ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Цифрова платформа для планування та резервування зустрічей на базі фреймворку Next.js»

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

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

_____ Костянтин БІЛЕЦЬКИЙ
(підпис)

Виконав: здобувач(ка) вищої освіти групи ТЦР-41

Костянтин БІЛЕЦЬКИЙ

Керівник: Микола ГЕРЦЮК
д-р філософії

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Білецькому Костянтину Євгеновичу

1. Тема кваліфікаційної роботи: «Цифрова платформа для планування та резервування зустрічей на базі фреймворку Next.js»
керівник кваліфікаційної роботи д-р філософії Микола ГЕРЦЮК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від « 20 » лютого 2026 року № _51__.

2. Строк подання кваліфікаційної роботи «___» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи онлайн-бронювання, технології автоматичного оновлення доступності, інтеграцію з календарями та технічну документацію фреймворків Next.js та Nylas.

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

1. Аналіз предметної області та існуючих систем онлайн-бронювання.
2. Вимоги до програмного забезпечення та проектування архітектури додатку.
3. Програмна реалізація веб-сервісу BKalendar.
4. Тестування та оцінка розробленого додатку.

5. Перелік графічного матеріалу: *презентація*

6. Дата видачі завдання «_20_» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	20.02-25.02.2026	
2	Аналіз та дослідження існуючих аналогів	20.02-05.03.2026	
3	Визначення вимог та проєктування архітектури системи	01.03-15.03.2026	
4	Проектування бази даних та API	10.03-20.03.2026	
5	Програмна реалізація застосунку	15.03-10.04.2026	Основний етап розробки
6	Інтеграція з Nylas, реальний час та додатковий функціонал	05.04-15.04.2026	
7	Тестування, виправлення помилок та оптимізація	10.04-25.04.2026	
8	Оформлення роботи: вступ, висновки, реферат. Розробка демонстраційних матеріалів	20.04-10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач(ка) вищої освіти _____ Костянтин БІЛЕЦЬКИЙ
(підпис)

Керівник
кваліфікаційної роботи _____ Микола ГЕРЦЮК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: _60_ стор., __5__ табл., __46__ рис., __26__ джерела.

Мета роботи – підвищити ефективність планування та бронювання зустрічей шляхом розробки сучасної цифрової платформи BKalendar, яка дозволяє суттєво спростити процес узгодження часу, автоматизувати визначення вільних слотів та забезпечити глибоку інтеграцію з календарними сервісами, тим самим зменшити час на організацію зустрічей і підвищити продуктивність роботи фрілансерів, викладачів та бізнес-команд.

Об'єкт дослідження – процеси онлайн-бронювання зустрічей, управління доступністю користувачів та обмін даними в реальному часі.

Предмет дослідження – програмне забезпечення для онлайн-бронювання зустрічей з підтримкою гнучкого налаштування доступності, автоматичного визначення зайнятих слотів та інтеграцією з календарними сервісами.

Короткий зміст роботи: Проведено аналіз сучасних підходів до онлайн-бронювання зустрічей та систем управління доступністю. Досліджено інтеграцію з календарними сервісами та методи автоматичного визначення зайнятих слотів з використанням Nylas.

Проаналізовано існуючі рішення у сфері онлайн-бронювання (Calendly, YouCanBook.me, Google Calendar), їх переваги та недоліки. Спроектовано архітектуру веб-сервісу BKalendar з використанням технологій Next.js 16 [16], Nylas [17], MongoDB [14], Tailwind CSS [22] та date-fns [6].

Реалізовано функціонал створення типів подій, гнучкого налаштування доступності по днях тижня, автоматичного визначення та блокування зайнятих слотів, інтерактивного календаря з вибором часу, генерації персональних посилань для бронювання, інтеграції з Google Meet, системи перегляду заброньованих зустрічей та профілю користувача з власним, налаштовуваним ім'ям користувача.

Спроектовано базу даних для зберігання типів подій, профілів користувачів, заброньованих зустрічей та налаштувань доступності. Реалізовано серверну та клієнтську частини застосунку з використанням Next.js App Router, Server Components, API Routes та MongoDB.

Розроблений веб-сервіс BKalendar може бути використаний як самостійне рішення для особистих та бізнес-потреб, а також як основа для створення корпоративних систем бронювання.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, NEXT.JS, NYLAS, GOOGLE MEET, ОНЛАЙН-БРОНЮВАННЯ, ДОСТУПНІСТЬ, ЗАЙНЯТІ СЛОТИ, РЕАЛЬНИЙ ЧАС, ПРОФІЛЬ КОРИСТУВАЧА, КАЛЕНДАР.

ЗМІСТ

ВСТУП

Актуальність теми. Сьогодні онлайн-бронювання зустрічей є невід'ємною частиною як особистого, так і професійного життя. Зростання популярності віддаленої роботи, фрілансу та гібридних форматів взаємодії призвело до значного збільшення попиту на зручні та гнучкі системи планування. Однак існуючі рішення, такі як Calendly [4], YouCanBook.me [26] та Google Calendar [8], часто мають обмеження: недостатню гнучкість налаштування доступності, відсутність автоматичного оновлення доступності, слабку інтеграцію з календарями або незручний інтерфейс.

Особливо актуальною є проблема автоматичного визначення зайнятих слотів, коли користувач має декілька календарів, також є потреба в персоналізації посилань для бронювання та зручному управлінні профілем. Користувачі очікують не просто інструмент для створення посилань, а повноцінну систему з гнучким управлінням часом, інтеграцією з онлайн-зустрічами та можливістю тонкого налаштування під власні потреби.

Тому розробка сучасного веб-додатку для онлайн-бронювання зустрічей, який поєднує гнучке налаштування доступності, технології автоматичного оновлення та зручний інтерфейс, є актуальним завданням.

Об'єкт дослідження – процеси онлайн-бронювання зустрічей, управління доступністю користувачів та обмін даними в реальному часі в інформаційних системах.

Предмет дослідження – програмне забезпечення для онлайн-бронювання зустрічей з підтримкою гнучкого налаштування доступності, автоматичного визначення зайнятих слотів та інтеграції з календарними сервісами.

Метою роботи є проектування та розробка сучасного веб-сервісу для онлайн-бронювання зустрічей з гнучким управлінням доступністю, підтримкою технологій автоматичного оновлення доступності та створенням онлайн-зустрічей.

Для досягнення мети необхідно:

1. Провести аналіз існуючих систем онлайн-бронювання.
2. Дослідити недоліки та обмеження популярних рішень.
3. Визначити основні вимоги.
4. Розробити структуру бази даних.
5. Спроектувати архітектуру програмного забезпечення.
6. Реалізувати функціонал створення типів подій та гнучкого налаштування доступності по днях тижня.
7. Розробити механізм автоматичного визначення та блокування зайнятих слотів з інтеграцією до календаря користувача.
8. Створити інтерактивний календар з вибором часу та генерацію персональних посилань для бронювання.
9. Реалізувати систему профілю користувача.
10. Провести тестування розробленого веб-сервісу та оцінити його ефективність.

Методами дослідження є методи аналізу та узагальнення існуючих підходів до онлайн-бронювання, методи об’єктно-орієнтованого проектування, методи розробки веб-застосунків на основі Next.js, методи інтеграції з зовнішніми сервісами (Nylas), а також методи тестування програмного забезпечення.

Практична значущість результатів полягає у створенні повноцінного веб-додатка BKalendar, який може використовуватись як самостійне рішення для особистих та бізнес-потреб, а також як основа для створення корпоративних систем бронювання. Розроблений сервіс дозволить підвищити ефективність планування зустрічей, зменшити кількість непотрібних листувань та навантаження на користувачів. Завдяки реалізації гнучкого налаштування доступності по днях тижня, автоматичного визначення та блокування зайнятих часових слотів, генерації персоналізованих посилань для бронювання та інтеграції з платформою Nylas для автоматичного створення зустрічей у Google Meet, система суттєво спрощує процес планування зустрічей. Це дозволяє значно скоротити час на узгодження розкладів з днів/годин до кількох кліків, мінімізувати ризик подвійного бронювання, зменшити кількість непотрібних листувань та

загальне навантаження на користувачів. Крім того, реалізована можливість створення персонального профілю з унікальним іменем користувача (username) забезпечує зручну персоналізацію посилань для бронювання, а адаптивний інтерфейс робить сервіс комфортним для використання на комп'ютері, планшеті та мобільних пристроях. У результаті розроблене рішення підвищує загальну ефективність роботи команд і спеціалістів, забезпечуючи високий рівень зручності, автоматизації та надійності процесу бронювання зустрічей.

Апробація результатів та публікації. Основні положення і результати бакалаврської роботи доповідались на наступних науково-практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій:

1. Білецький К.Є., Герцюк М.М. Визначення методу планування та резервування зустрічей у цифровій платформі на базі Next.js. // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 20.04.2026

2. Білецький К.Є., Герцюк М.М. Інтеграція Nylas API для синхронізації календарів та автоматизації резервування зустрічей у цифровій платформі на базі Next.js. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». 25.04.2026

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

1.1 Опис предметної області

Онлайн зустрічі вже є невід'ємною частиною особистого та професійного життя для більшості людей, працюючих в інформаційних та інших сферах. Зростання популярності фрілансу, гібридних форматів зайнятості та міжнародної співпраці призвело до значного збільшення потреби в зручних і ефективних інструментах планування.

Процес планування зустрічей часто супроводжувався значними труднощами: необхідністю вести тривале листування для узгодження зручного часу, урахування різних часових зон, а також постійного моніторингу власного календаря. Такі дії забирають чимало часу та створюють додаткове навантаження як для організатора, так і для учасників зустрічі.

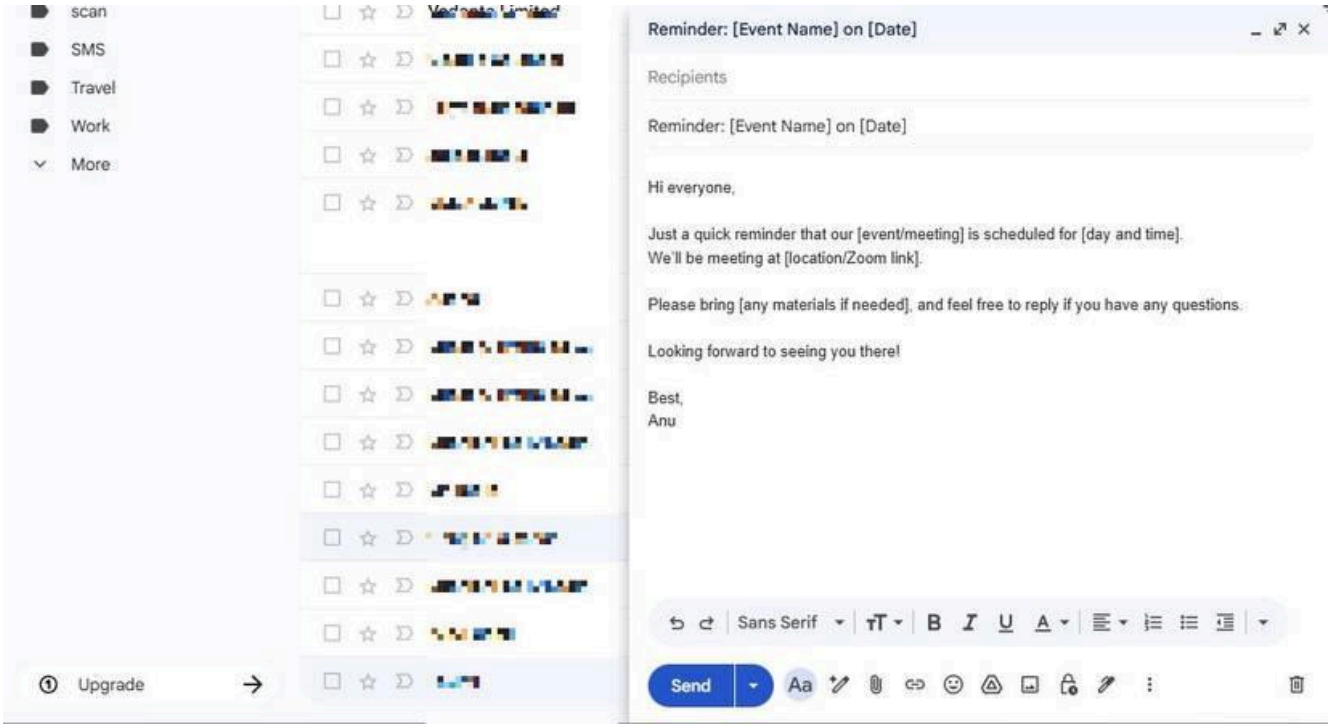


Рис. 1.1 Приклад типового процесу планування зустрічі через електронну пошту з великою кількістю листів (Gmail)

Онлайн-бронювання охоплює сукупність процесів, пов’язаних зі створенням, налаштуванням та використанням систем для планування зустрічей у цифровому середовищі. Основними елементами такої системи є управління доступністю, генерація посилань для бронювання, інтеграція з календарями, автоматичне створення онлайн-зустрічей та управління профілем користувача.

Управління доступністю передбачає можливість гнучкого налаштування вільних часових слотів користувачем по днях тижня та годинах. Генерація посилань для бронювання дозволяє створювати персоналізовані URL-адреси, за якими інші користувачі можуть самостійно обирати зручний час для зустрічі. Інтеграція з календарями забезпечує синхронізацію зі сторонніми сервісами (Google Calendar, Outlook та іншими) для автоматичного визначення вже зайнятих слотів. Автоматичне створення онлайн-зустрічей реалізується через інтеграцію з платформами відеозв’язку, наприклад Google Meet. Управління профілем користувача включає можливість налаштування персональних даних та створення унікального імені користувача (username), що дає змогу формувати зручні та легко-запам’ятовуванні персональні посилання для бронювання.

Користувачі очікують від систем бронювання не лише базового функціоналу, а й високого рівня гнучкості та автоматизації. Зокрема, важливими є можливості тонкого налаштування доступності по днях тижня, автоматичне блокування вже зайнятих слотів, а також зручний інтерфейс для перегляду заброньованих зустрічей.

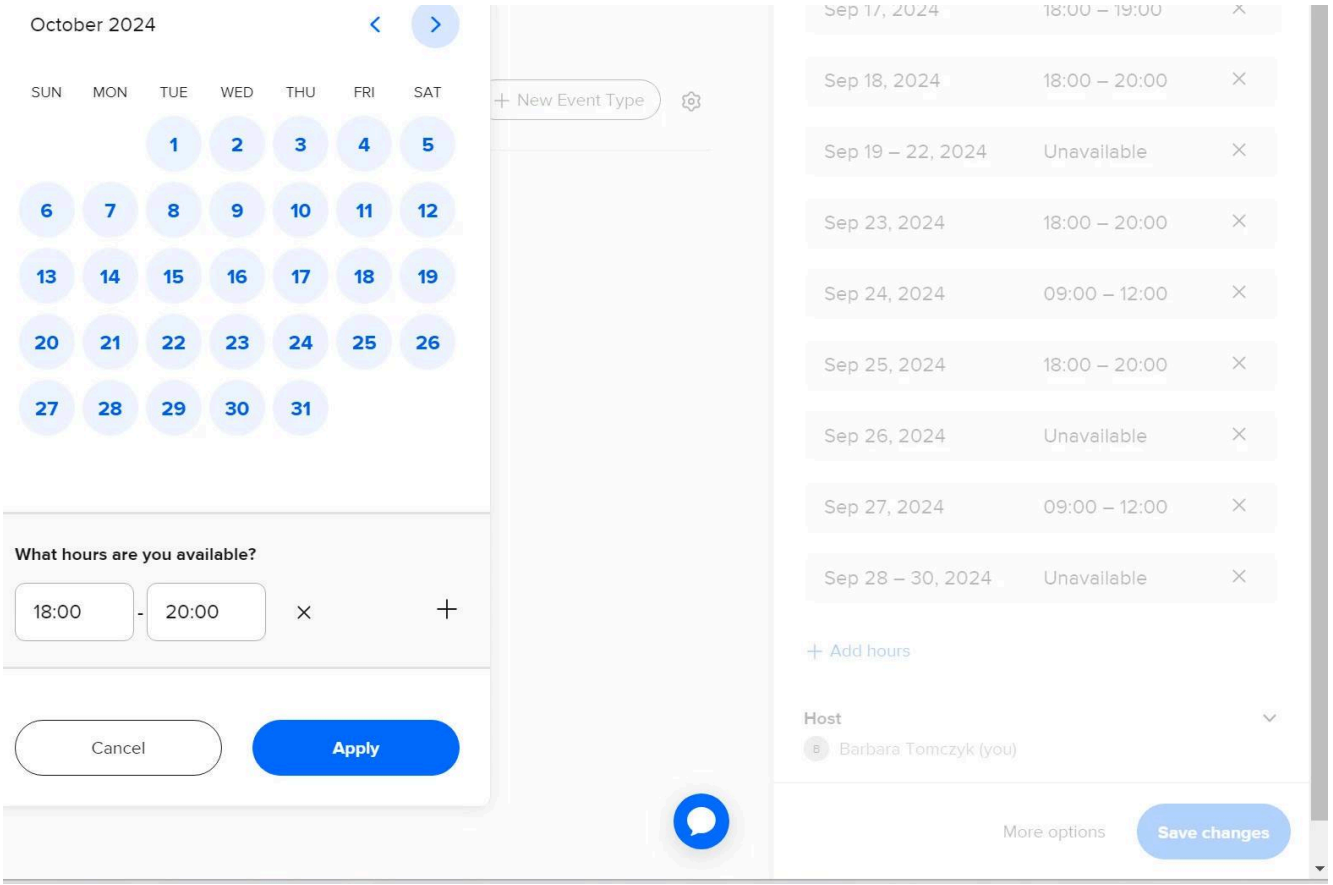


Рис. 1.2 Приклад сучасного інтерфейсу системи онлайн-бронювання з календарем і налаштуваннями доступності

Таким чином, предметна область онлайн-бронювання зустрічей є складною та комплексною, що обумовлює необхідність розробки сучасних веб-сервісів, здатних ефективно вирішувати існуючі проблеми планування та відповідати зростаючим вимогам користувачів.

1.2 Аналіз існуючих систем онлайн-бронювання

Для визначення сильних та слабких сторін сучасних рішень у сфері онлайн-бронювання було проведено детальний аналіз трьох найпопулярніших систем: Calendly [4], YouCanBook.me [26] та Google Calendar (з функцією Appointment Slots) [8]. Аналіз проводився за такими критеріями: функціональність, зручність інтерфейсу, гнучкість налаштування доступності,

інтеграційні можливості (зовнішні сервіси чи системи), автоматизованість та додаткові функції.

Calendly є однією з найвідоміших платформ для онлайн-бронювання зустрічей. Сервіс дозволяє користувачам створювати різні типи подій, гнучко налаштовувати свою доступність, генерувати персональні посилання для бронювання, а також інтегруватися з більшістю популярних календарних сервісів і інструментів комунікації.

На головній сторінці користувач може створювати нові типи подій, переглядати статистику бронювань та керувати своїми посиланнями (рис. 1.3).

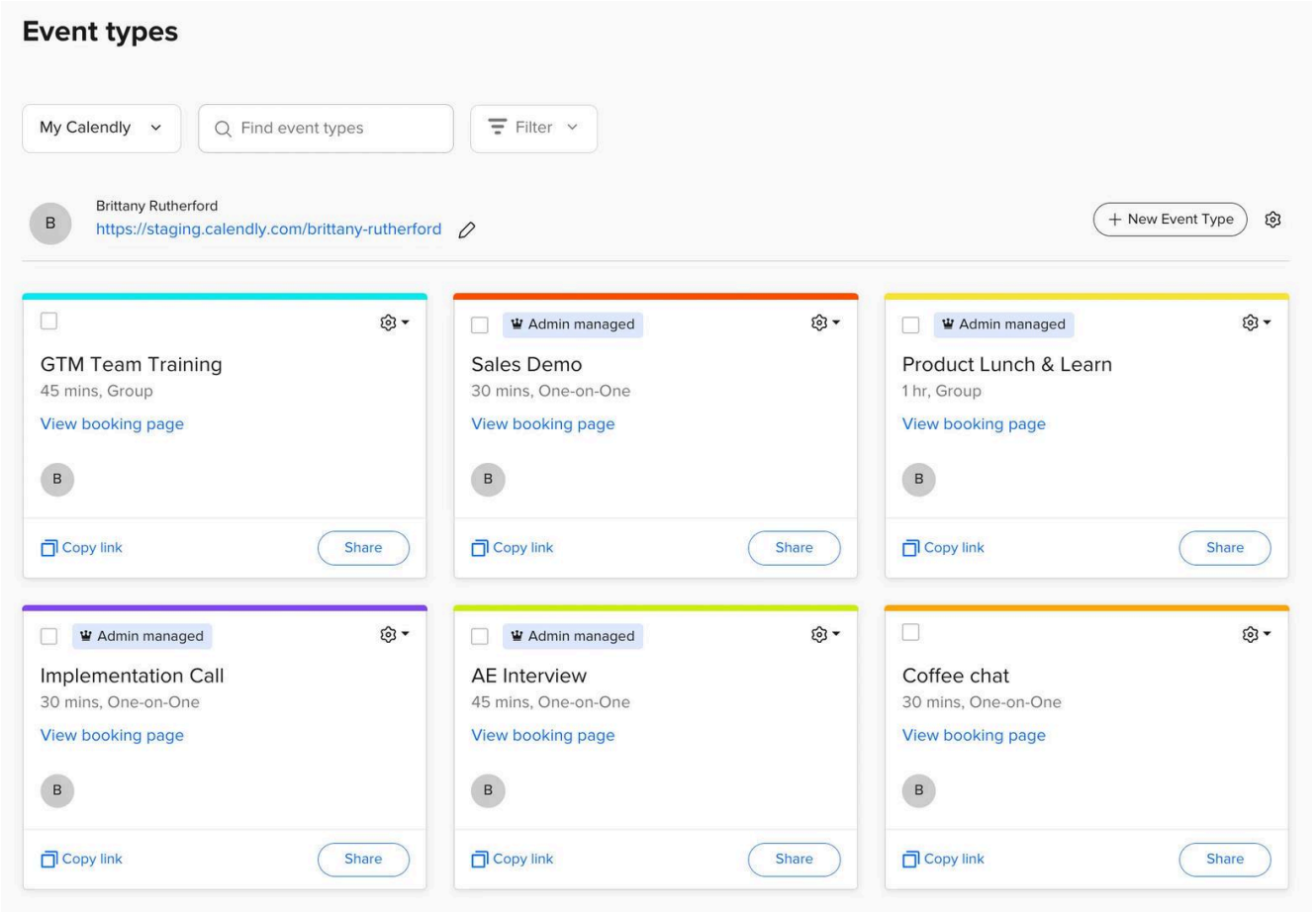


Рис. 1.3 Головна сторінка Calendly з переліком типів подій

Система має потужний інструмент налаштування доступності. Користувач може вказувати робочі години, додавати буфери між зустрічами, налаштовувати мінімальний і максимальний час бронювання, а також створювати кілька різних розкладів (рис. 1.4).

