Resource, Booking, UserProfile,
ActionLog

@admin.register(Resource)
class ResourceAdmin(admin.ModelAdmin):
    list_display = (
        'name',
        'resource_type',
        'department',
        'location',
        'capacity',
        'is_active',
        'created_at',
    )
    list_filter = ('resource_type', 'department', 'is_active')
    search_fields = ('name', 'location')

```

```
@admin.register(Booking)
class BookingAdmin(admin.ModelAdmin):
    list_display = ('resource', 'user', 'start_time', 'end_time',
'status')
    list_filter = ('status', 'resource')
    search_fields = ('resource__name', 'user__username')
```

```
@admin.register(UserProfile)
class UserProfileAdmin(admin.ModelAdmin):
    list_display = ('user', 'role')
```

```
@admin.register(ActionLog)
class ActionLogAdmin(admin.ModelAdmin):
    list_display = ('created_at', 'user', 'action_type',
'object_type', 'object_repr')
    list_filter = ('action_type', 'object_type')
    search_fields = ('user__username', 'object_repr',
'description')
settings.py
from pathlib import Path
import os
```

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
SECRET_KEY = 'django-insecure-change-me'
```

```
DEBUG = True
```

```
ALLOWED_HOSTS = ['*']
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    'core.apps.CoreConfig',
]
```

```
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
```

```
'django.contrib.sessions.middleware.SessionMiddleware',
'django.middleware.common.CommonMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
```

```
'django.contrib.auth.middleware.AuthenticationMiddleware',
,
```

```
'django.contrib.messages.middleware.MessageMiddleware',
```

```
'django.middleware.clickjacking.XFrameOptionsMiddlewa
re',
]
```

```
ROOT_URLCONF = 'config.urls'
```

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

```
WSGI_APPLICATION = 'config.wsgi.application'
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

```
AUTH_PASSWORD_VALIDATORS = [
    {'NAME':
'django.contrib.auth.password_validation.UserAttributeSim
ilarityValidator'},
    {'NAME':
'django.contrib.auth.password_validation.MinimumLength
Validator'},
    {'NAME':
'django.contrib.auth.password_validation.CommonPasswor
dValidator'},
    {'NAME':
'django.contrib.auth.password_validation.NumericPasswor
dValidator'},
]
```

```
LANGUAGE_CODE = 'en-us-uk'
TIME_ZONE = 'UTC'
```

```
USE_I18N = True
USE_TZ = True
```

```
STATIC_URL = 'static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = [BASE_DIR / 'static']
```

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
```

```
DEFAULT_AUTO_FIELD =
'django.db.models.BigAutoField'
```