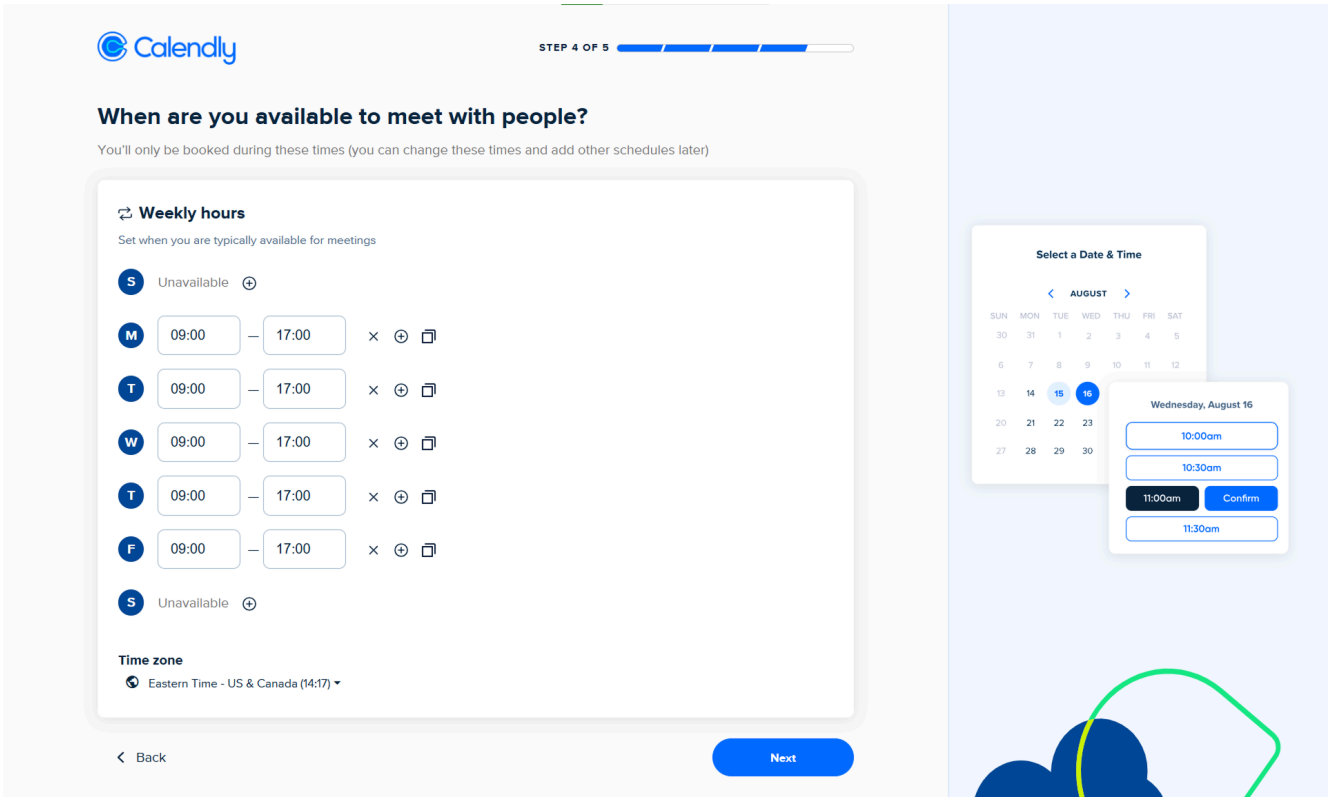


Рис. 1.4 Налаштування доступності в Calendly

Calendly підтримує створення різних типів подій: одноразові зустрічі, групові сесії, колективне планування та круглі столи. Кожний тип події має власні налаштування тривалості, локації та питань для учасників (рис. 1.5).

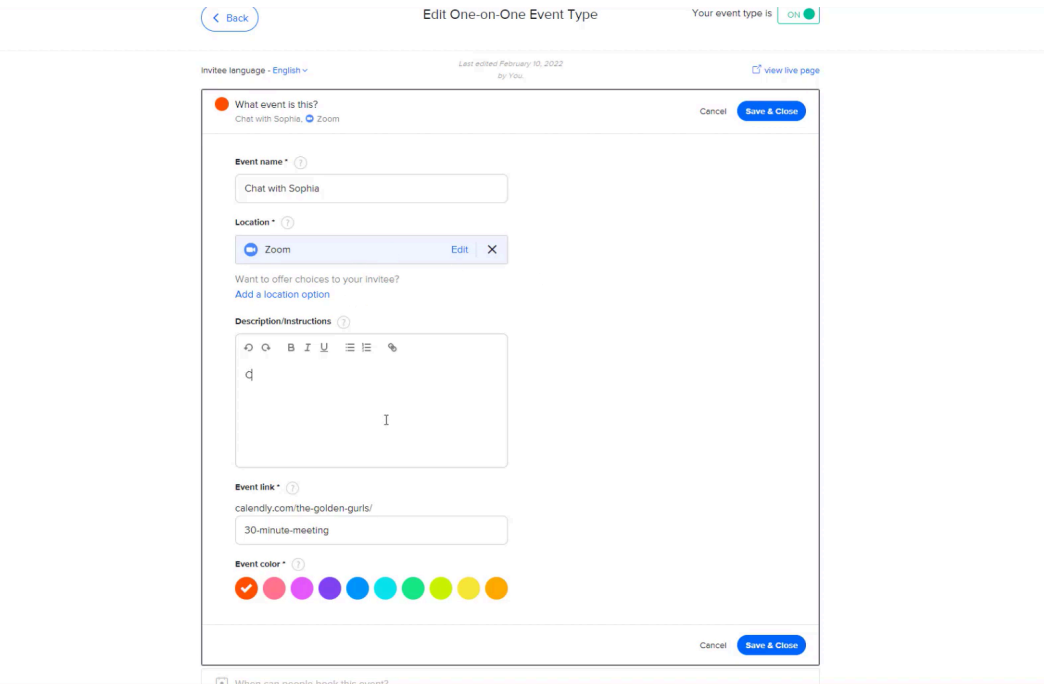


Рис. 1.5 Створення нового типу події в Calendly

Calendly вирізняється сучасним та інтуїтивно зрозумілим інтерфейсом, високою гнучкістю налаштування доступності, а також широкими можливостями інтеграції з популярними сервісами, такими як Google Calendar, Outlook, Zoom та Slack. Платформа підтримує створення різних типів подій і пропонує зручну аналітику бронювань.

Водночас сервіс має ряд суттєвих обмежень. У безкоштовній версії функціонал значно урізаний. Відсутня командна робота та обмежена кількість типів подій. Крім того, Calendly не забезпечує автоматичного оновлення доступності з кількох календарів одночасно, не дозволяє змінювати формат персонального посилання (залишається тільки calendly.com/ім'я), а вартість платної підписки є досить високою, від 8 до 16 доларів США на місяць за одного користувача.

YouCanBook.me це платформа, орієнтована насамперед на простоту та швидкість впровадження. Сервіс дозволяє користувачам швидко створювати сторінки бронювання та легко підключати їх до Google Calendar.

Система YouCanBook.me має досить простий інтерфейс створення сторінки бронювання (рис. 1.6).

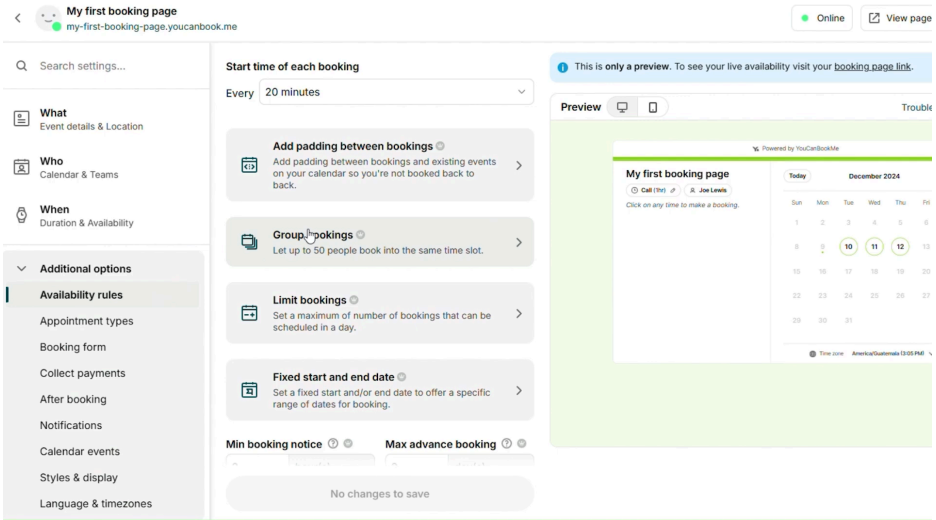


Рис. 1.6 Інтерфейс створення сторінки бронювання в YouCanBook.me

Користувач може налаштовувати доступність, додавати питання для учасників та змінювати зовнішній вигляд сторінки. Однак можливості персоналізації значно обмежені (рис. 1.7).

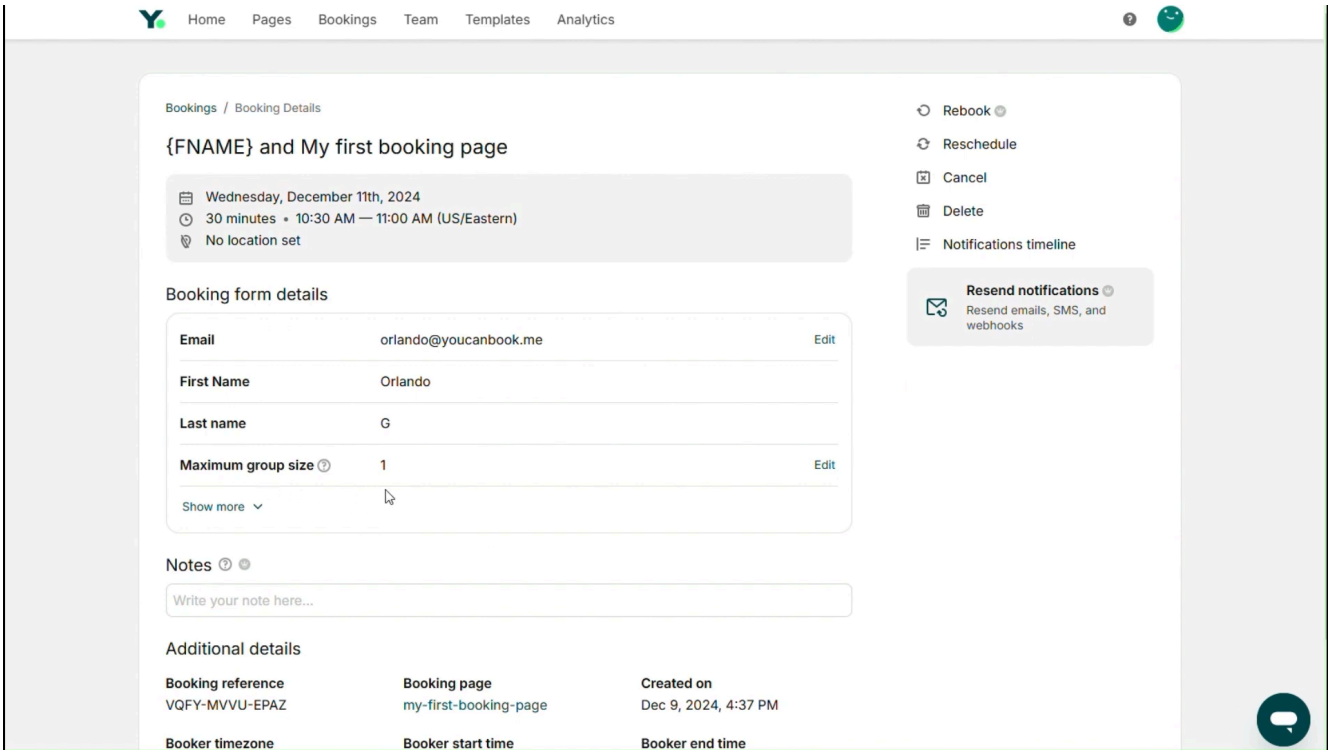


Рис. 1.7 Налаштування сторінки бронювання в YouCanBook.me

YouCanBook.me вирізняється простотою та швидкістю налаштування. Платформа пропонує безкоштовну версію з базовим функціоналом і дозволяє створювати кілька сторінок бронювання одночасно.

Однак сервіс має ряд обмежень. Інтерфейс виглядає застарілим, можливості налаштування доступності досить скромні, а інтеграція з іншими сервісами слабка. Крім того, система не підтримує автоматичне визначення зайнятих слотів з кількох календарів і не має вбудованої аналітики та звітності.

Google Calendar з функцією Appointment Slots це вбудована можливість Google Calendar, що дозволяє створювати часові слоти для бронювання безпосередньо в календарі.

Функція досить проста, користувач створює слоти певної тривалості і система генерує посилання для бронювання (рис. 1.8).

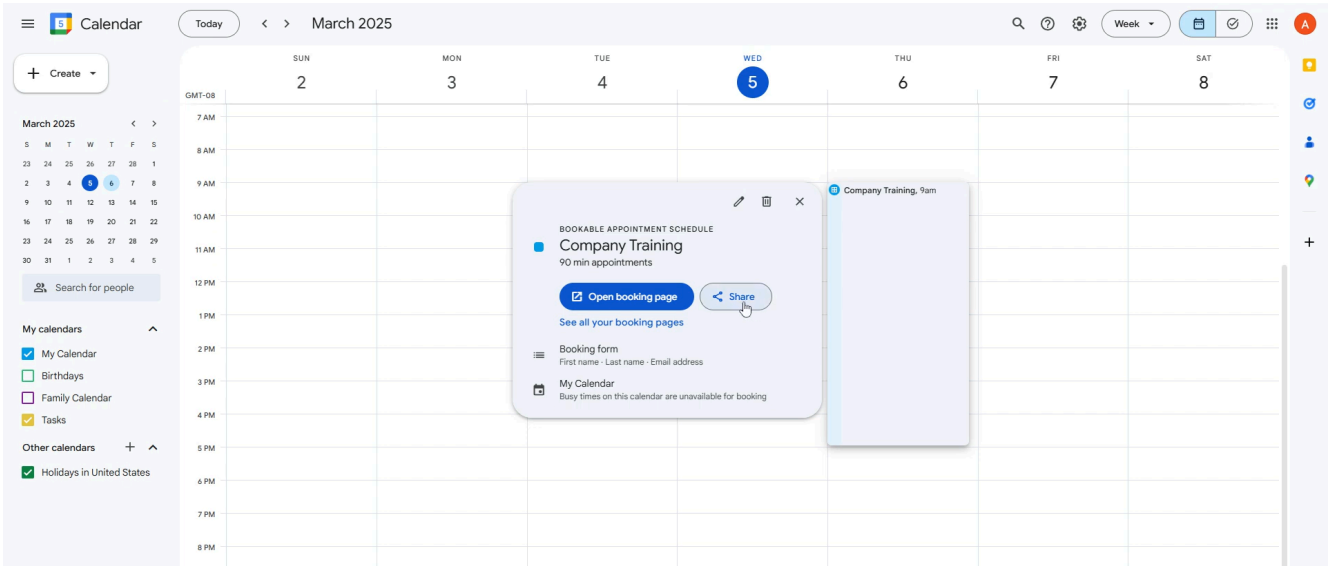


Рис. 1.8 Створення Appointment Slots у Google Calendar

Google Calendar з функцією Appointment Slots є повністю безкоштовним рішенням і має повну інтеграцію з основним календарем Google, що робить його простим і звичним для більшості користувачів.

Однак функціональність цього інструменту досить обмежена. Він не підтримує персоналізовані посилання для бронювання, не дозволяє створювати різні типи подій, не забезпечує автоматичного визначення та блокування зайнятих слотів, а також не має додаткових налаштувань, таких як буфери між зустрічами чи питання до учасників.

Проведений аналіз існуючих систем онлайн-бронювання показав, що жодна з розглянутих платформ не задовольняє повністю потреби сучасних користувачів. Calendly, YouCanBook.me та Google Calendar з функцією Appointment Slots мають суттєві обмеження, які знижують їх практичну ефективність.

Найпоширенішими недоліками є відсутність автоматичного оновлення доступності та визначення зайнятих слотів з кількох календарів одночасно, обмежені можливості персоналізації посилань для бронювання, недостатня гнучкість налаштування доступності по днях тижня та годинах, а також висока вартість повноцінного функціоналу в платних тарифах. Крім того, більшість рішень не забезпечують глибокої інтеграції зі створенням онлайн-зустрічей та не дозволяють користувачам зручно керувати власним профілем через унікальне ім’я.

Виявлені проблеми підтверджують доцільність розробки власної цифрової платформи BKalendar, яка усуне зазначені недоліки і запропонує сучасне, більш гнучке та зручне рішення для онлайн-бронювання зустрічей.

2. АНАЛІЗ АРХІТЕКТУРИ ТА АЛГОРИТМІВ СИСТЕМИ

2.1 Вимоги до системи

BKalendar є сучасною цифровою платформою для онлайн-бронювання зустрічей, розроблена на базі фреймворку Next.js 16 з інтеграцією платформи Nylas. Система дозволяє користувачам створювати типи подій, гнучко налаштовувати доступність по днях тижня, автоматично визначати зайняті слоти та генерувати унікальні посилання для бронювання. Основними функціями застосунку є інтерактивний календар з блокуванням зайнятих слотів, автоматичне створення онлайн-зустрічей, система профілю з власним іменем користувача (username) та зручний перегляд заброньованих зустрічей. Очікується, що система покаже високу зручність користування, надійність та швидкодію при роботі.

2.1.1 Функціональні вимоги

Функціональні вимоги визначають, які дії система повинна виконувати. Основні функціональні можливості платформи BKalendar охоплюють авторизацію та управління профілем користувача, створення та налаштування типів подій, автоматичне визначення зайнятих слотів, генерацію персональних посилань, інтерактивний календар бронювання, роботу із забронюваними зустрічами та оповіщення.

Користувачі можуть зареєструватися та авторизуватися в системі за допомогою Google або Microsoft акаунту. Після авторизації доступна можливість встановити унікальне ім’я користувача (username), яке використовується для формування персонального посилання на бронювання.

Система дозволяє створювати різні типи подій, вказуючи для кожного назву, опис та тривалість. Для кожного типу події користувач може гнучко налаштовувати доступність по днях тижня (від понеділка до неділі) та встановлювати індивідуальні робочі години для кожного дня.

Важливою функцією є автоматичне визначення зайнятих слотів. Платформа інтегрується з Google Calendar користувача через сервіс Nylas, отримує актуальну інформацію про зайнятість і автоматично блокує часові слоти, які вже зайняті в календарі.

Генерація персональних посилань для бронювання, інтерактивний календар з вибором дати та часу, автоматичне блокування недоступних слотів, збереження та перегляд історії заброньованих зустрічей, а також автоматичні оповіщення (створення Email для гостя та додавання події до календаря організатора) забезпечують повний цикл роботи з бронюванням від початку до завершення.

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якості системи. Нефункціональні вимоги системи розписано у таблиці 2.1.

Таблиця 2.1

Нефункціональні вимоги системи

№	Вимога	Опис
1	Зручність використання	Інтерфейс має бути інтуїтивно зрозумілим та сучасним
2	Швидкодія	Час завантаження сторінки не повинен перевищувати 3-4 секунди
3	Надійність	Система повинна стабільно працювати
4	Безпека	Захист даних користувачів та дотримання вимог GDPR
5	Адаптивність	Коректне відображення на сучасних пристроях та браузерах
6	Масштабованість	Можливість розширення функціоналу без значних змін архітектури

2.1.3 Вимоги до безпеки

Система BKalendar має відповідати сучасним вимогам інформаційної безпеки. Зокрема, передбачено захист від CSRF-атак, жорстке обмеження доступу до даних користувача (лише авторизовані учасники можуть переглядати власні події та бронювання), а також обов’язкове використання захищеного протоколу HTTPS для всіх мережевих з’єднань.

2.2 Обґрунтування вибору технологій та інструментів

Для реалізації BKalendar було обрано сучасний набір технологій, який забезпечує високу продуктивність, зручність розробки та масштабованість системи. Далі наведено обґрунтування вибору основних технологій та інструментів.

2.2.1 Мова програмування та фреймворк

Для розробки як клієнтської, так і серверної частини системи було обрано мову TypeScript [23] у поєднанні з фреймворком Next.js 16 [16].

TypeScript було обрано замість JavaScript через сувору типізацію, що значно зменшує кількість помилок на етапі розробки та спрощує підтримку коду в майбутньому.

Next.js 16 було обрано як основний фреймворк завдяки вбудованій підтримці Server Components та Server Actions, що дозволяє значно зменшити обсяг коду. Фреймворк забезпечує високу продуктивність за рахунок автоматичної оптимізації та ефективного кешування, зручну реалізацію серверної логіки через API Routes, повну підтримку TypeScript, а також великий вибір плагінів, бібліотек і активну спільноту розробників.

Альтернативи були відхилені через більшу складність налаштування та меншу продуктивність порівняно з Next.js.

2.2.2 База даних

Для зберігання даних було обрано MongoDB [14] у поєднанні з ODM Mongoose [15]. MongoDB ідеально підходить для цього проекту завдяки гнучкій схемі даних, яка дозволяє зручно зберігати типи подій з різними налаштуваннями доступності по днях тижня. Крім того, база забезпечує високу швидкість роботи з документами, просту інтеграцію з Next.js через Mongoose, а також легку масштабованість системи в майбутньому.

Mongoose було обрано як ODM, оскільки він забезпечує зручну роботу з моделями, валідацією даних та кодом.

2.2.3 Інтеграція з календарними сервісами

Для інтеграції з Google Calendar та іншими сервісами було обрано платформу Nylas [17]. Nylas забезпечує потужний API для роботи з календарями, що дозволяє отримувати інформацію про зайняті слоти та створювати нові події. Крім того, платформа підтримує автоматичне створення онлайн-зустрічей у Google Meet, пропонує зручну авторизацію користувачів через OAuth та демонструє високу надійність і швидкість роботи.

Альтернативи, такі як пряма робота з Google Calendar API, були відхилені через складність реалізації та необхідність підтримувати декілька різних API одночасно.

2.2.4 Додаткові бібліотеки та інструменти

У проєкті також використовується низка додаткових бібліотек та інструментів. Для роботи з датами та часом (вибір доступних слотів, форматування) застосовується бібліотека date-fns [6]. Стилізація інтерфейсу реалізована за допомогою Tailwind CSS [22], що забезпечує швидку та сучасну стилізацію. Іконки додано через бібліотеку Lucide-react [12]. HTTP-запити виконуються вбудованим у Next.js fetch API, а для роботи з формами та їх валідації використовується комбінація React Hook Form та Zod.

2.2.5 Середовище розробки

Як основне середовище розробки обрано Visual Studio Code. Для хмарного хостингу бази даних використовується MongoDB Atlas.

Обрані технології (Next.js 16 + TypeScript + MongoDB + Nylas) забезпечують оптимальну продуктивность, зручність розробки та можливість для реалізації всіх функціональних вимог системи. Використання цих технологій дозволяє створити швидку, надійну та масштабовану цифрову платформу.

2.3 Загальна архітектура системи

Розроблений веб-сервіс BKalendar побудовано за принципом трирівневої архітектури (Three-Tier Architecture), що забезпечує чіткий розподіл відповідальності між компонентами та спрощує підтримку й масштабування системи.

Система складається з трьох основних рівнів:

1. Рівень представлення (Presentation Layer).
2. Рівень бізнес-логіки (Business Logic Layer).
3. Рівень доступу до даних (Data Access Layer).

Рівень представлення (Presentation Layer) відповідає за безпосередню взаємодію з користувачем і повністю реалізований на базі фреймворку Next.js 16 з використанням App Router.

До нього належать Server Components, які забезпечують рендеринг більшості сторінок на сервері (наприклад, dashboard/page.tsx, event-types/page.tsx, profile/page.tsx), Client Components для інтерактивних елементів (TimePicker.tsx, EventTypeForm.tsx, BookingForm.tsx), а також багаторазово використовуванні UI-компоненти (DashboardNav.tsx, Header.tsx, Hero.tsx), що розташовані в папці src/components/.

Таким чином, рівень представлення забезпечує зручну та сучасну взаємодію користувача з системою завдяки поєднанню серверного рендерингу сторінок, клієнтських компонентів для динамічних елементів та перевикористовуваних

UI-компонентів. Це дозволяє досягти високої продуктивності та комфортного користування на всіх пристроях.

Рівень бізнес-логіки (Business Logic Layer) є ключовим компонентом системи, оскільки саме тут відбувається обробка всіх запитів, валідація даних та реалізація основної функціональності застосунку. Основні модулі бізнес-логіки перелічені та описані в таблиці 2.2. Кожен модуль виконує чітко визначену задачу – від авторизації користувачів до створення онлайн-зустрічей через платформу Nylas.

Таблиця 2.2

Основні модулі рівня бізнес-логіки

№	Модуль (API Route)	Опис
1	api/auth	Авторизація та реєстрація користувачів
2	api/profile/	Робота з профілем користувача
3	api/event-types/	Створення та редагування типів подій
4	api/bookings/	Створення та перегляд заброньованих зустрічей
5	api/busy/	Отримання зайнятих слотів через Nylas
6	api/oauth/exchange/	Обмін коду авторизації на токен Nylas

Рівень бізнес-логіки реалізує всю основну функціональність системи через API Routes, забезпечуючи авторизацію користувачів, управління профілем, роботу з типами подій, створення бронювань та інтеграцію з платформою Nylas. Що дозволяє чітко розділити логіку обробки запитів від рівня представлення та рівня доступу до даних.

Рівень доступу (Data Access Layer) до даних відповідає за зберігання, отримання та обробку інформації в системі. Він реалізований за допомогою бази даних MongoDB [14] та ODM Mongoose [15], що забезпечує зручну роботу з документами та гнучку схему даних. (Див. рис. 2.1)

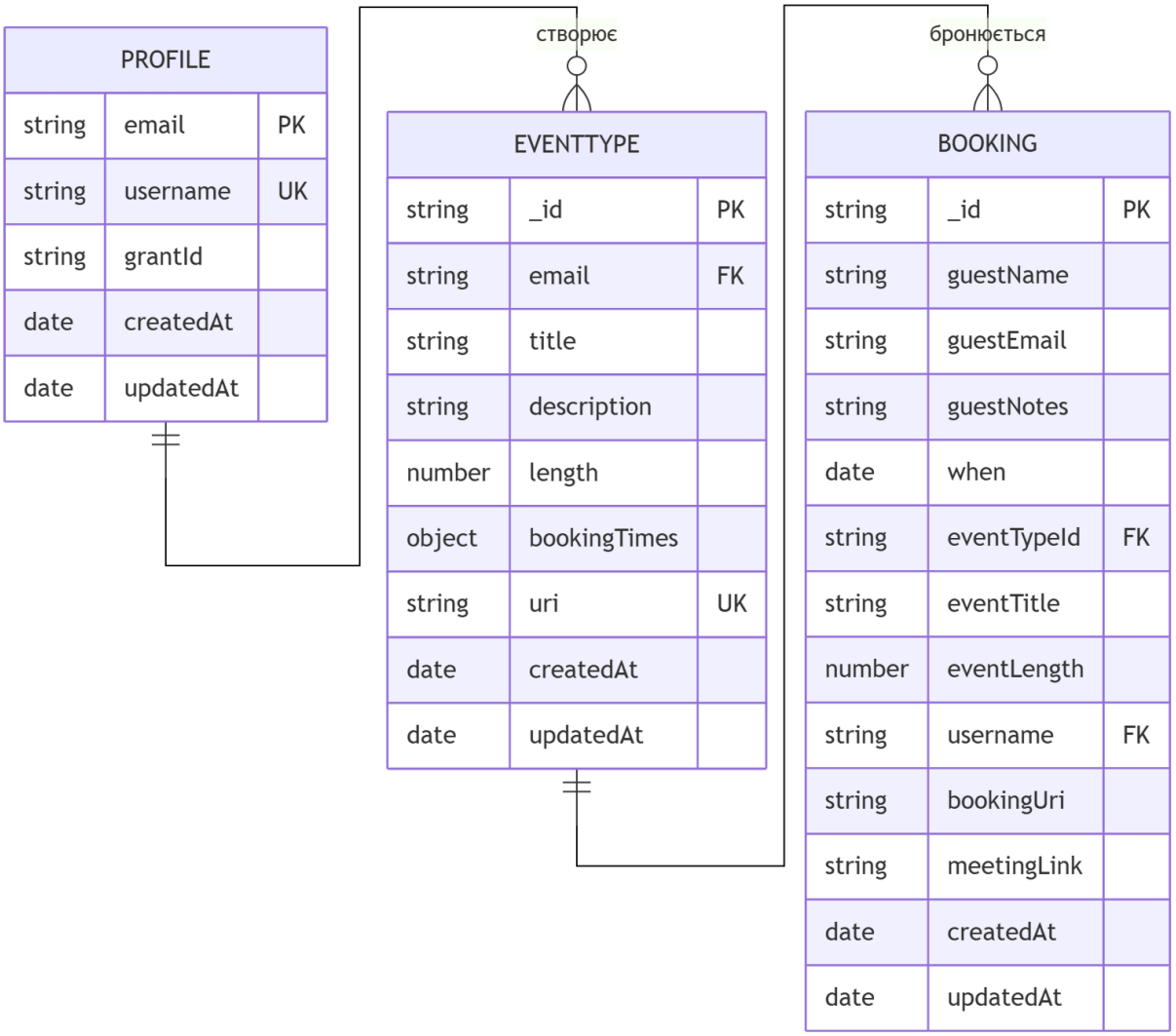


Рис. 2.1 Архітектура бази даних Mongo DB

Архітектура бази даних побудована на основі трьох основних колекцій, які відповідають ключовим елементам системи. Колекція EventType зберігає типи подій користувача разом з налаштуваннями доступності по днях тижня. Колекція Profile містить профілі користувачів, включаючи grantId від платформи Nylas та унікальне ім’я користувача (username). Колекція Booking призначена для зберігання інформації про заброньовані зустрічі, зокрема посилання на створені події в Google Meet.

Модель EventType призначена для зберігання типів подій, які створює користувач. Вона включає основні поля: email власника, назву події (title), опис

(description), тривалість у хвилинах (length) та унікальний ідентифікатор посилання (uri).

Особливістю моделі є вкладена схема bookingTimes, яка дозволяє зберігати гнучкі налаштування доступності для кожного дня тижня окремо (від понеділка до неділі). Для кожного дня вказуються час початку (from) та закінчення (to). Схема також автоматично додає поля createdAt та updatedAt завдяки опції timestamps.

Повний код моделі EventType наведено в Додатку Б, 1.1.

Модель Profile призначена для зберігання персональних даних користувача. Вона містить три основні поля: email (унікальний ідентифікатор користувача), username (унікальне ім’я користувача, яке використовується для формування персональних посилань на бронювання) та grantId (токен авторизації Nylas, необхідний для роботи з календарем користувача).

Схема забезпечує унікальність полів email та username, що унеможливлює створення дублюючих записів. Як і в інших моделях, автоматично додаються поля createdAt та updatedAt завдяки опції timestamps.

Повний код моделі Profile наведено в Додатку Б, 1.2.

Модель Booking призначена для зберігання інформації про кожну заброньовану зустріч. Вона містить дані про гостя (ім’я, email та додаткові нотатки), дату і час проведення зустрічі (when), а також пов’язує бронювання з конкретним типом події через поле eventTypeId (посилання на модель EventType).

Крім того, у моделі зберігаються назва події (eventTitle), її тривалість (eventLength), username організатора, uri бронювання та meetingLink – посилання на автоматично створену зустріч у Google Meet. Схема забезпечує обов’язковість ключових полів і автоматично додає поля createdAt та updatedAt завдяки опції timestamps.

Повний код моделі Booking наведено в Додатку Б, 1.3.

Рівень доступу до даних забезпечує зручну та ефективну взаємодію з базою даних MongoDB [14] через ODM Mongoose [15], що дозволяє гнучко зберігати та обробляти дані про типи подій, профілі користувачів та заброньовані зустрічі. Використання вкладених схем (наприклад, BookingSchema у моделі EventType)

забезпечує логічну структуру даних та спрощує роботу з доступністю по днях тижня.

Система також використовує платформу Nylas [17] (файл src/libs/nylas.ts) як основний зовнішній сервіс. Nylas забезпечує отримання інформації про зайняті слоти з Google Calendar користувача, автоматичне створення онлайн-зустрічей у Google Meet, а також авторизацію користувачів через протокол OAuth.

Схематично, архітектуру системи додатка відображено на рис. 2.2.

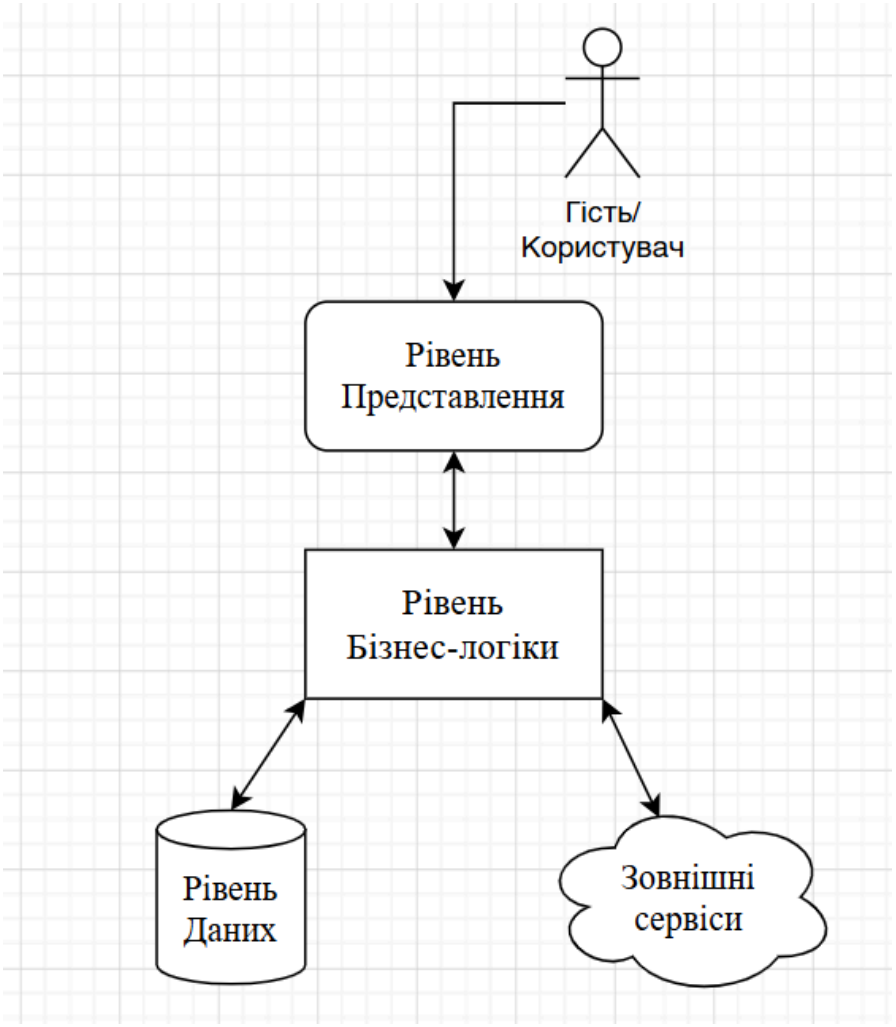


Рис. 2.2 Схема архітектури системи

З рисунку видно, що архітектура системи BKalendar побудована за принципом трирівневої архітектури з чітким розподілом відповідальності між компонентами.

Користувач взаємодіє з “фронтендом”, реалізованим на базі Next.js 16 App Router. “Фронтенд” складається з Server Components для рендерингу більшості сторінок на сервері та Client Components (TimePicker, EventTypeForm, BookingForm) для інтерактивних елементів. HTTP-запити від “фронтенду” надходять до рівня бізнес-логіки, який реалізований через API Routes у папці src/app/api/.

Рівень бізнес-логіки обробляє запити, виконує валідацію даних та взаємодіє з платформою Nylas для отримання зайнятих слотів з Google Calendar користувача та автоматичного створення онлайн-зустрічей у Google Meet. Окрім цього, рівень бізнес-логіки робить запити до бази даних MongoDB через ODM Mongoose для зберігання та отримання даних про типи подій, профілі користувачів та заброньовані зустрічі.

Зовнішній сервіс Nylas забезпечує синхронізацію з Google Calendar та Google Meet, дозволяючи автоматично створювати онлайн-зустрічі та отримувати інформацію про зайнятість користувача в реальному часі.

2.4 Основні алгоритми системи

Під час розробки веб-сервісу BKalendar було реалізовано низку важливих алгоритмів, що забезпечують основну функціональність системи. У цьому підрозділі детально описується найбільш важливі алгоритми.

2.4.1 Алгоритм створення та збереження типу події

Алгоритм створення нового типу події реалізовано у компоненті EventTypeForm.tsx та обробляється на сервері через маршрут POST /api/event-types.

Під час завантаження компонента формується об’єкт formData зі стандартними значеннями доступності: для робочих днів (понеділок–п’ятниця) встановлюється інтервал 09:00–17:00, для вихідних (субота–неділя) – 10:00–14:00, при цьому всі дні тижня спочатку активні. Користувач може вмикати або вимикати

окремі дні тижня за допомогою функції toggleDay, а також змінювати час початку та закінчення для кожного дня за допомогою функції updateTime.

Перед збереженням система автоматично генерує унікальне посилання (URI / slug) на основі назви події. Для цього рядок назви переводиться в нижній регістр, обрізаються пробіли, замінюються всі нетипові символи на дефіс, а зайві дефіси видаляються. Якщо назва порожня, використовується значення «event».

Після цього формується об’єкт payload, який містить лише активні дні (activeBookingTimes). Дані надсилаються на сервер методом POST (при створенні нового типу події) або PUT (при редагуванні). На сервері отримані дані зберігаються в колекції EventType за допомогою моделі EventTypeModel, яка використовує вкладену схему BookingSchema для зберігання налаштувань доступності по днях тижня.

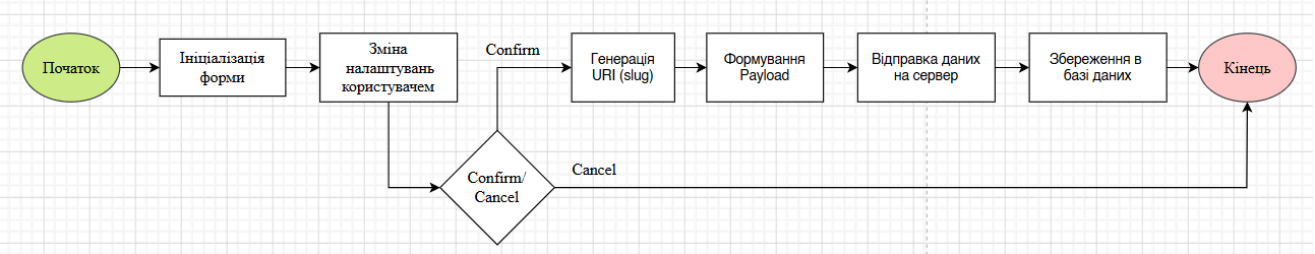


Рис. 2.3 Блок-Схема алгоритму створення нового типу події

Блок-схема алгоритму створення нового типу події (рис. 2.3) демонструє послідовність дій від ініціалізації форми до збереження даних у базі. Основними етапами є: формування початкових налаштувань доступності, зміна параметрів користувачем через функції toggleDay та updateTime, автоматична генерація унікального URI, формування та відправка даних на сервер методом POST або PUT, а також збереження типу події в колекції EventType за допомогою моделі EventTypeModel.

2.4.2 Алгоритм редагування типу події

Алгоритм редагування реалізовано на сторінці edit/[id]/page.tsx та у компоненті EventTypeForm.tsx.

Основні етапи:

1. Завантаження існуючих даних. При відкритті сторінки редагування сервер отримує id з параметрів маршруту, підключається до бази даних (MongoDB) та завантажує документ за допомогою EventTypeModel.findOne({ _id: id, email }).
2. Передача даних у форму. Отриманий документ перетворюється в простий об’єкт (JSON.parse(JSON.stringify(...))) і передається у компонент EventTypeForm через doc.
3. Ініціалізація форми з існуючими значеннями. У useState перевіряється наявність doc?.bookingTimes. Якщо дані існують - форма заповнюється ними, якщо ні, то використовуються стандартні значення.
4. Оновлення даних. При натисканні кнопки «Update» формується payload і надсилається методом PUT на /api/event-types/${doc._id}.
5. Обробка на сервері. У файлі event-types/[id]/route.ts реалізовано обробник PUT, який оновлює документ за допомогою findByIdAndUpdate.

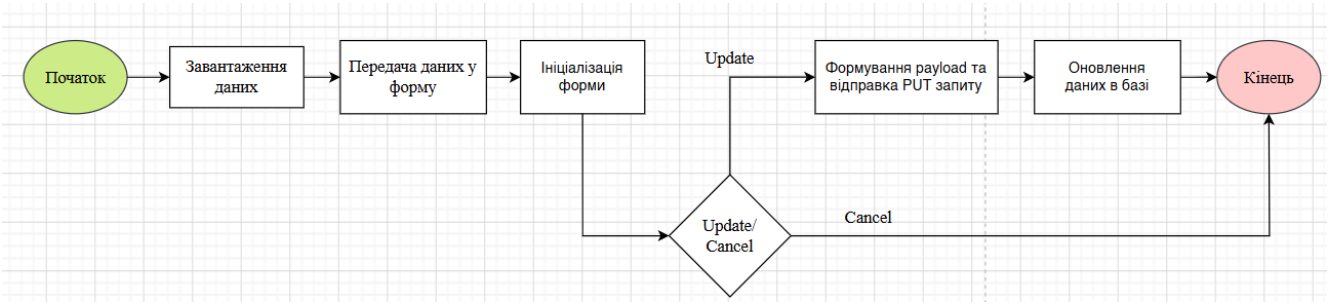


Рис. 2.4 Блок-Схема алгоритму редагування типу події

Блок-схема алгоритму редагування типу події (рис. 2.4) ілюструє послідовність дій від завантаження існуючих даних до оновлення документа в базі. Схема має п’ять основних етапів: завантаження поточного типу події з бази даних, передача даних у форму, ініціалізація форми з існуючими значеннями, формування та відправка оновлених даних методом PUT, а також обробка запиту на сервері за допомогою методу findByIdAndUpdate.

2.4.3 Алгоритм визначення доступних часових слотів

Основний алгоритм реалізовано у компоненті TimePicker.tsx.

Етапи роботи:

1. Вибір дати. Користувач обирає дату у календарі. Система визначає день тижня через функцію getWeekdayName.
2. Отримання налаштувань доступності. З об’єкта bookingTimes витягується час from та to для вибраного дня.
3. Генерація слотів. Функція generateTimeSlots(from, to, duration) генерує масив можливих часів з урахуванням тривалості події (length). Слоти створюються з кроком в 30 хвилин.
4. Відображення слотів. Згенеровані слоти відображаються у вигляді кнопок. Кожен слот перевіряється функцією isSlotBusy.

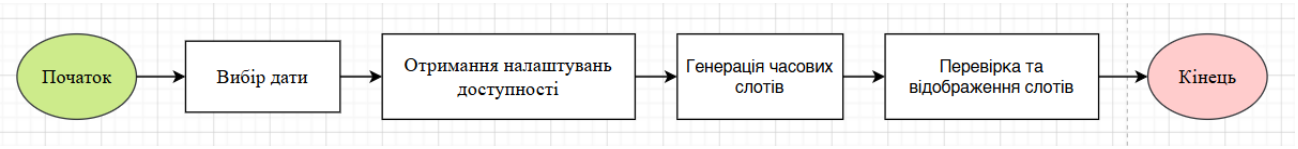


Рис. 2.5 Блок-Схема алгоритму визначення доступних часових слотів

Блок-схема алгоритму визначення доступних часових слотів (рис. 2.5) ілюструє послідовність дій від вибору дати користувачем до відображення доступних часових слотів. Схема має чотири основні етапи: вибір дати у календарі та визначення дня тижня через функцію getWeekdayName, отримання налаштувань доступності для обраного дня, генерація часових слотів за допомогою функції generateTimeSlots, а також перевірка зайнятості кожного слота функцією isSlotBusy та їх відображення у вигляді кнопок.

2.4.4 Алгоритм перевірки зайнятості слотів

Алгоритм реалізовано у TimePicker.tsx та серверному файлі busy/route.ts.

Етапи:

1. Запит на сервер. При виборі дати компонент надсилає GET запит на /api/busy з параметрами username, from та to.
2. Отримання зайнятих слотів на сервері. У файлі busy/route.ts виконується пошук усіх бронювань користувача за вказаний період.
3. Формування масиву зайнятих інтервалів. Кожен документ бронювання перетворюється в об’єкт { startTime, endTime }.
4. Перевірка перетину. Функція isSlotBusy(time) перевіряє, чи перетинається згенерований слот з будь-яким зайнятим проміжком часу (з урахуванням тривалості події).

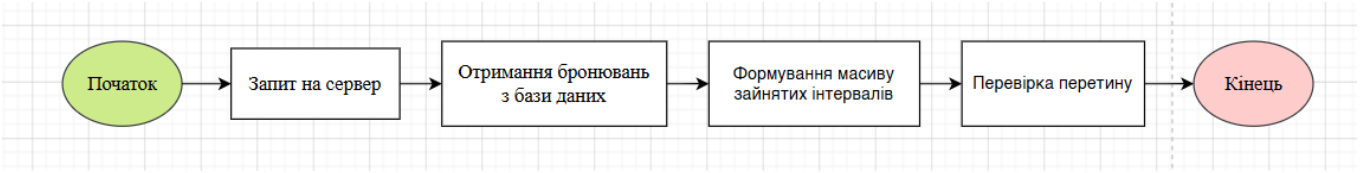


Рис. 2.6 Блок-Схема алгоритму перевірки зайнятості слотів

Блок-схема алгоритму перевірки зайнятості слотів (рис. 2.6) ілюструє послідовність дій від запиту клієнта до перевірки доступності часового слота. Алгоритм реалізовано у компоненті TimePicker.tsx та серверному файлі busy/route.ts. Основні етапи включають відправку GET-запиту на сервер з параметрами username, from та to, отримання всіх бронювань користувача за вказаний період з бази даних, формування масиву зайнятих інтервалів у форматі {startTime, endTime}, а також перевірку перетину згенерованого слота з будь-яким зайнятим проміжком часу за допомогою функції isSlotBusy.

2.4.5 Алгоритм бронювання зустрічі

Алгоритм реалізовано у файлі bookings/route.ts (метод POST).

Етапи:

1. Валідація вхідних даних. Перевіряється наявність обов’язкових полів (guestName, guestEmail, when, username, bookingUri).
2. Пошук профілю та типу події. Система знаходить профіль користувача (ProfileModel) та тип події (EventTypeModel).

3. Створення події в Google Calendar через Nylas. Виконується запит nylas.events.create з наступними параметрами: Назва та опис події, час початку та закінчення, учасники зустрічі.
4. Збереження бронювання. Створюється документ у колекції Booking з усіма необхідними даними, включно з meetingLink.
5. Повернення результату. Клієнту повертається success: true та meetingLink.

Рис. 2.7 Блок-Схема алгоритму бронювання зустрічі

Блок-схема алгоритму бронювання зустрічі (рис. 2.7) ілюструє послідовність дій від отримання запиту до створення події в Google Meet. Алгоритм реалізовано у файлі bookings/route.ts. Основні етапи включають валідацію вхідних даних, пошук профілю користувача та типу події в базі даних, створення події через Nylas з автоматичним додаванням Google Meet, збереження інформації про бронювання в колекції Booking, а також повернення клієнту результату з посиланням на зустріч.

Реалізовані алгоритми забезпечують повноцінну роботу системи онлайн-бронювання. Використання бібліотеки date-fns для роботи з датами, інтеграція з Nylas для синхронізації з календарем та автоматична генерація URI дозволяють досягти високої точності, надійності та зручності користування веб-сервісом BKalendar.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

3.1 Загальна структура системи

Веб-сервіс BKalendar реалізовано на основі фреймворку Next.js 16 [16] з використанням архітектури App Router. Проєкт має чітку та логічну структуру, що забезпечує зручність розробки та підтримки коду.

3.1.1 Структура файлів проєкту

Основна структура проєкту виглядає наступним чином:

```
src/
├── app/                        # Основна директорія Next.js App Router
│   ├── (booking)/              # Група маршрутів для сторінок бронювання
│   │   └── [username]/[booking-uri]/
│   │       ├── [booking-time]/   # Сторінка бронювання конкретного часу
│   │       └── page.tsx          # Сторінка вибору часу
│   ├── (site)/                 # Група публічних та захищених сторінок
│   │   ├── about/
│   │   ├── dashboard/
│   │   │   ├── booked-events/
│   │   │   ├── event-types/
│   │   │   │   ├── edit/[id]/
│   │   │   │   └── new/
│   │   │   └── page.tsx
│   │   ├── features/
│   │   ├── pricing/
│   │   └── layout.tsx
│   ├── api/                    # API Routes
│   │   ├── auth/
│   │   ├── bookings/
│   │   ├── busy/
│   │   ├── event-types/
│   │   │   └── [id]/
│   │   ├── logout/
│   │   ├── oauth/
│   │   │   └── exchange/
│   │   └── profile/
│   ├── layout.tsx
│   └── page.tsx
├── components/                 # Перевикористовувані компоненти
│   ├── DashboardNav.tsx
│   ├── EventTypeForm.tsx
│   ├── EventTypeDelete.tsx
│   ├── Header.tsx
│   ├── Hero.tsx
│   ├── ProfileForm.tsx
│   ├── RightNav.tsx
│   └── TimePicker.tsx
├── libs/                       # Бібліотеки та утиліти
│   ├── nylas.ts
│   ├── session.ts
│   ├── shared.ts
│   └── types.ts
├── models/                     # Mongoose моделі
│   ├── Booking.ts
│   ├── EventTypes.ts
│   └── Profile.ts
└── globals.css
```

Рис. 3.1 Основна структура проекту

3.1.2 Основні компоненти системи

Рівень представлення системи BKalendar включає декілька ключових клієнтських компонентів, які забезпечують основний користувацький інтерфейс.

Компонент TimePicker.tsx реалізує інтерактивний календар бронювання. Він дозволяє користувачеві або гостю обирати дату, автоматично визначає день тижня, генерує доступні часові слоти з урахуванням налаштувань доступності та перевіряє їх зайнятість через запит до сервера. Зайняті слоти візуально блокуються, що унеможливлює подвійне бронювання.

Компонент EventTypeForm.tsx призначений для створення та редагування типів подій. Він містить зручну форму для введення назви, опису та тривалості події, а також дозволяє користувачеві гнучко налаштовувати доступність по днях тижня: вмикати або вимикати окремі дні та встановлювати індивідуальні робочі години для кожного з них.

DashboardNav.tsx забезпечує навігацію в особистому кабінеті користувача. Компонент реалізує вкладки для швидкого переходу між основними розділами: «Event types» (список типів подій), «Booked events» (історія заброньованих зустрічей) та «Profile» (редагування профілю).

BookingForm.tsx використовується на публічній сторінці бронювання. Він містить форму для введення даних гостя (ім’я, email та додаткові нотатки), обробляє підтвердження бронювання та після успішного створення зустрічі відображає підтвердження з посиланням на Google Meet.

3.1.3 Основні API Routes

Серверна частина BKalendar повністю реалізована через API Routes фреймворку Next.js 16. Кожен маршрут виконує чітко визначену бізнес-операцію і забезпечує взаємодію між клієнтською частиною, базою даних та зовнішніми сервісами.

Маршрут POST /api/bookings є основним для створення бронювання. Він виконує повну валідацію вхідних даних, знаходить профіль користувача та тип події, створює подію в Google Calendar через платформу Nylas з автоматичним

додаванням посилання на онлайн-зустріч Google Meet, зберігає всі дані в колекції Booking і повертає клієнту результат із посиланням на зустріч.

GET /api/busy призначений для динамічного отримання інформації про зайняті часові слоти користувача за вказаний період. Він використовується компонентом TimePicker для автоматичного блокування вже зайнятих інтервалів при виборі дати та часу бронювання.

Маршрути POST /api/event-types, PUT /api/event-types/[id] та DELETE /api/event-types/[id] забезпечують повноцінні CRUD-операції з типами подій: створення нового типу події, його редагування та видалення.

Маршрути GET /api/profile та POST /api/profile відповідають за роботу з профілем користувача. Вони дозволяють отримувати поточне унікальне ім’я (username) та оновлювати дані профілю, зокрема встановлювати grantId від Nylas.

GET /api/oauth/exchange виконує обмін авторизаційного коду, отриманого від Google, на grantId (токен) платформи Nylas, який необхідний для подальшої роботи з календарем користувача.

3.1.4 Моделі бази даних

Система використовує три основні моделі бази даних для зберігання ключової інформації.

Модель EventType призначена для зберігання типів подій, створених користувачем, разом з детальними налаштуваннями доступності по днях тижня.

Модель Profile містить персональні дані користувача, зокрема унікальне ім’я (username), електронну пошту та grantId – токен авторизації від платформи Nylas, необхідний для доступу до календаря.

Модель Booking зберігає інформацію про кожне заброньоване бронювання, включаючи дані гостя (ім’я, email, нотатки), дату і час зустрічі, тип події, а також посилання на автоматично створену онлайн-зустріч у Google Meet.

3.2 Рівень представлення (інтерфейс користувача)

Рівень представлення системи BKalendar реалізований за допомогою фреймворку Next.js 16 з використанням React 19 та бібліотеки Tailwind CSS. Інтерфейс розроблено з ціллю зробити його зручним, сучасним та адаптивним.

Архітектура рівня представлення базується на поєднанні двох типів компонентів Next.js. Більшість сторінок генеруються на сервері за допомогою Server Components, що забезпечує швидке завантаження, кращий SEO та менший обсяг JavaScript, який передається клієнту. Інтерактивні елементи (календар вибору часу, форми, динамічні списки) реалізовані як Client Components. Крім того, у проекті активно використовуються багаторазово перевикористовувані UI-компоненти (DashboardNav.tsx, Header.tsx, Hero.tsx та інші), розташовані в папці src/components/. Це дозволяє підтримувати єдину дизайн-систему та значно спрощує подальшу розробку і підтримку інтерфейсу.

3.2.1 Головна сторінка та маркетингові розділи

Головна сторінка (app/(site)/page.tsx) містить Hero-блок з анімованим заголовком «Welcome to BKalendar» та закликом до дії. Нижче розміщено секцію з логотипами компаній, які довіряють сервісу. (Рис. 3.2)

Сторінка Features (app/(site)/features/page.tsx) описує ключові можливості системи: гнучке налаштування доступності, автоматичне визначення зайнятих слотів, інтеграцію з Google Meet та персоналізовані посилання. (Рис. 3.3)

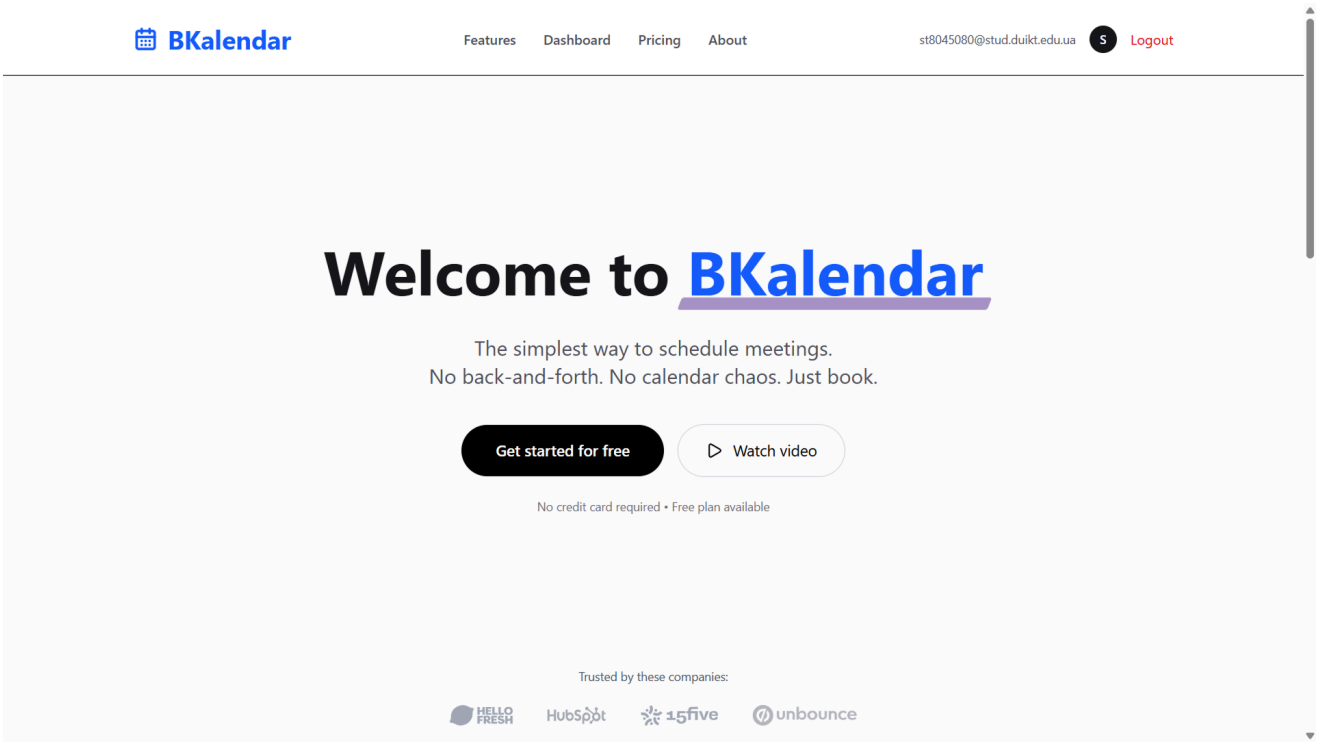


Рис. 3.2 Головна сторінка BKalendar

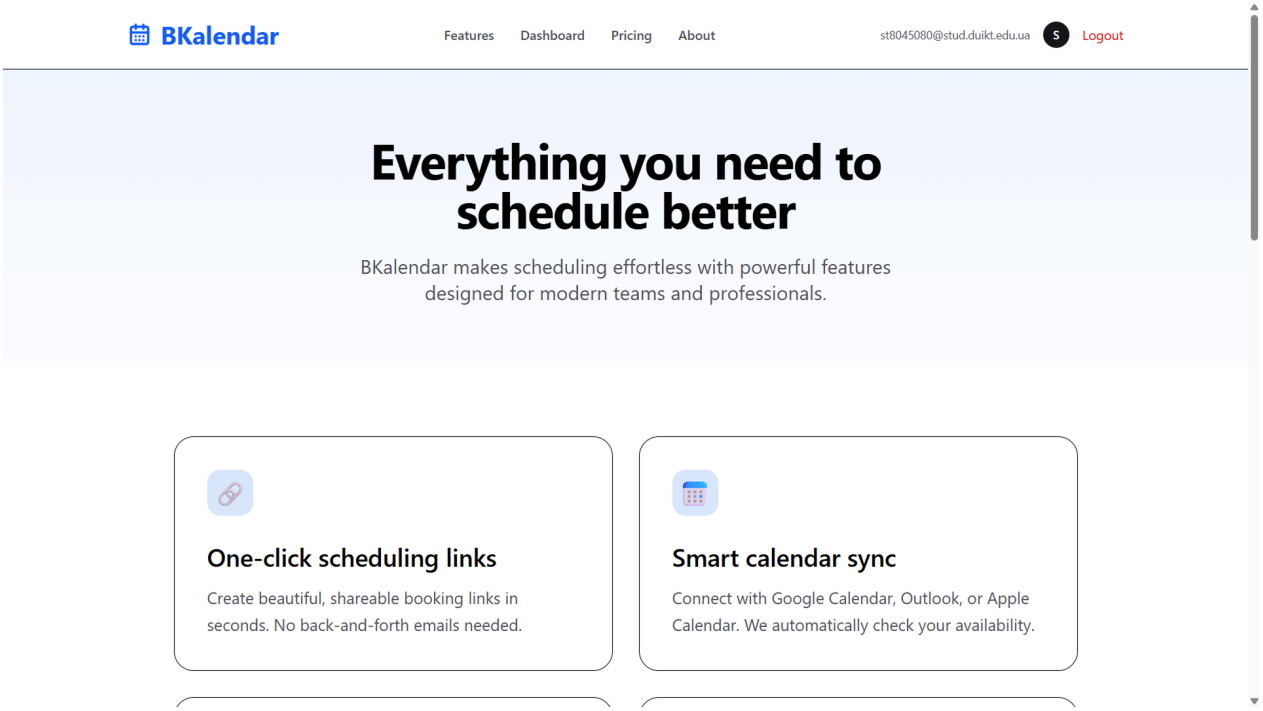


Рис. 3.3 Сторінка Features

Сторінка Pricing (app/(site)/pricing/page.tsx) пропонує три тарифні плани: Free, Pro та Team з описом можливостей кожного. (Рис. 3.4)

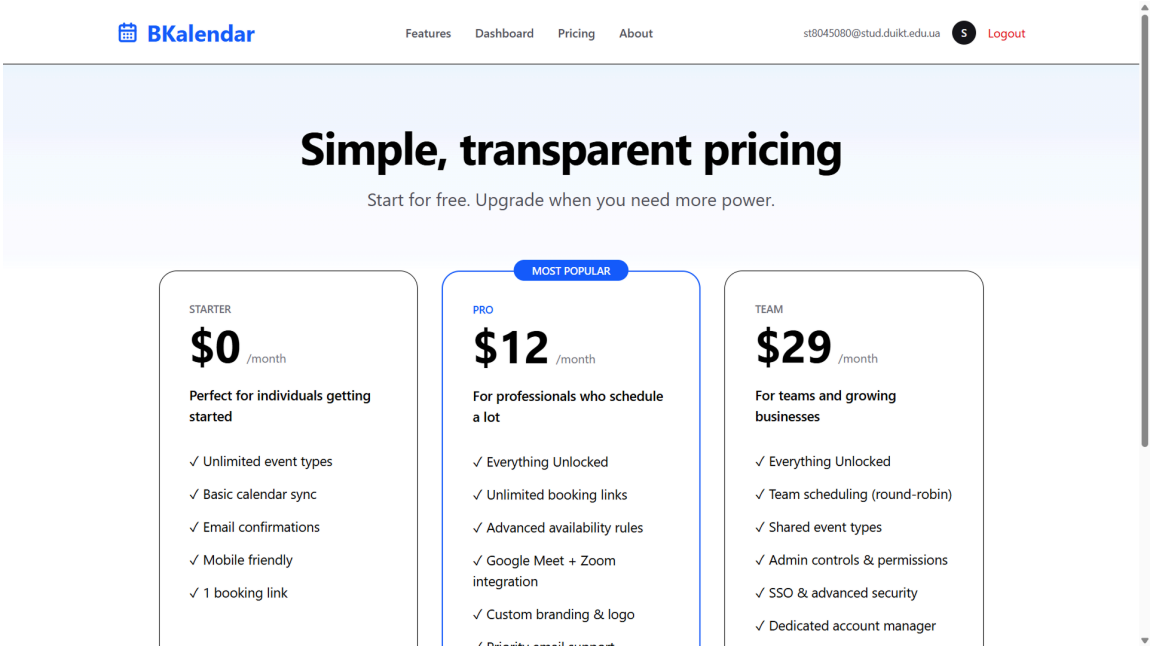


Рис. 3.4 Сторінка Pricing

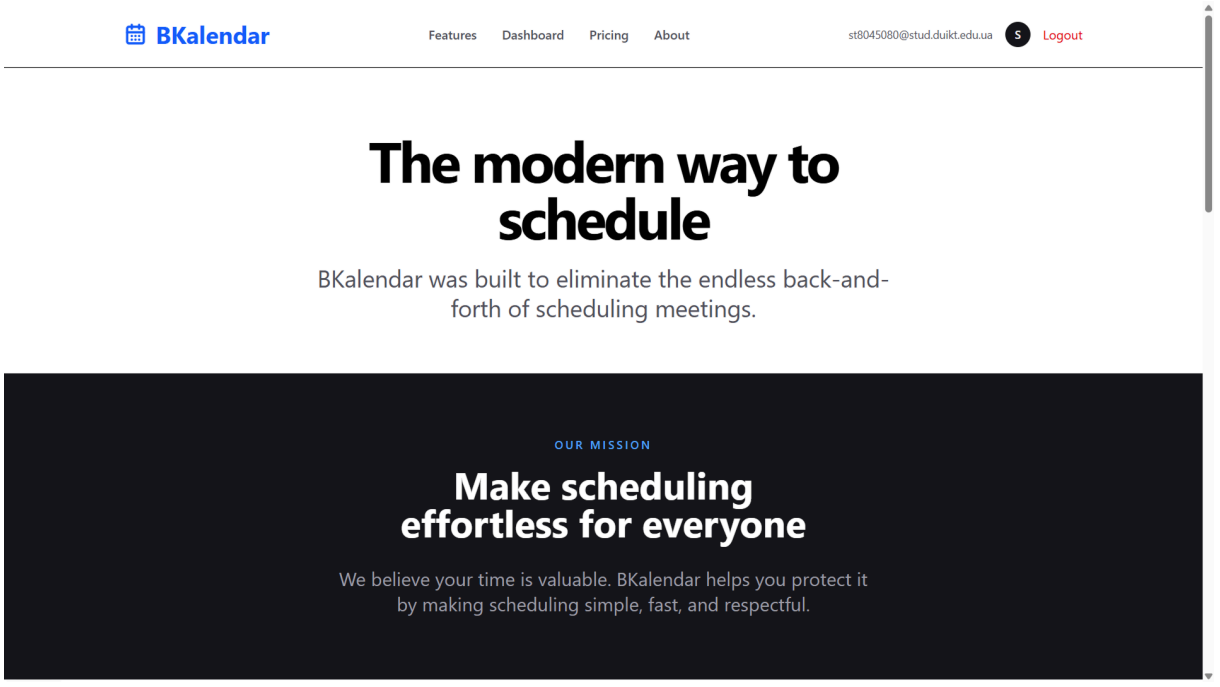


Рис. 3.5 Сторінка About

Сторінка About (app/(site)/about/page.tsx) містить різноманітну інформацію. Сторінка виконує інформаційну функцію та допомагає новим користувачам краще зрозуміти цінність сервісу (Рис. 3.5).

3.2.2 Особистий кабінет користувача (Dashboard)

Після авторизації користувач потрапляє до особистого кабінету (app/(site)/dashboard/page.tsx). Навігація здійснюється через компонент DashboardNav.tsx, який містить три вкладки: Booked events, Event types та Profile.

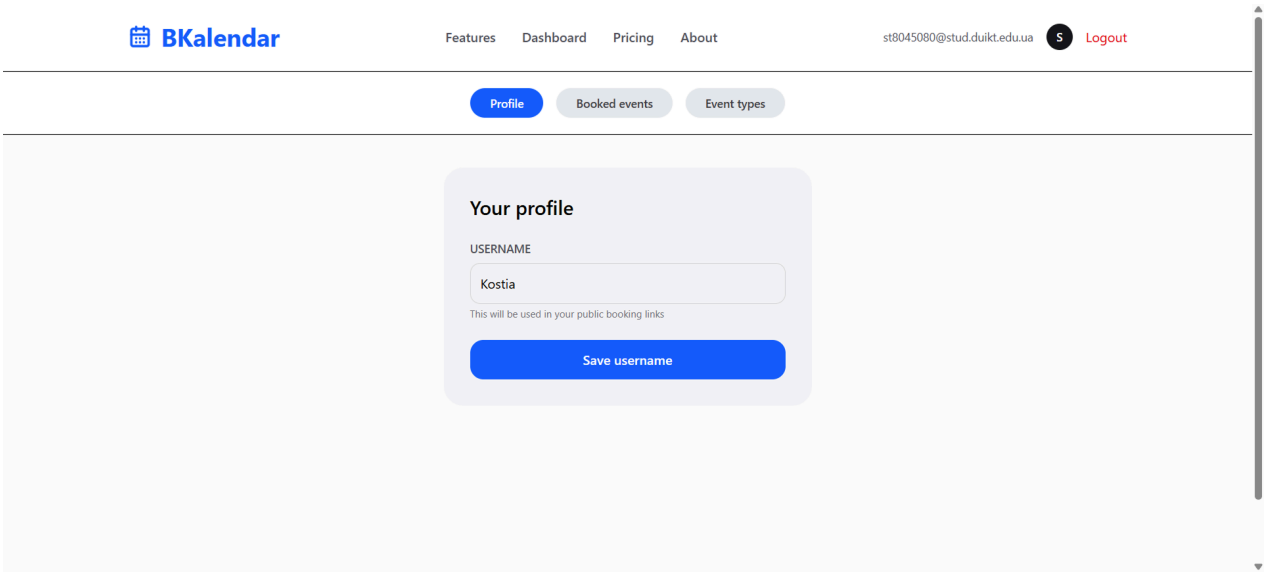


Рис. 3.6 Dashboard. Сторінка Profile

Сторінка Event types (app/(site)/dashboard/event-types/page.tsx) відображає список створених типів подій з можливістю редагування, видалення та копіювання посилання для бронювання (Рис. 3.7).

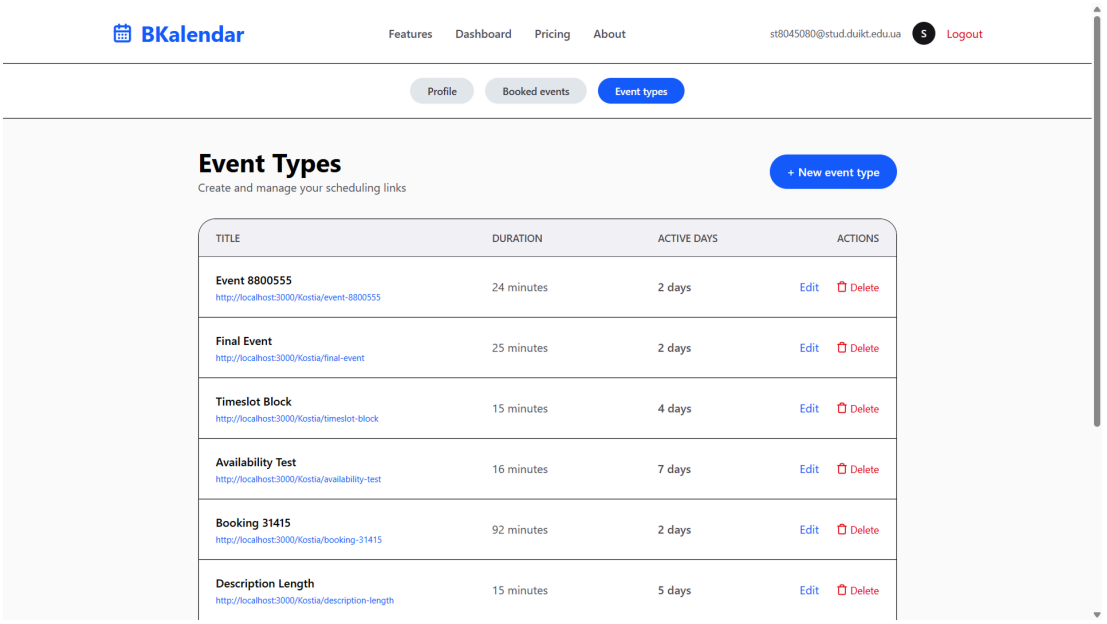


Рис. 3.7 Dashboard. Сторінка Event Types

Сторінка Booked events (app/(site)/dashboard/booked-events/page.tsx) показує всі заброньовані зустрічі з інформацією про гостя, дату, тривалість та посилання на Google Meet. (Рис. 3.8)

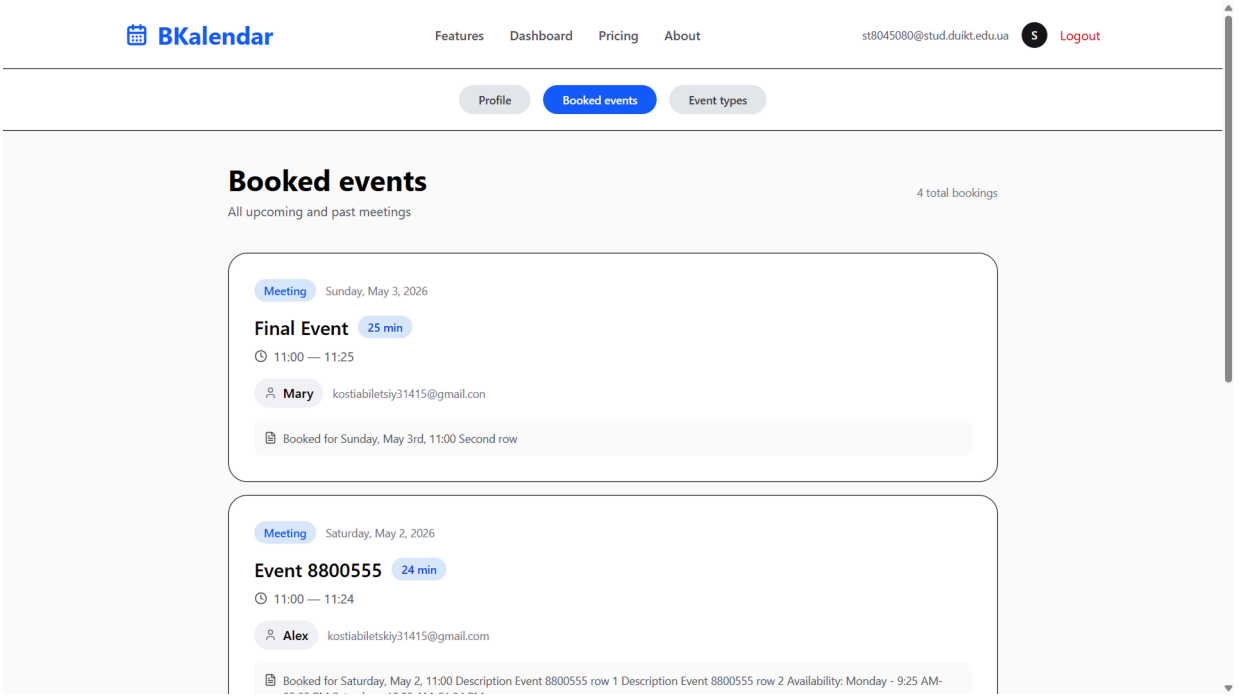


Рис. 3.8 Dashboard. Сторінка Booked Events

Сторінка Profile (app/(site)/dashboard/profile/page.tsx) надає користувачеві повноцінні можливості управління особистим профілем. Тут можна переглядати та редагувати персональні дані (ім’я, прізвище та електронну пошту), змінювати унікальне ім’я користувача (username), яке використовується для генерації персоналізованих посилань на бронювання, підключати або відключати інтеграцію з Google Calendar через платформу Nylas [17], а також переглядати статистику своїх бронювань. Форма редагування профілю реалізована у компоненті ProfileForm.tsx.

3.2.3 Створення та редагування типів подій

Форма створення та редагування типів подій реалізована у компоненті EventTypeForm.tsx (Рис. 3.9). Вона надає користувачеві повний набір інструментів для роботи з типами подій. Користувач може вказати назву події, її опис та тривалість у хвилинах, а також гнучко налаштувати доступність по днях тижня:

вмикати або вимикати окремі дні та встановлювати індивідуальні інтервали часу (from – to) для кожного активного дня.

Важливою особливістю форми є автоматична генерація унікального посилання (URI / slug) на основі введеної назви події. Це посилання використовується для формування персональної сторінки бронювання (наприклад, bkalendar.com/username/event-name).

Форма використовується як на сторінці створення нового типу події (/dashboard/event-types/new), так і на сторінці редагування існуючого типу (/dashboard/event-types/edit/[id]).

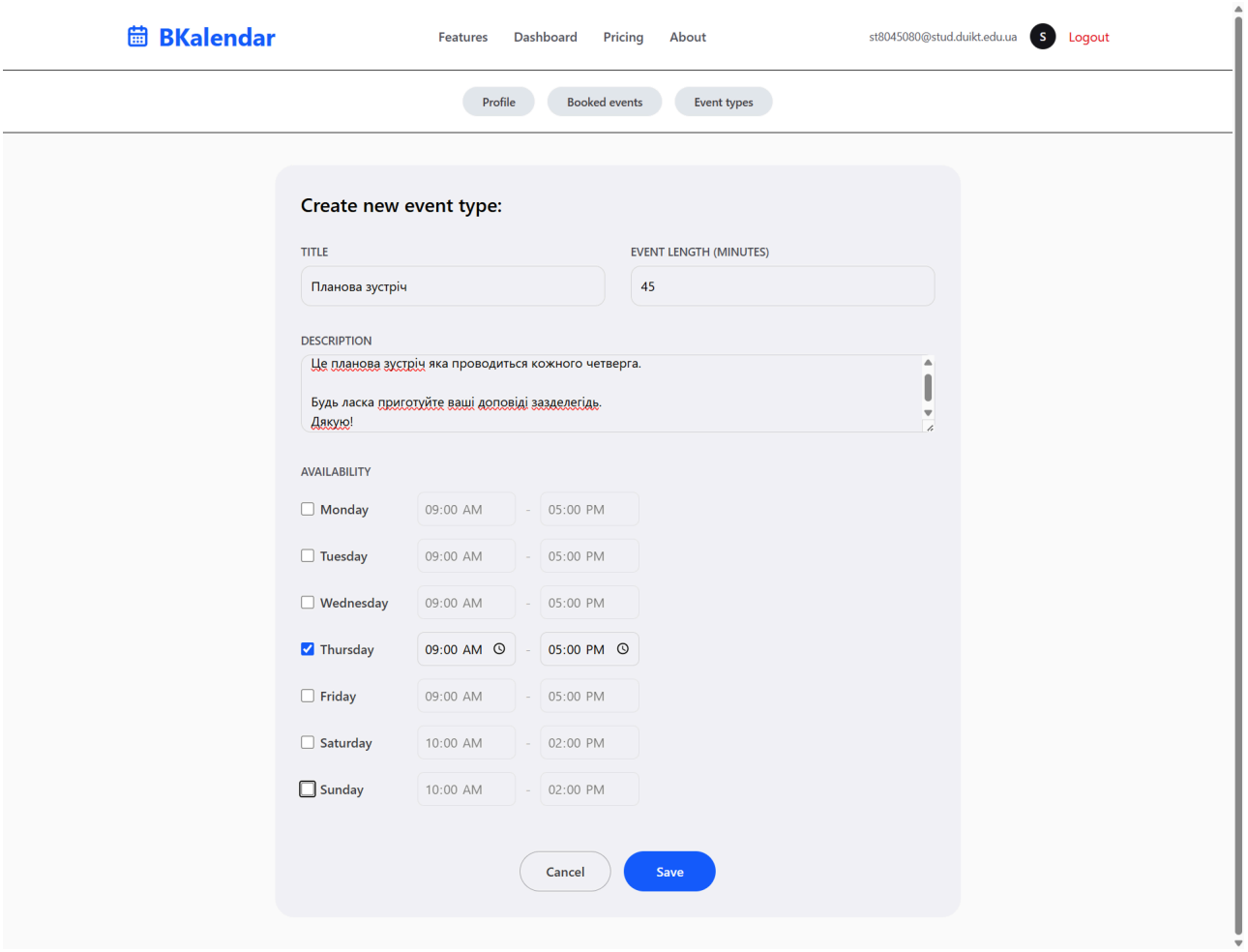


Рис. 3.9 Dashboard. Створення типу події з налаштування доступності

Форма створення та редагування типів подій надає користувачу повний контроль над налаштуванням доступності, тривалістю зустрічі та описом події.

Після збереження типу події система автоматично генерує унікальне посилання, за яким гості можуть записатися на зустріч. Це значно спрощує процес створення персоналізованих сторінок для бронювання.

3.2.4 Інтерактивний календар бронювання

Ключовим елементом системи є компонент TimePicker.tsx, який реалізує повноцінний інтерактивний календар бронювання (Рис. 3.10 та 3.11). Користувач або гість може вільно перемикати місяці за допомогою стрілок, обирати доступну дату в календарі, а також переглядати список доступних часових слотів на обраний день з урахуванням актуальної зайнятості.

Згенеровані слоти відображаються у вигляді кнопок. Система в реальному часі перевіряє кожний слот через функцію isSlotBusy і автоматично блокує зайняті інтервали. Зайняті слоти візуально позначаються сірим кольором з перекресленням тексту (Рис. 3.12), що унеможливлює випадкове подвійне бронювання.

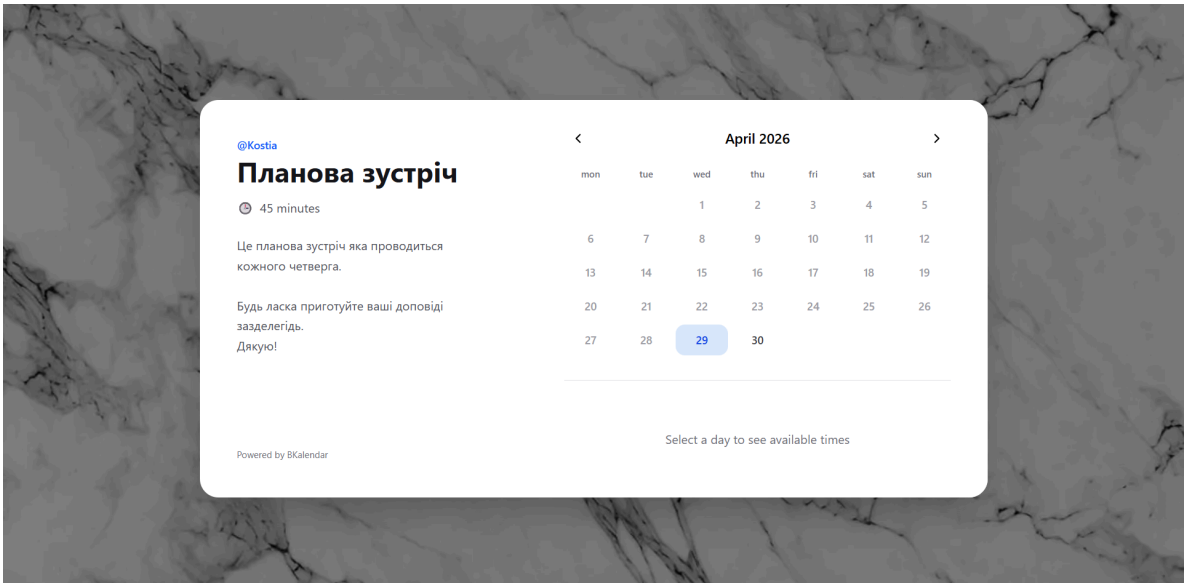


Рис. 3.10 Інтерактивний Календар

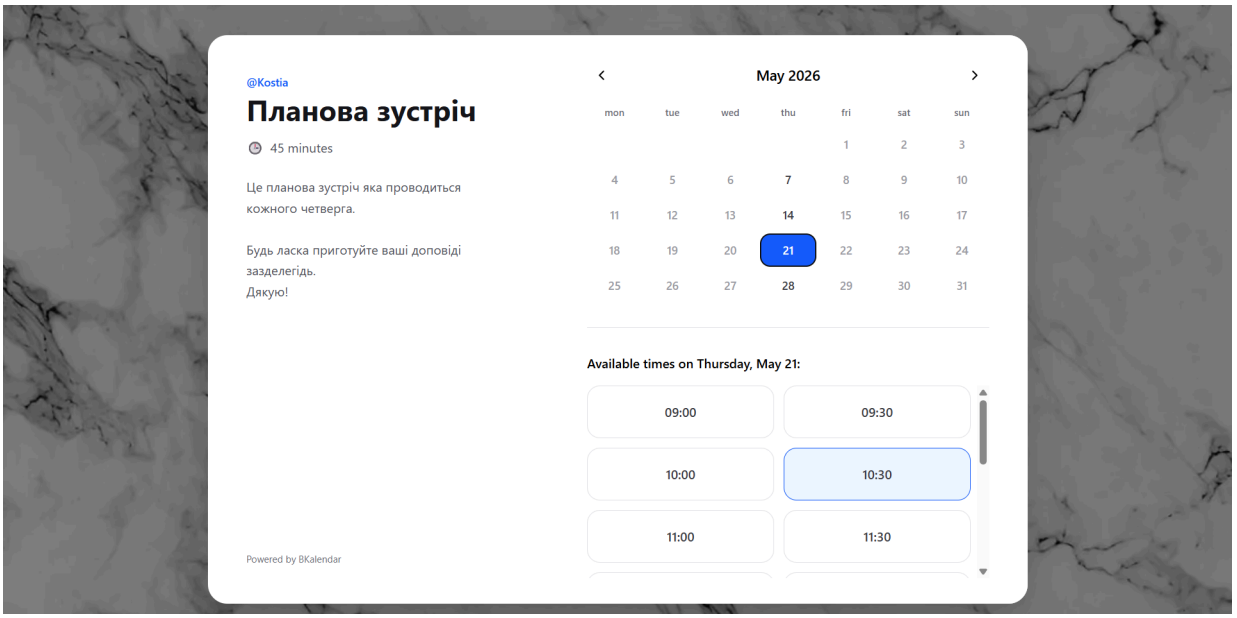


Рис. 3.11 Інтерактивний Календар, часові слоти

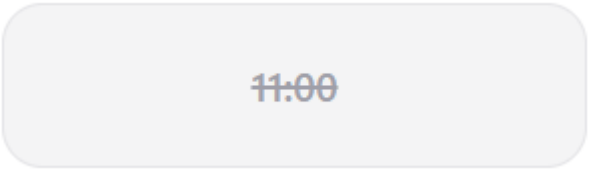


Рис. 3.12 Зайнятий Тайм-слот

Блокування зайнятих слотів забезпечує коректність бронювання та запобігає накладенню зустрічей. Це підвищує надійність системи та довіру користувачів до сервісу.

3.2.5 Сторінка бронювання зустрічей

Сторінка бронювання зустрічей ([username]/[booking-uri]/page.tsx) є публічною сторінкою, яку бачить гість при переході за персональним посиланням (Рис. 3.10 та 3.11). На цій сторінці відображається повна інформація про тип події: назва, тривалість та опис. Нижче розташовано інтерактивний календар компонента TimePicker, який дозволяє гостю обрати зручну дату та час. Після вибору дати та часу з’являється форма введення даних гостя, реалізована у компоненті BookingForm.tsx (Рис. 3.13).

Після успішного підтвердження бронювання гість бачить повідомлення про успішне створення зустрічі та отримує посилання на онлайн-зустріч у Google Meet (Рис. 3.14 та 3.15).

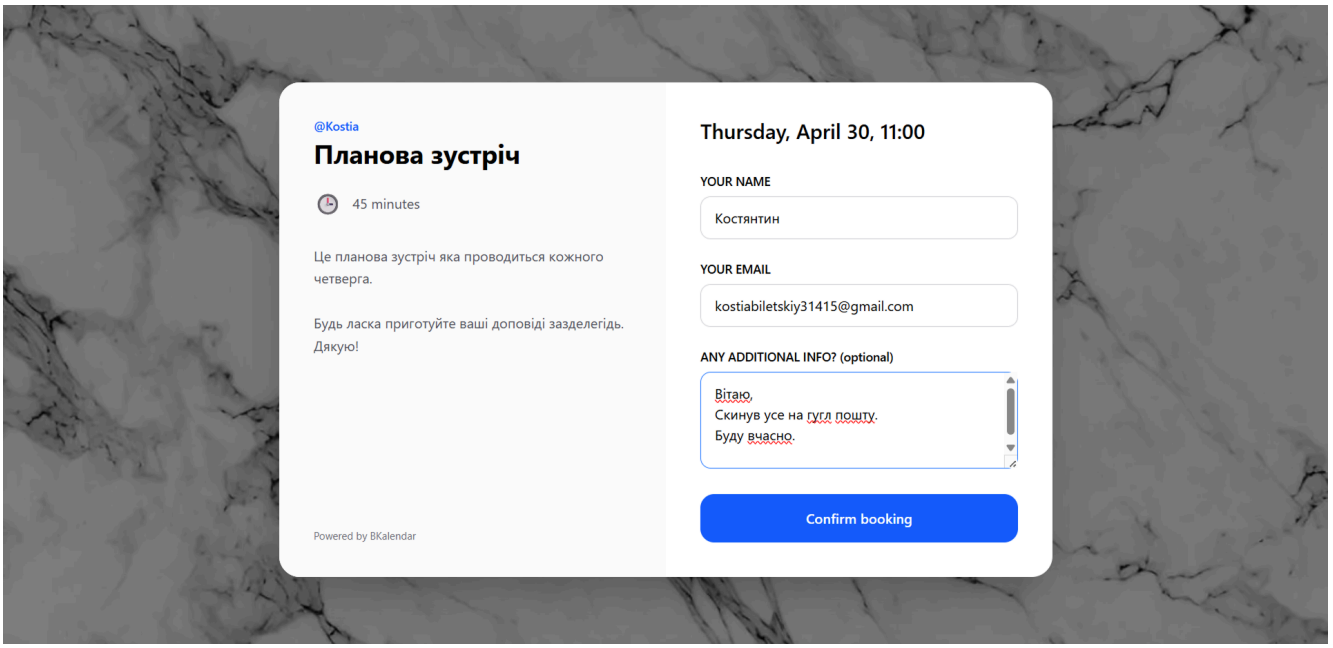


Рис. 3.13 Форма введення даних гостя

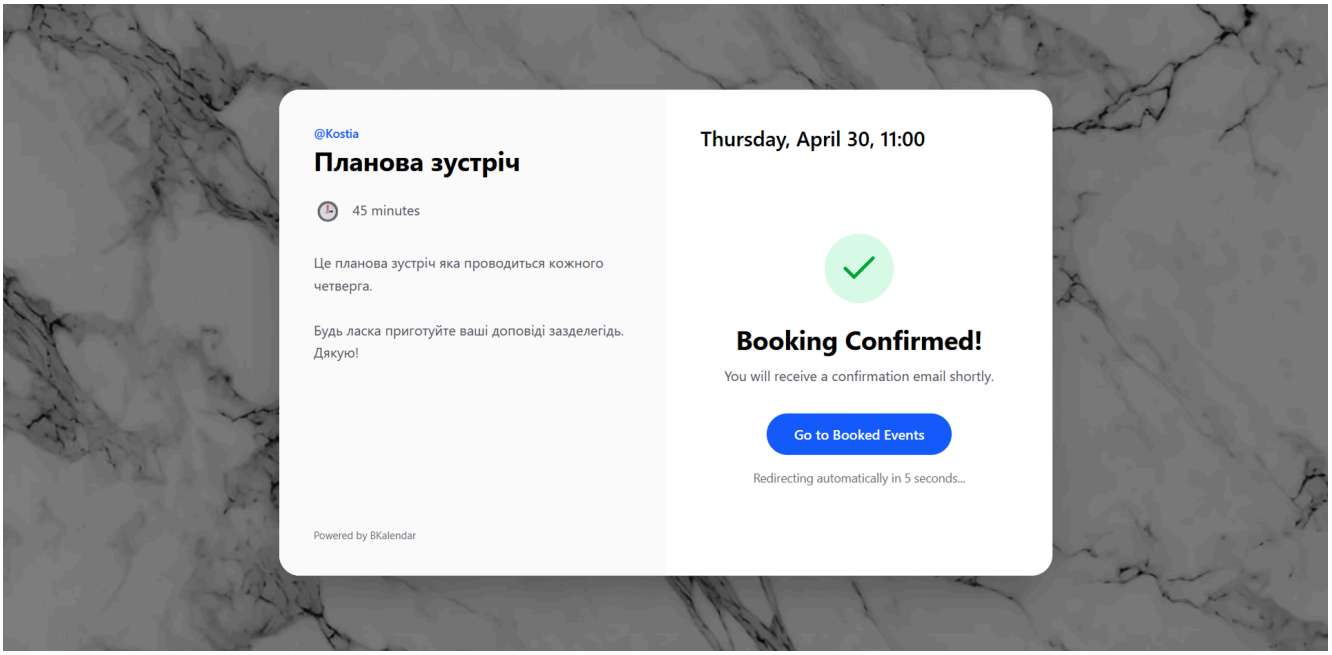


Рис. 3.14 Успішне бронювання

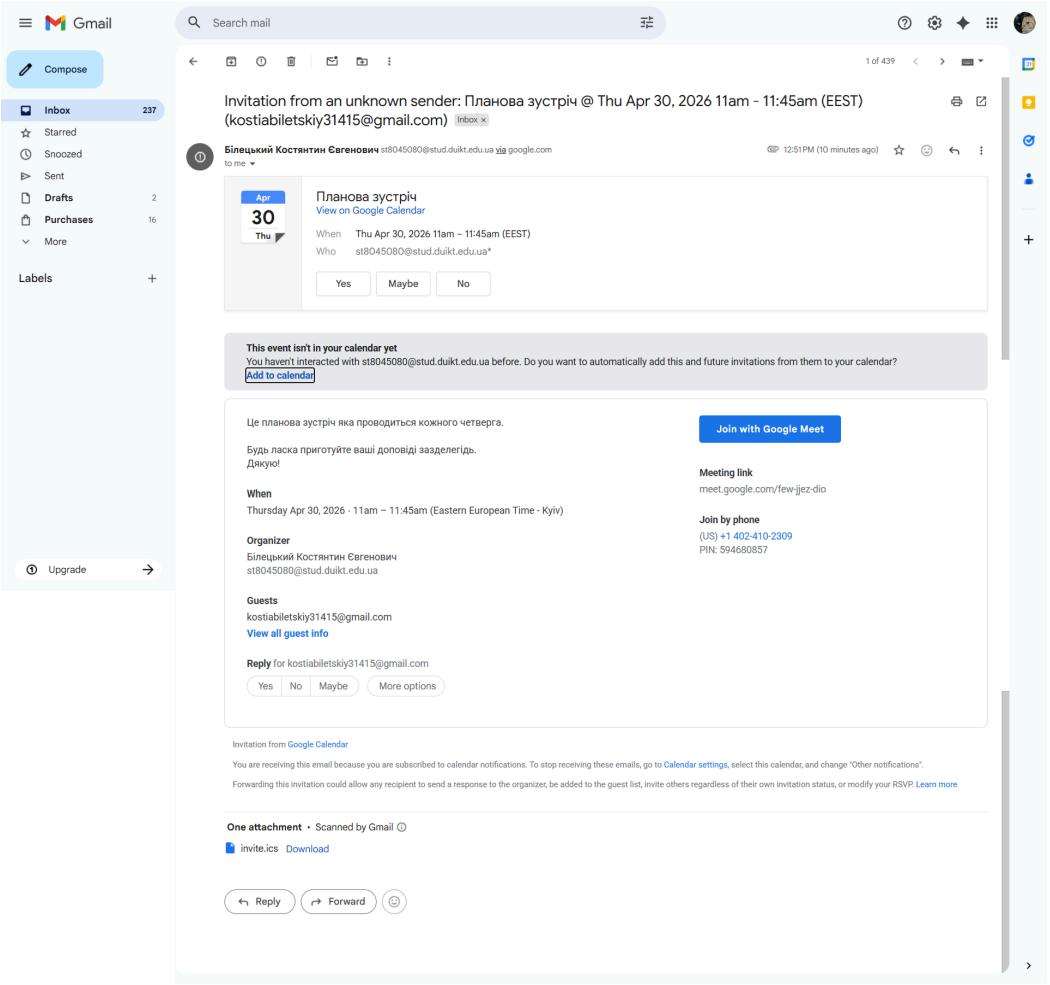


Рис. 3.15 Email з посиланням на зустріч, який отримує гість після успішного бронювання

Отримання підтвердження про успішне бронювання та посилання на Google Meet завершує процес запису на зустріч з боку гостя. Це забезпечує прозорість процесу бронювання та зручну комунікацію між організатором та учасником зустрічі.

3.2.6 Адаптивність та дизайн

Весь інтерфейс системи побудовано з використанням Tailwind CSS [22] і повністю адаптивний. Система коректно відображається на комп'ютерних моніторах, планшетах та мобільних пристроях. Використано сучасний мінімалістичний дизайн з синім акцентним кольором (#3B82F6) та чисті, мінімалістичні форми.

Рівень представлення системи BKalendar реалізовано з використанням сучасних технологій та підходів до розробки інтерфейсів. Інтуїтивно зрозумілий дизайн, зручна навігація та інтерактивні елементи забезпечують високий рівень користувацького досвіду як для організаторів зустрічей, так і для гостей.

3.3 Рівень логіки (серверна частина)

Рівень логіки системи BKalendar повністю реалізований через API Routes у Next.js 16 [16]. Вся бізнес-логіка зосереджена в директорії src/app/api/. Нижче наведено детальний опис кожного модуля серверної частини.

3.3.1 Система авторизації та управління сесіями

Система авторизації базується на протоколі OAuth 2.0 через платформу Nylas.

Файл src/app/api/auth/route.ts містить єдиний обробник GET-запиту, який генерує URL для авторизації користувача. Повний код наведено в додатку Б. За допомогою методу nylas.auth.urlForOAuth2() формується посилання на сторінку авторизації Google з використанням clientId та redirectUri з конфігурації Nylas. Після формування URL відбувається автоматичне перенаправлення користувача за допомогою функції redirect().

Після успішної авторизації в Google користувач перенаправляється на маршрут /api/oauth/exchange, де відбувається обмін авторизаційного коду на grantId та email.

Файл src/libs/session.ts містить дві ключові допоміжні функції для управління сесіями користувача. Функція setSession(grantId, email) зберігає отримані дані авторизації в HTTP-only cookies з терміном дії 7 днів. Функція getSession() дозволяє отримати поточні значення grantId та email з cookies.

Ці функції використовуються в усіх захищених API-маршрутах системи для перевірки авторизації користувача.

3.3.2 Робота з профілем користувача

Маршрут src/app/api/profile/route.ts містить два основні обробники. GET-запит /api/profile повертає поточне унікальне ім’я користувача (username), яке зберігається в базі даних. PUT-запит /api/profile дозволяє оновити username та зберегти grantId від платформи Nylas. У разі відсутності профілю для даного користувача новий запис створюється автоматично завдяки опції upsert.

Цей маршрут забезпечує користувачеві можливість встановлювати та змінювати унікальне ім’я (username), яке використовується для формування персональних посилань на бронювання. Повний код наведено в Додатку Б.

3.3.3 Управління типами подій (CRUD)

Управління типами подій реалізовано через два основні API-маршрути фреймворку Next.js.

Файл src/app/api/event-types/route.ts відповідає за створення нового типу події. При надходженні POST-запиту /api/event-types дані валідації та зберігаються в базі даних за допомогою методу EventTypeModel.create().

Файл src/app/api/event-types/[id]/route.ts містить обробники для редагування та видалення. PUT-запит /api/event-types/[id] оновлює існуючий тип події за допомогою методу findByIdAndUpdate. DELETE-запит /api/event-types/[id] повністю видаляє тип події з бази даних за допомогою методу findByIdAndDelete.

Модель EventType використовує вкладену схему BookingSchema для зберігання налаштувань доступності по днях тижня.

3.3.4 Отримання зайнятих слотів

Маршрут GET /api/busy призначений для отримання актуальної інформації про зайняті часові інтервали користувача за вказаний період. Він активно використовується компонентом TimePicker.tsx для динамічного блокування вже зайнятих слотів при виборі дати та часу бронювання.

При обробці запиту система виконує пошук усіх бронювань користувача в базі даних за допомогою моделі BookingModel.find() з фільтрацією по username та

діапазону дат (from та to). Отримані дані перетворюються в масив об’єктів, що містить startTime та endTime кожного занятого інтервалу (з урахуванням тривалості події), і повертається клієнту у форматі JSON. Повний код наведено в Додатку Б.

3.3.5 Створення бронювання та інтеграція з Nylas

Маршрут POST /api/bookings є найскладнішим і найбільш критичним у всій системі. Він відповідає за повний цикл створення бронювання та автоматичну інтеграцію з Google Meet.

При надходженні запиту спочатку виконується валідація обов’язкових вхідних даних. Далі система знаходить профіль користувача та відповідний тип події в базі даних. Після успішної перевірки через платформу Nylas створюється нова подія в Google Calendar користувача. Під час створення події автоматично генерується конференція Google Meet. Отримане посилання на зустріч зберігається разом з усіма даними бронювання в колекції Booking (включаючи eventTitle, eventLength та meetingLink).

Після успішного збереження клієнту повертається результат з полем success: true та посиланням на створену онлайн-зустріч. Повний код наведено в Додатку Б.

3.3.6 Логіка виходу з системи

Маршрут GET /api/logout забезпечує безпечний вихід користувача з системи. При його виклику відбувається очищення HTTP-only cookies grantId та email шляхом встановлення їм терміну дії в нуль (maxAge: 0). Після цього користувач автоматично перенаправляється на головну сторінку з параметром ?logged-out=1, який може використовуватися для відображення відповідного повідомлення. Повний код наведено в Додатку Б.

3.3.7 Обробка помилок та підключення до MongoDB

У всіх API Routes системи BKalendar використовується єдиний стандартизований підхід до підключення до бази даних MongoDB. Він включає

попереднє налаштування публічних DNS-серверів та встановлення таймауту з’єднання. Детальний опис цього механізму наведено в підрозділі 3.4.1.

Такий підхід забезпечує стабільну роботу з базою даних у середовищі розробки та усуває типові проблеми з розпізнаванням SRV-записів хмарного MongoDB Atlas. Повний код наведено в Додатку Б.

3.4 Робота з базою даних

BKalendar використовує MongoDB [14] як основну базу даних та Mongoose [15] як ODM (Object Data Modeling) для роботи з нею. Вибір MongoDB [14] обумовлений гнучкою схемою даних, високою швидкістю роботи з документами та зручною інтеграцією з Next.js [16].

3.4.1 Підключення до бази даних

Підключення до MongoDB [14] реалізовано в усіх API Routes за допомогою єдиного стандартизованого підходу. Перед встановленням з’єднання система налаштовує публічні DNS-сервери (1.1.1.1 та 8.8.8.8) за допомогою функції setServers. Після цього виконується підключення до бази даних за допомогою mongoose.connect() з параметром serverSelectionTimeoutMS: 8000.

Використання публічних DNS-серверів було додано спеціально для вирішення проблем з SRV-записами, які періодично виникали у середовищі розробки.

3.4.2 Моделі бази даних

У системі реалізовано три основні моделі бази даних, які розташовані в папці src/models/.

Модель EventType призначена для зберігання типів подій, створених користувачем. Вона містить поля: email власника, назву події (title), опис (description), тривалість у хвилинах (length), вкладену схему bookingTimes для

налаштувань доступності по днях тижня, а також унікальне поле uri, яке використовується для формування персонального посилання на бронювання.

Модель Profile зберігає персональні дані користувача. Вона включає електронну пошту (email), унікальне ім’я користувача (username) та grantId – токен авторизації від платформи Nylas, необхідний для роботи з календарем.

Модель Booking відповідає за зберігання інформації про кожне заброньоване бронювання. Вона містить дані про гостя (ім’я, email, додаткові нотатки), дату і час зустрічі (when), посилання на тип події, назву та тривалість події, username організатора, uri бронювання, а також посилання на створену зустріч у Google Meet (meetingLink). Повний код наведено в Додатку Б.

3.4.3 Переваги використання MongoDB

Використання MongoDB [14] як основної бази даних у проєкті BKalendar зумовлено кількома важливими перевагами. Гнучка схема даних ідеально підходить для зберігання складних налаштувань доступності по днях тижня у вигляді вкладених об’єктів (bookingTimes). База забезпечує високу швидкість роботи з документами, що критично важливо для динамічних операцій, таких як отримання зайнятих слотів у реальному часі. Крім того, MongoDB добре інтегрується з Next.js [16] через ODM Mongoose [15], що спрощує роботу з моделями, валідацію даних та підтримку коду в цілому.

Три основні моделі (EventType, Profile, Booking) забезпечують повноцінне та ефективне зберігання інформації про типи подій, профілі користувачів та заброньовані зустрічі.

4. ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ

4.1 Функціональне тестування

Функціональне тестування проводилося з метою перевірки відповідності реалізованого функціоналу вимогам, сформованим у розділі 2. Тестування здійснювалося без аналізу вихідного коду.

4.1.1 Методика тестування

Для функціонального тестування було розроблено набір тест-кейсів, які охоплюють основні сценарії використання системи. Кожен тест-кейс містить опис тесту, вхідні дані, очікуваний результат, фактичний результат, а також статус виконання (Пройдено / Не пройдено). Такий структурований підхід дозволяє системно перевірити відповідність реалізованого функціоналу поставленим вимогам.

4.1.2 Результати функціонального тестування

Для перевірки було розроблено та виконано набір тест-кейсів, які охоплюють усі ключові функціональні можливості платформи: авторизацію користувачів через Google, створення, редагування та видалення типів подій, повний цикл бронювання зустрічей як від імені гостя, автоматичне визначення та блокування зайнятих слотів, роботу з персональним профілем користувача (включаючи встановлення унікального username) та перегляд історії забронюваних зустрічей. Кожен тест-кейс містив чіткий опис тесту, вхідні дані, очікуваний результат, фактичний результат та статус виконання.

Результати виконання всіх тест-кейсів представлено в таблиці 4.1.

Таблиця 4.1

Тест-кейси функціонального тестування

Назва тест-кейсу	Очікуваний результат	Фактичний результат	Статус
Авторизація через Google	Успішний редирект у кабінет	Успішний редирект	Пройдено
Створення типу події	Тип події з’являється у списку	Тип події створено	Пройдено
Редагування типу події	Зміни зберігаються та відображаються	Зміни збережено	Пройдено
Видалення типу події	Тип події видаляється зі списку	Тип події видалено	Пройдено
Бронювання зустрічі (як гість)	Зустріч створюється + надсилається посилання	Зустріч створено	Пройдено
Перевірка блокування зайнятих слотів	Зайняті слоти відображаються сірим кольором та не є активними	Слоти заблоковано	Пройдено
Перегляд заброньованих зустрічей	Список зустрічей відображається коректно	Список відображено	Пройдено
Встановлення власного username	Посилання для бронювання змінюється	Посилання оновлено	Пройдено

Усі розроблені тест-кейси були успішно пройдені. Реалізований функціонал повністю відповідає функціональним вимогам, сформованим у розділі 2. Система стабільно працює при виконанні ключових операцій: авторизації через Google, створенні, редагуванні та видаленні типів подій, бронюванні зустрічей, автоматичному блокуванні зайнятих слотів, перегляді історії бронювань та роботі з профілем користувача.

4.1.3 Детальний опис тест-кейсів

Тест-кейс №1: Авторизація через Google

Мета: Перевірити можливість входу в систему через Google-акаунт.

Вхідні дані: Обліковий запис Google.

Очікуваний результат: Після авторизації користувач перенаправляється на сторінку Dashboard.

Фактичний результат: Користувач успішно авторизувався та потрапив на Dashboard. (Див. рис. 4.1-4.4)

Статус: Пройдено.

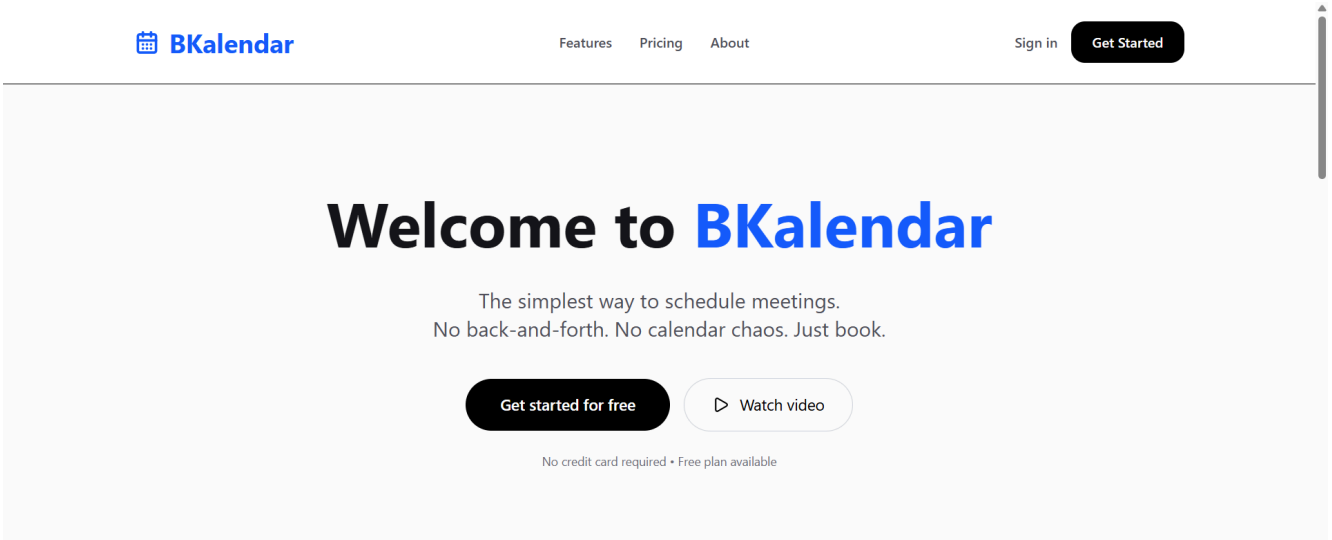


Рис. 4.1 Немає входу в систему (Логіну)

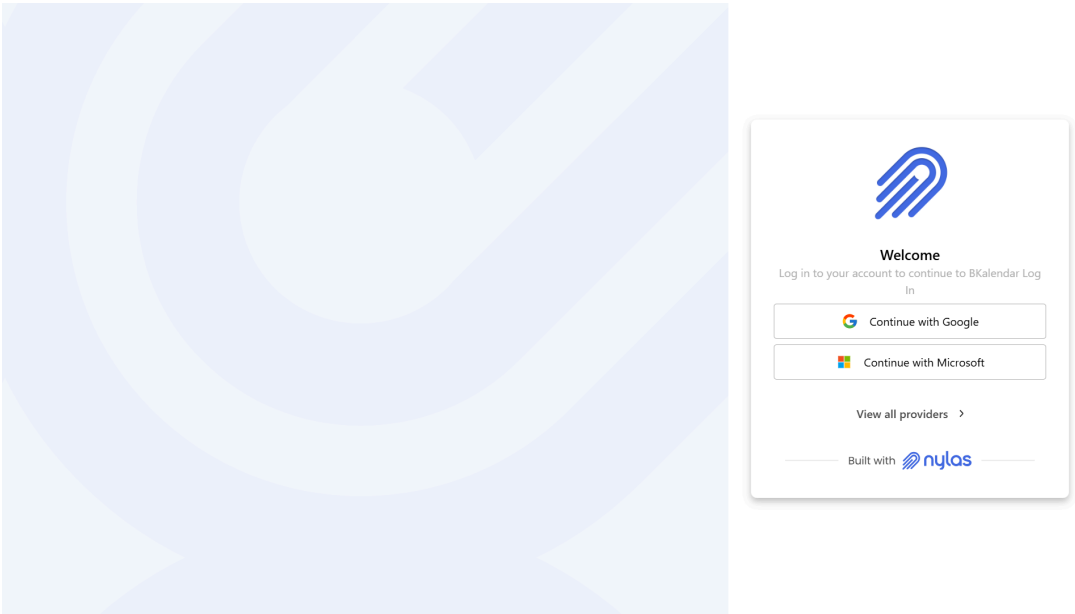


Рис. 4.2 Натискаємо на кнопку авторизації/реєстрації

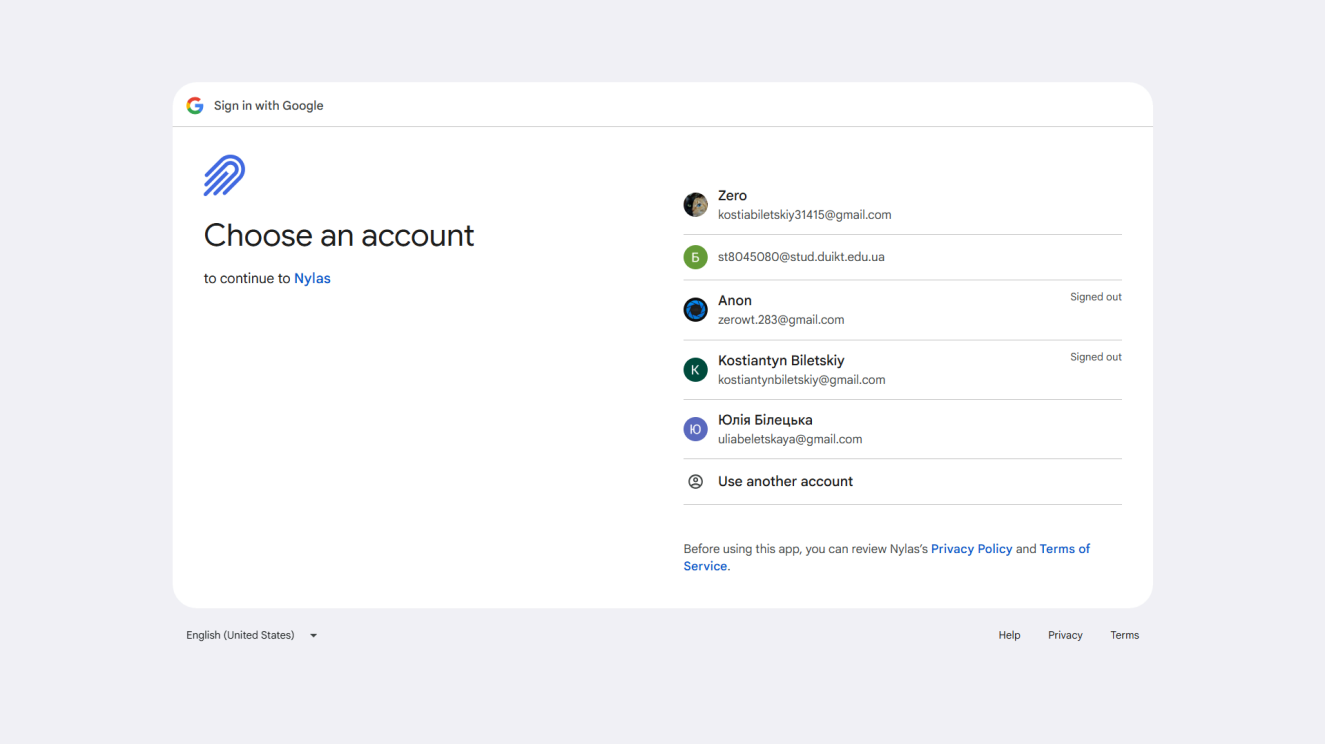


Рис. 4.3 Далі виконується звичайний вхід в гугл аккаунт

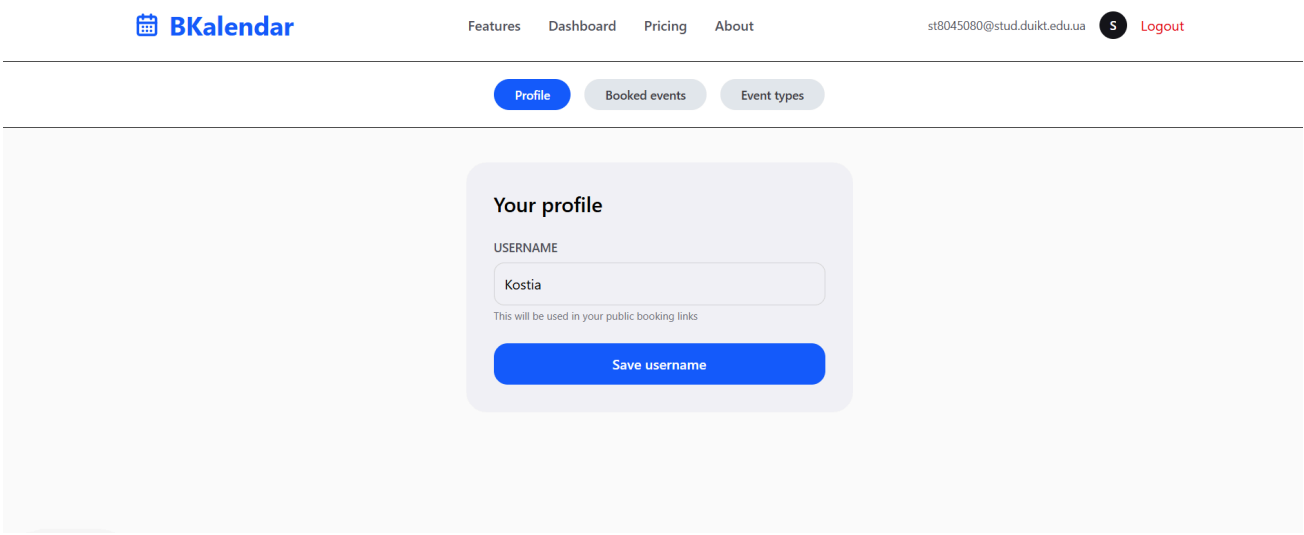


Рис. 4.4 Авторизація успішна

Тест-кейс №2: Створення типу події

Мета: Перевірити функціонал створення нового типу події.

Вхідні дані: Назва «Консультація», тривалість 45 хв, доступність Пн–Пт 10:00–18:00.

Очікуваний результат: Новий тип події з’являється у списку Event types.

Фактичний результат: Тип події успішно створено та відображається у списку. (Див. рис. 4.5 та 4.6)

Статус: Пройдено.

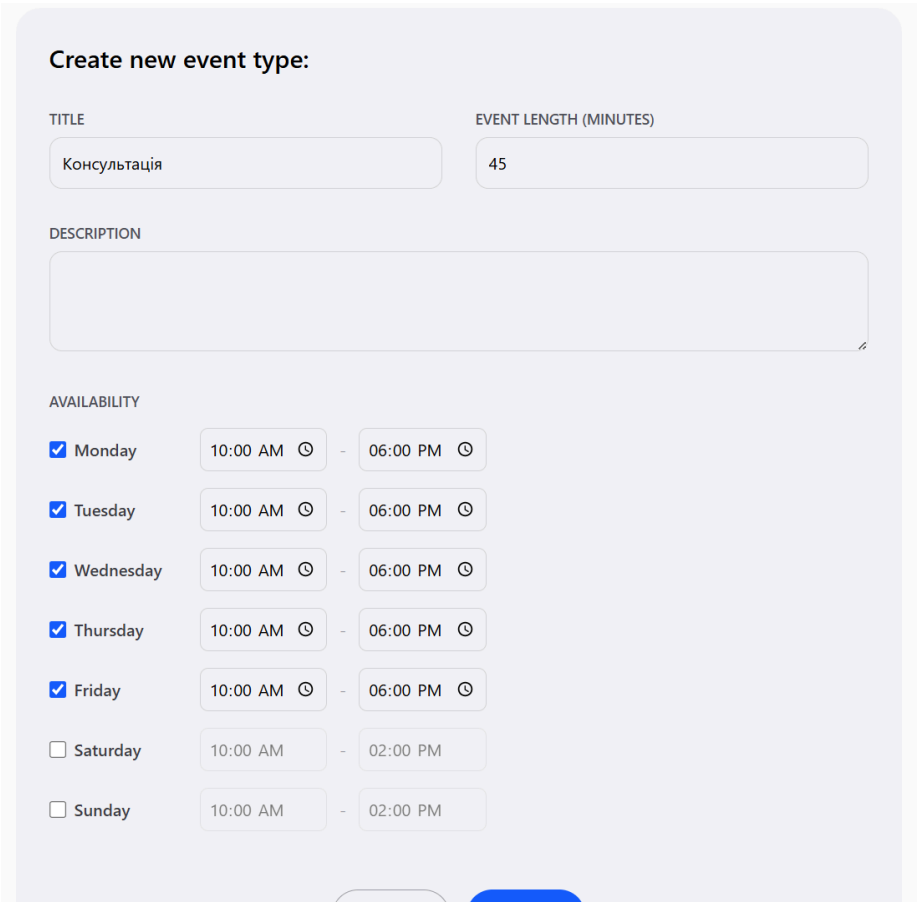


Рис. 4.5 Заповнення форми створення типу події

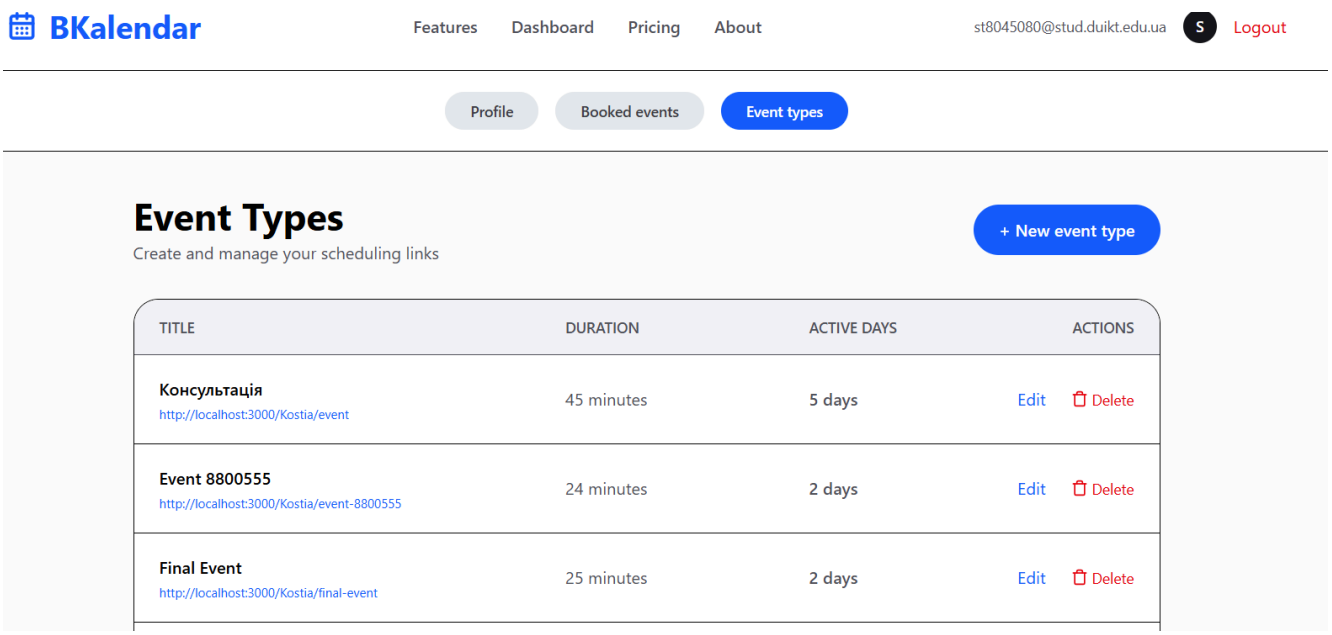


Рис. 4.6 Новий тип події збережений успішно

Тест-кейс №3: Редагування типу події

Мета: Перевірити можливість редагування існуючого типу події.

Вхідні дані: Зміна тривалості з 45 на 25 хвилин.

Очікуваний результат: Зміни відображаються у списку.

Фактичний результат: Тривалість успішно змінено.

Статус: Пройдено.

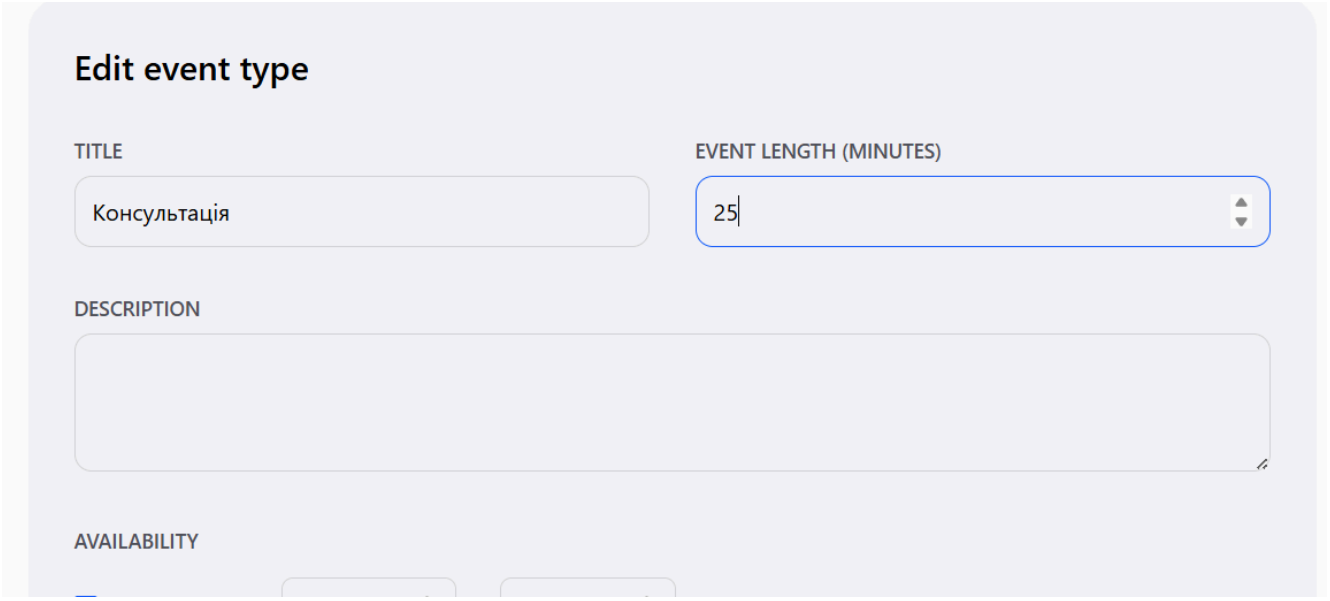


Рис. 4.7 Редагування події (зміна тривалості з 45 на 25 хв)

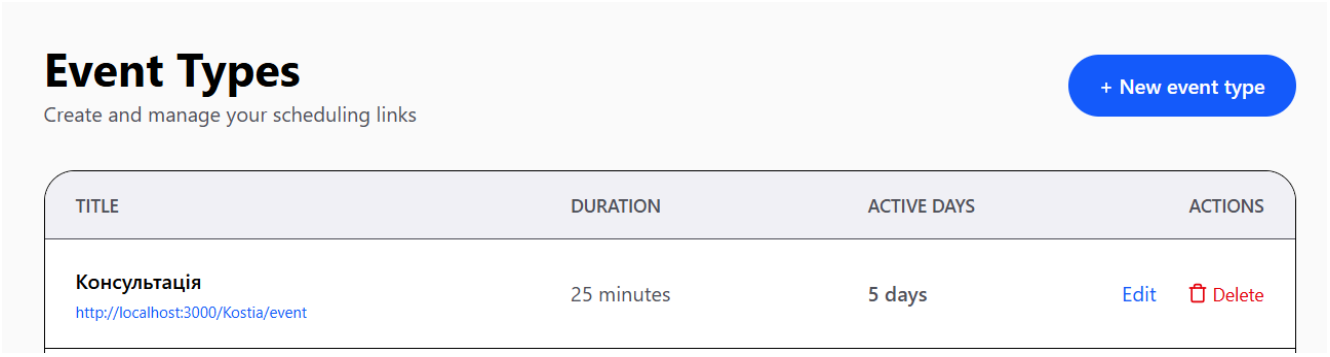


Рис. 4.8 Тривалість змінена успішно

Тест-кейс №5: Бронювання зустрічі (як гість)

Мета: Перевірити повний процес бронювання з боку гостя.

Вхідні дані: Ім’я, email, обраний час.

Очікуваний результат: Зустріч створюється, гість отримує Email з деталями посилання.

Фактичний результат: Зустріч створено, Email надіслано. (Див. Рис. 4.9-

Статус: Пройдено.

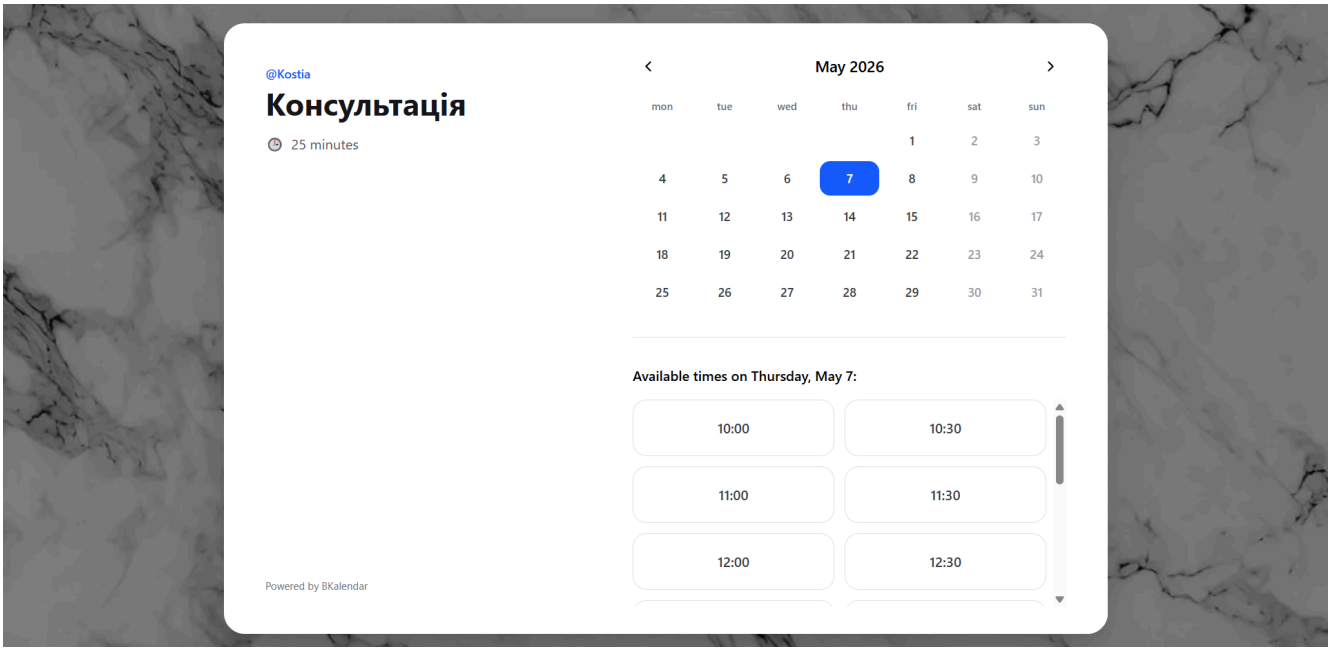


Рис. 4.9 Гість робить вибір дня в календарі.

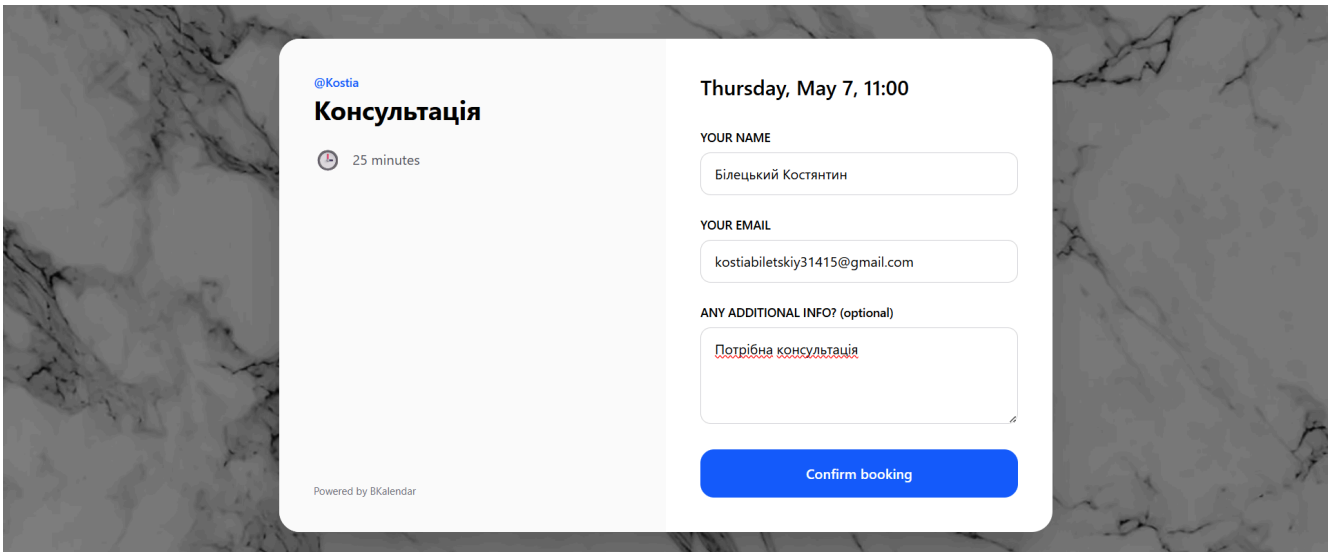


Рис. 4.10 Гість робить вибір дня в календарі.

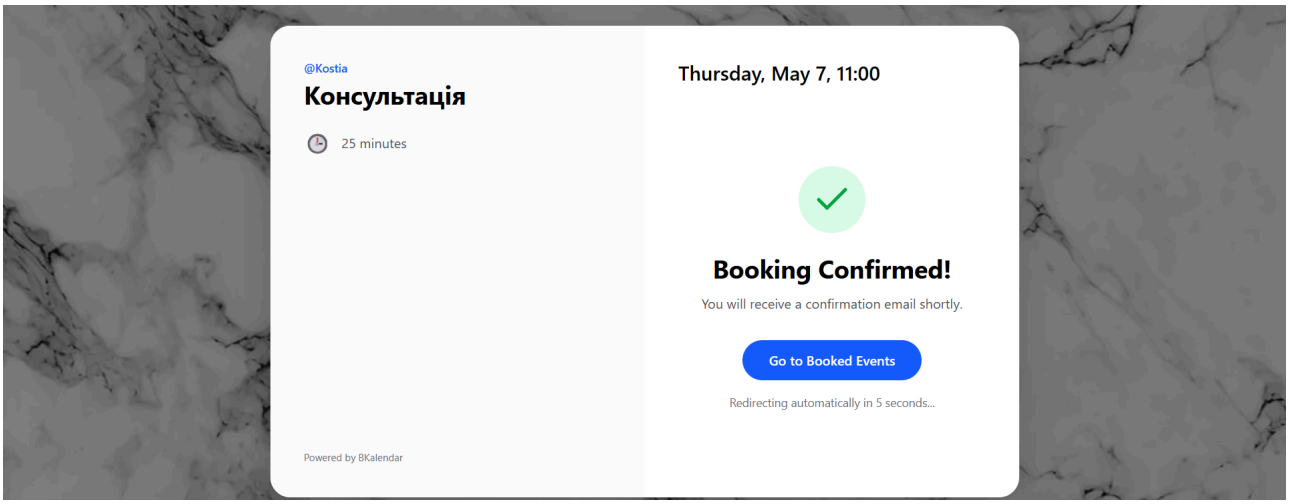


Рис. 4.11 Запис успішний

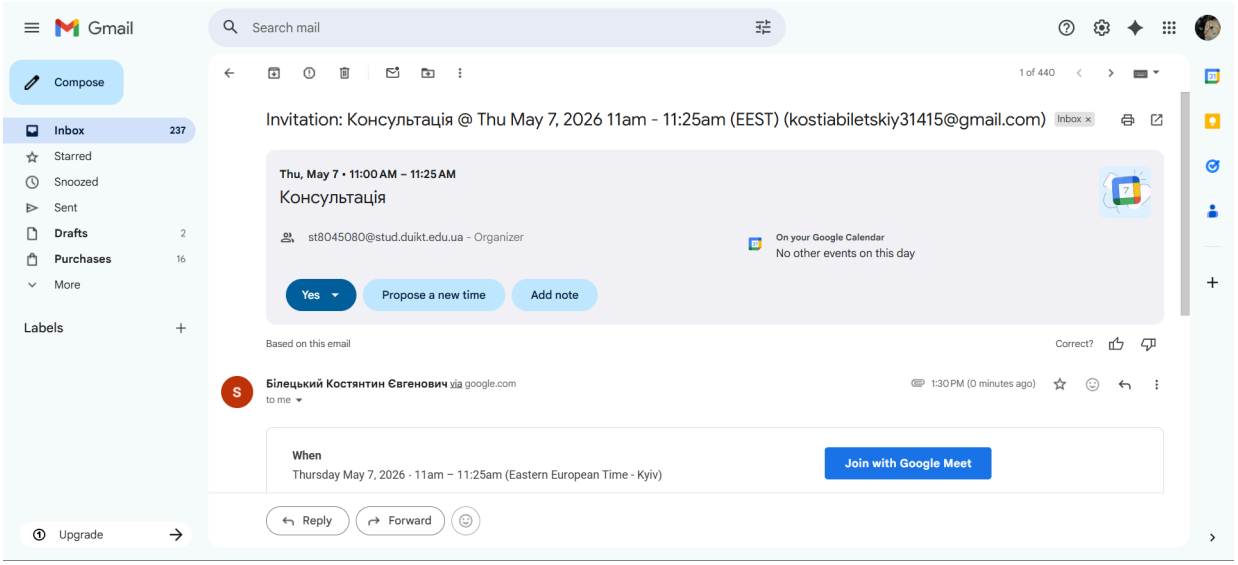


Рис. 4.12 Гість отримує Email з посиланням на зустріч

Тест-кейс №6: Перевірка блокування зайнятих слотів

Мета: Перевірити автоматичне блокування зайнятих слотів.

Вхідні дані: Два бронювання на один день та час.

Очікуваний результат: Зайняті слоти відображаються сірим кольором з перекресленним текстом.

Фактичний результат: Слоти коректно заблоковані. (Див. рис. 4.13)

Статус: Пройдено.

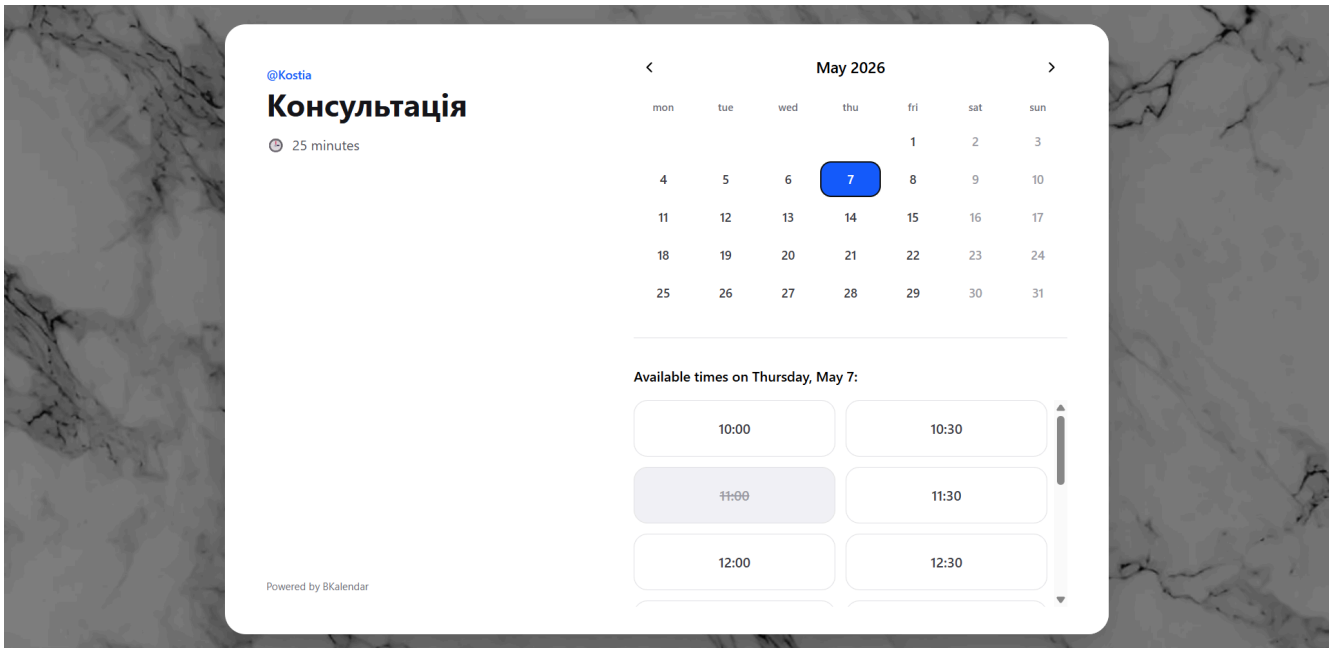


Рис. 4.13 Вибираємо той самий день що й в попередньому тест-кейсі і бачимо що зайнятий тайм слот успішно заблокований.

У ході функціонального тестування було перевірено 8 основних сценаріїв використання системи. Всі тест-кейси були пройдені успішно. Реалізований функціонал повністю відповідає вимогам, сформованим у розділі 2. Система стабільно працює при виконанні основних операцій: авторизація, створення та редагування типів подій, бронювання зустрічей та автоматичне блокування зайнятих слотів.

4.2 Нефункціональне тестування

Нефункціональне тестування BKalendar проводилося з метою перевірки якісних характеристик системи, таких як продуктивність, безпека, адаптивність та сумісність. Тестування виконувалося з використанням вбудованих інструментів браузера (Chrome DevTools) та ручного тестування.

4.2.1 Тестування продуктивності

Мета: Оцінити швидкість завантаження сторінок та час відповіді API.

Методика: Використовувались інструменти Chrome DevTools (вкладка Network та Lighthouse [11]).

Результати тестування продуктивності розписані у таблиці 4.2. Було протестовано час завантаження різних сторінок платформи використовуючи Lighthouse [11].

Таблиця 4.2

Результати тестування продуктивності

Сторінка / Запит	Час завантаження (с)	Оцінка Lighthouse	Статус
Головна сторінка	1.1	100	Відмінно
Dashboard (після авторизації)	1.5	98	Відмінно
Форма створення типу події	0.9	95	Відмінно
Інтерактивний календар (TimePicker)	2.1	85	Добре
API /api/bookings (POST)	0.6	–	Добре
API /api/busy (GET)	0.4	–	Відмінно

Аналіз результатів тестування продуктивності свідчить про ефективну оптимізацію як клієнтської, так і серверної частини платформи BKalendar. головна сторінка та особистий кабінет. Інтерактивний календар, попри складність реалізації та динамічну генерацію слотів, також працює зі стабільною швидкістю. Всі сторінки завантажуються менше ніж за 3 секунди. Загальна продуктивність системи оцінюється як добра.

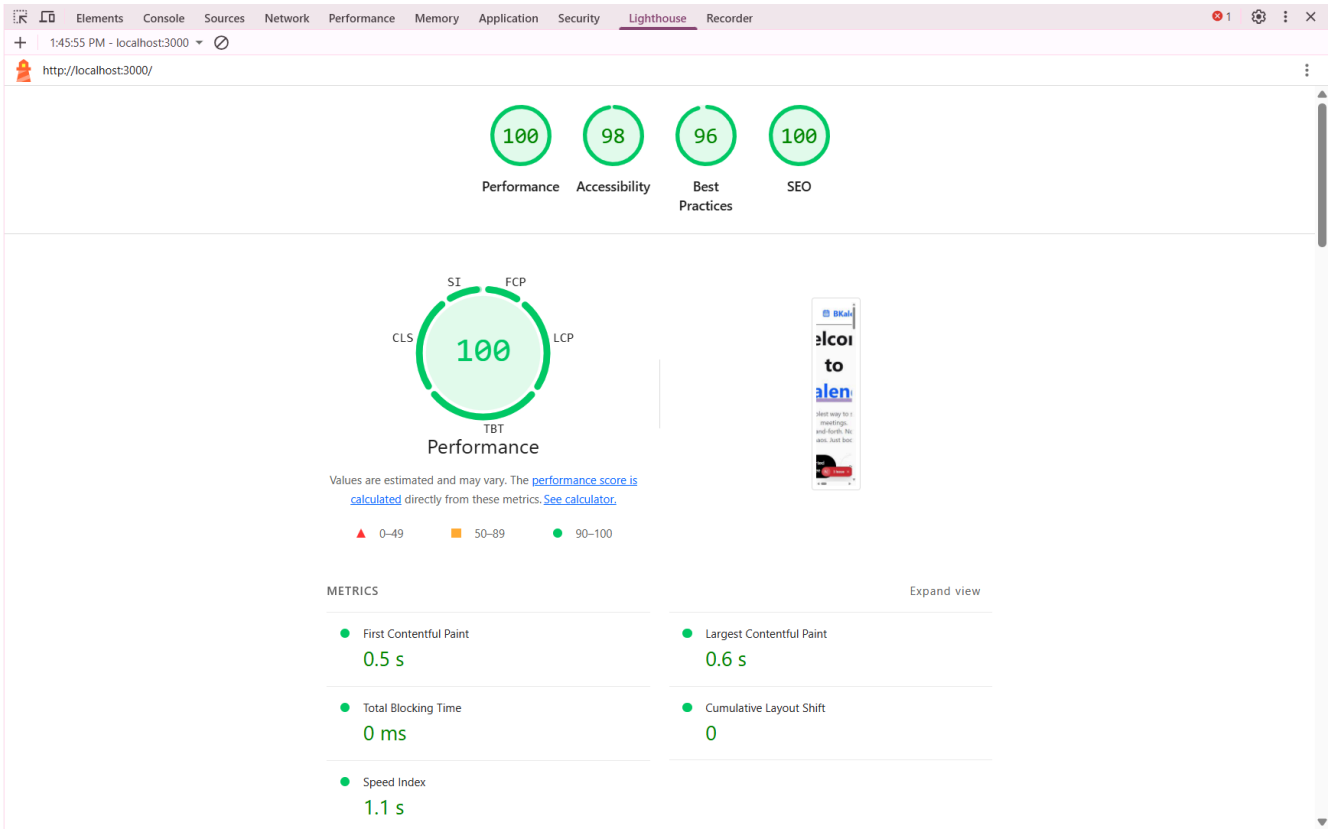


Рис. 4.14 Lighthouse звіт для головної сторінки

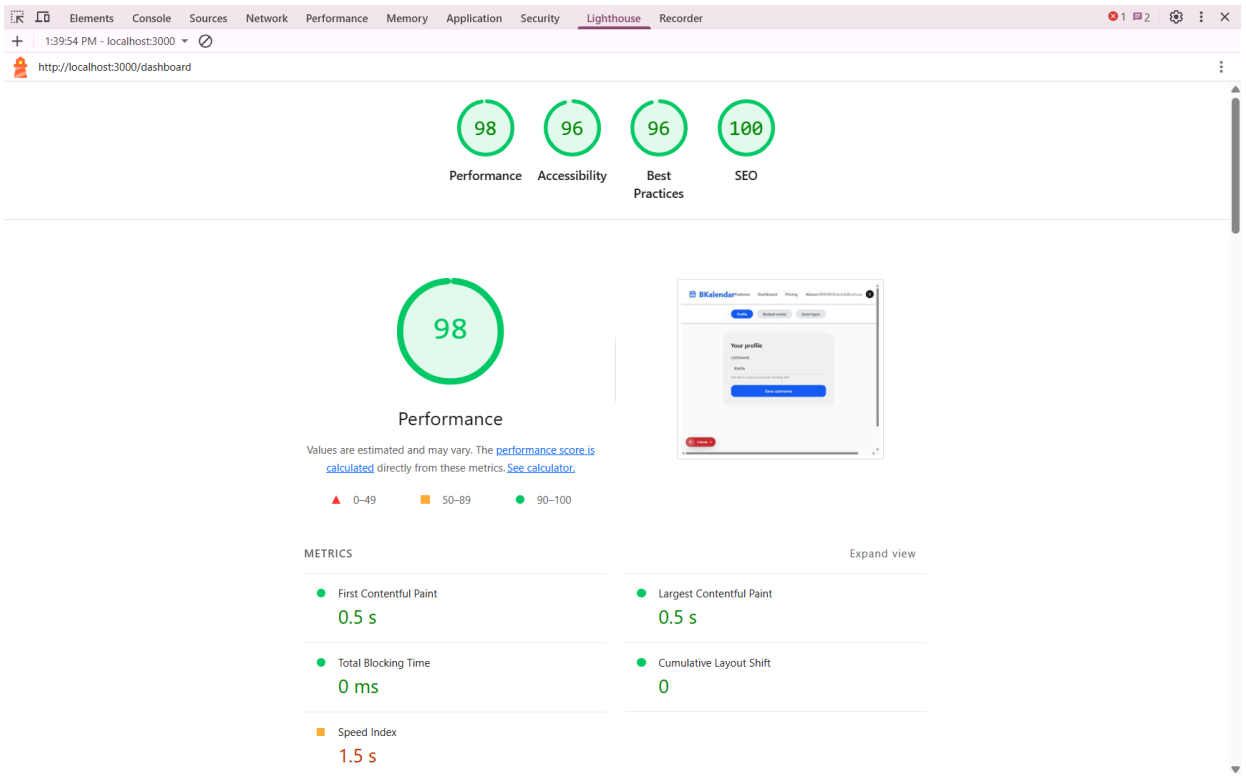


Рис. 4.15 Lighthouse звіт для Dashboard сторінки

Отримані результати тестування продуктивності підтверджують високу швидкодію системи. Всі сторінки завантажуються менше ніж за 3 секунди, а високі оцінки Lighthouse (98–100 балів) свідчать про якісну оптимізацію як фронтенду, так і серверної частини застосунку.

4.2.2 Тестування безпеки

Мета тестування безпеки полягала у перевірці захисту системи від основних веб-вразливостей. Було проаналізовано ключові аспекти захисту, зокрема аутентифікацію та управління сесіями, захист від CSRF-атак, захист від XSS-атак, а також безпеку API Routes.

У процесі тестування встановлено, що cookies з grantId та email мають атрибути httpOnly та secure, а сесія автоматично завершується через 7 днів. Для захисту від CSRF-атак усі форми використовують fetch-запити з JSON, без застосування традиційних HTML-форм. Від XSS-атак дані на сторінках екрануються засобами React. Крім того, всі захищені маршрути API перевіряють наявність grantId та email через функцію getSession().

Результати тестування показали, що система демонструє високий рівень безпеки. Основні веб-вразливості (CSRF, XSS, несанкціонований доступ) відсутні.

4.2.3 Тестування адаптивності та сумісності

Результати тестування адаптивності системи на різних пристроях та браузерах наведено в таблиці 4.3. Тестування проводилося вручну на десктопних комп'ютерах (1920×1080), планшетах (768×1024) та мобільних пристроях (375×667) з використанням браузерів Google Chrome та Microsoft Edge. Оцінювалися такі параметри, як відображення меню, робота інтерактивного календаря та форм введення даних. Результати тестування адаптивності системи наведено в таблиці 4.3.

Таблиця 4.3

Результати тестування адаптивності

Пристрій / Браузер	Відображення меню	Календар	Форми	Статус
Desktop (Chrome)	Коректно	Коректно	Коректно	Пройдено
Desktop (Edge)	Коректно	Коректно	Коректно	Пройдено
Tablet (Chrome)	Коректно	Добре	Коректно	Пройдено
Mobile (Chrome)	Добре	Добре	Добре	Пройдено

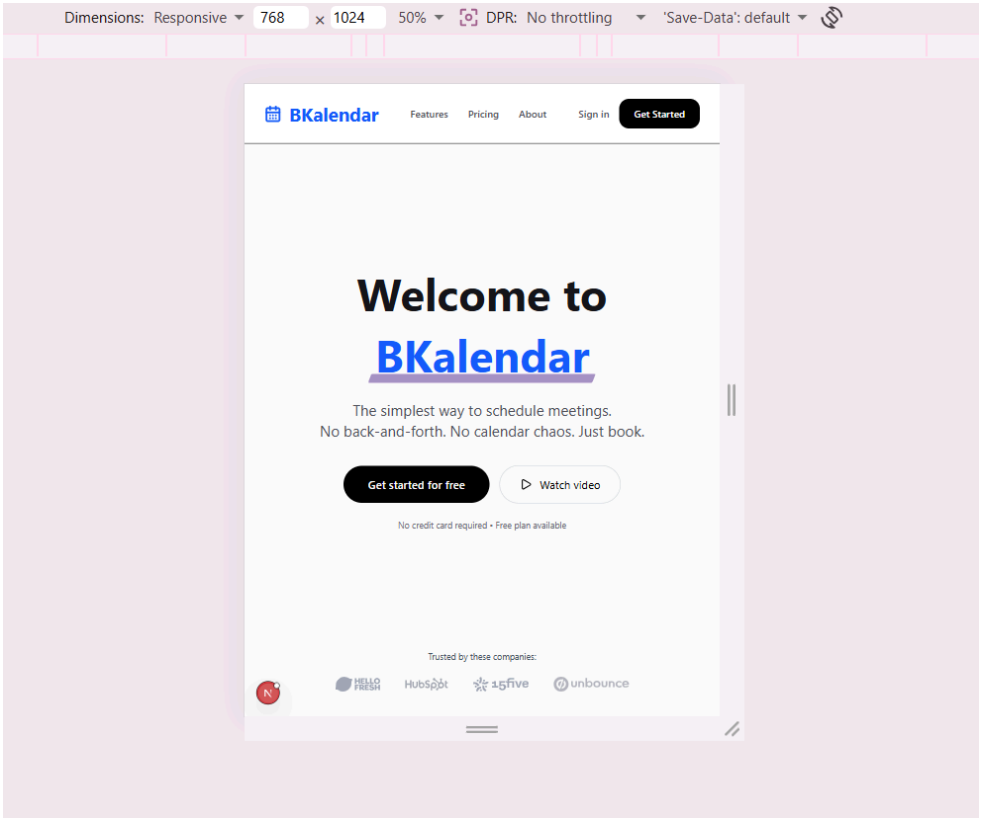


Рис. 4.16 Відображення сторінки на планшеті

Система повністю адаптивна. На мобільних пристроях, планшетах та моніторах різного розміру сторінки платформи коректно масштабуються і зберігають зручність користування.

Нефункціональне тестування системи показало, що вона відповідає сучасним вимогам до веб-сервісів. Продуктивність оцінюється як добра – усі сторінки завантажуються менше ніж за 3 секунди. Рівень безпеки високий – основні веб-вразливості (CSRF, XSS, несанкціонований доступ) відсутні. Адаптивність визнана повною – інтерфейс коректно відображається на всіх типах пристроїв. Загалом система готова до експлуатації та подальшого масштабування.

ВИСНОВКИ

У кваліфікаційній роботі було спроектовано та розроблено сучасну цифровну платформу для планування та резервування зустрічей на базі фреймворку Next.js – BKalendar.

Мета роботи була досягнута. Розроблена платформа дозволяє користувачам створювати типів подій, гнучко налаштовувати доступність по днях тижня, автоматично визначати зайняті слоти, генерувати персональні посилання для бронювання та зустрічатися у Google Meet.

У ході роботи було виконано всі поставлені завдання:

1. Проведено аналіз існуючих систем онлайн-бронювання (Calendly, YouCanBook.me, Google Calendar) та виявлено їх основні переваги та недоліки.
2. Сформовано вимоги до системи та спроектовано трирівневу архітектуру на основі Next.js 16 та Nylas.
3. Розроблено структуру бази даних для зберігання типів подій, профілів користувачів та заброньованих зустрічей.
4. Спроектовано архітектуру програмного забезпечення з використанням трирівневої архітектури (Presentation Layer, Business Logic Layer, Data Access Layer).
5. Реалізовано функціонал створення типів подій та гнучкого налаштування доступності по днях тижня.
6. Розроблено механізм автоматичного визначення та блокування зайнятих слотів з інтеграцією до календаря користувача через платформу Nylas.
7. Створено інтерактивний календар з вибором часу та автоматичним блокуванням зайнятих слотів.
8. Реалізовано систему профілю користувача з можливістю встановлення унікального імені (username) та генерації персональних посилань для бронювання.

9. Проведено функціональне та нефункціональне тестування розробленого веб-сервісу, яке підтвердило відповідність системи всім вимогам.
10. Оцінено ефективність системи за показниками продуктивності, безпеки та адаптивності.

Платформа поєднує гнучке налаштування доступності, автоматичне блокування зайнятих слотів на основі інтеграції з календарями та сучасні технології (Next.js 16, Nylas, MongoDB). Це дозволяє створювати персоналізовані посилання для бронювання та забезпечує зручне управління профілем користувача з власним username.

Створений веб-сервіс може бути повноцінно використаний як самостійне рішення для особистих та бізнес-потреб, а також як основа для створення корпоративних систем бронювання. Система значно спрощує процес планування зустрічей та підвищує ефективність використання часу.

Цифрова платформа BKalendar відповідає сучасним вимогам до продуктивності, безпеки та адаптивності, що підтверджено результатами тестування. Система готова до використання.

ПЕРЕЛІК ПОСИЛАНЬ

1. Білецький К.С., Герцюк М.М. Визначення методу планування та резервування зустрічей у цифровій платформі на базі Next.js. // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 20.04.2026.
2. Білецький К.С., Герцюк М.М. Інтеграція Nylas API для синхронізації календарів та автоматизації резервування зустрічей у цифровій платформі на базі Next.js. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». 25.04.2026.
3. Building Scalable Web Applications with Next.js [Електронний ресурс]. – Режим доступу: https://vercel.com/blog/building-scalable-web-applications-with-nextjs (дата звернення: 25.04.2026).
4. Calendly Official Website [Електронний ресурс]. – Режим доступу: https://calendly.com/ (дата звернення: 28.02.2026).
5. Color Hunt – Color Palette [Електронний ресурс]. – Режим доступу: https://colorhunt.co/ (дата звернення: 25.02.2026).
6. date-fns Documentation [Електронний ресурс]. – Режим доступу: https://date-fns.org/docs/Getting-Started (дата звернення: 20.03.2026).
7. GitHub – Next.js Repository [Електронний ресурс]. – Режим доступу: https://github.com/vercel/next.js (дата звернення: 15.03.2026).
8. Google Calendar – Appointment Slots [Електронний ресурс]. – Режим доступу: https://support.google.com/calendar/answer/7638168 (дата звернення: 28.02.2026).
9. Google Calendar API Documentation [Електронний ресурс]. – Режим доступу: https://developers.google.com/calendar/api (дата звернення: 10.04.2026).

10. Hotjar – User Behavior Analytics [Електронний ресурс]. – Режим доступу: https://www.hotjar.com/ (дата звернення: 10.04.2026).
11. Lighthouse Documentation [Електронний ресурс]. – Режим доступу: https://developer.chrome.com/docs/lighthouse/ (дата звернення: 25.04.2026).
12. Lucide Icons [Електронний ресурс]. – Режим доступу: https://lucide.dev/ (дата звернення: 20.03.2026).
13. Michele Riva. Real-World Next.js: Build scalable, high-performance, and modern web applications using Next.js, the React framework for production. – Packt Publishing, 2022. – 450 с.
14. MongoDB Documentation [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/docs/ (дата звернення: 15.03.2026).
15. Mongoose Documentation [Електронний ресурс]. – Режим доступу: https://mongoosejs.com/docs/ (дата звернення: 15.03.2026).
16. Next.js Documentation [Електронний ресурс]. – Режим доступу: https://nextjs.org/docs (дата звернення: 10.03.2026).
17. Nylas API Documentation [Електронний ресурс]. – Режим доступу: https://www.nylas.com/ (дата звернення: 05.04.2026).
18. Nylas Quickstart Guide [Електронний ресурс]. – Режим доступу: https://v2-5-0.developer.nylas.com/docs/v3/getting-started/quickstart/ (дата звернення: 05.04.2026).
19. OAuth 2.0 Authorization Framework [Електронний ресурс]. – Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749 (дата звернення: 15.04.2026).
20. React Documentation [Електронний ресурс]. – Режим доступу: https://react.dev/ (дата звернення: 27.02.2026).
21. REST API Design Best Practices [Електронний ресурс]. – Режим доступу: https://restfulapi.net/ (дата звернення: 20.04.2026).
22. Tailwind CSS Documentation [Електронний ресурс]. – Режим доступу: https://tailwindcss.com/docs (дата звернення: 15.03.2026).
23. TypeScript Documentation [Електронний ресурс]. – Режим доступу: https://www.typescriptlang.org/docs/ (дата звернення: 25.02.2026).

24. Web Application Architecture: Principles, Protocols and Practices / L. Shklar, R. Rosen. – 3rd ed. – Wiley, 2022. – 380 с.
25. Yarn Package Manager [Електронний ресурс]. – Режим доступу: https://yarnpkg.com/ (дата звернення: 10.03.2026).
26. YouCanBook.me Official Website [Електронний ресурс]. – Режим доступу: https://youcanbook.me/ (дата звернення: 28.02.2026).

ДОДАТОК А. ПРЕЗЕНТАЦІЯ ДОДАТКА

Державний університет
інформаційно-комунікаційних технологій

Цифрова платформа для планування та резервування зустрічей на базі фреймворку Next.js

Виконав:
студент 4 курсу
групи ТЦР-41
Білецький К.Є.

Науковий керівник
Герцюк М.М.

Київ 2026

Актуальність дослідження

Сьогодні онлайн-бронювання зустрічей стало невід'ємною частиною як особистого, так і професійного життя. Зростання популярності віддаленої роботи, фрілансу та гібридних форматів взаємодії значно збільшило попит на зручні інструменти планування.

Проте традиційні методи (листування та телефонні дзвінки) займають велику кількість часу і можуть створити плутанину. В той час, більшість аналогів мають суттєві обмеження: недостатню гнучкість налаштування доступності, відсутність автоматичного визначення зайнятих слотів, відсутню персоналізацію посилань.

Тому розробка сучасної цифрової платформи, яка поєднує гнучке управління доступністю, автоматичне блокування зайнятих слотів та інтеграцію з Google Meet, є актуальним і затребуваним завданням.

3

Мета

Мета – підвищити ефективність планування та бронювання зустрічей шляхом розробки сучасної цифрової платформи BKalendar, яка дозволяє суттєво спростити процес узгодження часу, автоматизувати визначення вільних слотів та забезпечити глибоку інтеграцію з календарними сервісами, тим самим зменшити час на організацію зустрічей і підвищити продуктивність роботи фрілансерів, викладачів та бізнес-команд.

4

Завдання

1. Провести аналіз існуючих систем онлайн-бронювання.
2. Дослідити недоліки та обмеження популярних рішень.
3. Визначити основні вимоги.
4. Розробити структуру бази даних.
5. Спроектувати архітектуру програмного забезпечення.
6. Реалізувати функціонал створення типів подій та гнучкого налаштування доступності по днях тижня.
7. Розробити механізм автоматичного визначення та блокування зайнятих слотів з інтеграцією до календаря користувача.
8. Створити інтерактивний календар з вибором часу та генерацію персональних посилань для бронювання.
9. Реалізувати систему профілю користувача.
10. Провести тестування розробленого веб-сервісу та оцінити його ефективність.

5

Об’єкт та предмет дослідження

Об’єкт дослідження – процеси онлайн-бронювання зустрічей, управління доступністю користувачів та обмін даними з календарними сервісами в інформаційних системах.

Предмет дослідження – програмне забезпечення для онлайн-бронювання зустрічей з підтримкою гнучкого налаштування доступності по днях тижня, автоматичного визначення зайнятих слотів та інтеграції з зовнішніми календарними сервісами (Nylas).

6

Методи дослідження

Для досягнення поставленої мети в роботі використано комплекс сучасних методів:

1. Теоретичні методи: Порівняння та аналіз існуючих систем онлайн-бронювання. Узагальнення кращих практик і виявлення недоліків.

2. Методи проектування: Об’єктно-орієнтоване та компонентне проектування. Проектування трирівневої архітектури системи.

3. Методи програмної реалізації: Розробка веб-застосунків на базі Next.js 16 (App Router, Server Components, Server Actions). Робота з датами та часом за допомогою бібліотеки date-fns. Інтеграція з зовнішніми сервісами через Nylas API.

4. Методи оцінки: Функціональне тестування. Нефункціональне тестування (продуктивність, безпека, адаптивність).

7

Аналіз аналогів

Аналоги/Сервіси	Сильна сторона аналогу	Недоліки аналогу	Висновок
Calendly	Зручний інтерфейс, швидке створення посилань, інтеграція з Google Calendar	Обмежена гнучкість налаштування доступності по днях тижня, немає автоматичного блокування слотів з повного календаря	Відсутня повна гнучкість по днях тижня та авто-синхронізація зайнятих слотів
YouCanBook.me	Хороша персоналізація посилань, базова інтеграція з календарями	Відсутність автоматичного визначення зайнятих слотів, обмежена інтеграція з Google Meet	Немає автоматичного блокування слотів і глибокої інтеграції з Google Meet
Google Calendar	Безкоштовний, повна синхронізація подій, знайомий інтерфейс	Немає готового механізму бронювання для зовнішніх користувачів, ручне узгодження часу	Не підходить для публічного бронювання, немає посилань і автоматизації

8

Вимоги до системи

Функціональні:

- Авторизація та створення ім’я користувача
- Створення та редагування типів подій
- Гнучке налаштування доступності по днях тижня
- Автоматичне визначення та блокування зайнятих часових слотів
- Генерація персональних посилань
- Інтерактивний календар з вибором часу
- Створення бронювання з автоматичним створенням зустічі
- Перегляд забронюваних зустрічей

Нефункціональні:

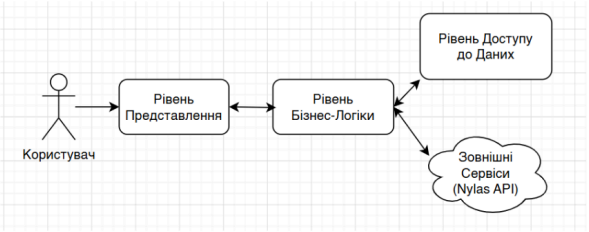
- Висока швидкодія (завантаження сторінок < 3 секунди)
- Адаптивність (коректне відображення на десктопі, планшеті, смартфоні)
- Безпека (захист даних, перевірка власника)
- Зручність та інтуїтивність інтерфейсу
- Масштабованість та надійність роботи

9

Проєктування архітектури системи

Система BKalendar побудована за трирівневою архітектурою (Three-Tier Architecture), що забезпечує чіткий розподіл відповідальності, масштабованість та зручність підтримки. Основні компоненти:

- Рівень представлення (Presentation Layer).
- Рівень бізнес-логіки (Business Logic Layer).
- Рівень доступу до даних (Data Access Layer).
- Зовнішні сервіси. (Nylas API)



10

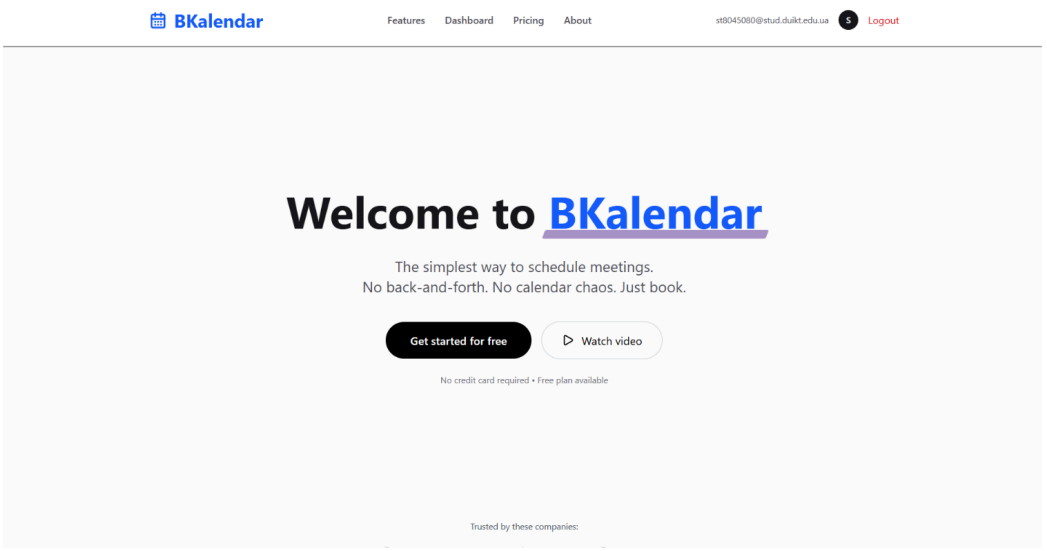
Проєктування алгоритму роботи системи

Алгоритм роботи системи BKalendar спрацьовує при переході за персональним посиланням та завершується після створення бронювання. Основний алгоритм бронювання включає:

- ➔ Вибір дати та часу в інтерактивному календарі
- ➔ Автоматичну перевірку зайнятості слотів через Nylas
- ➔ Введення даних гостя
- ➔ Створення події в Google Calendar з автоматичним генеруванням Google Meet
- ➔ Збереження бронювання в базі даних

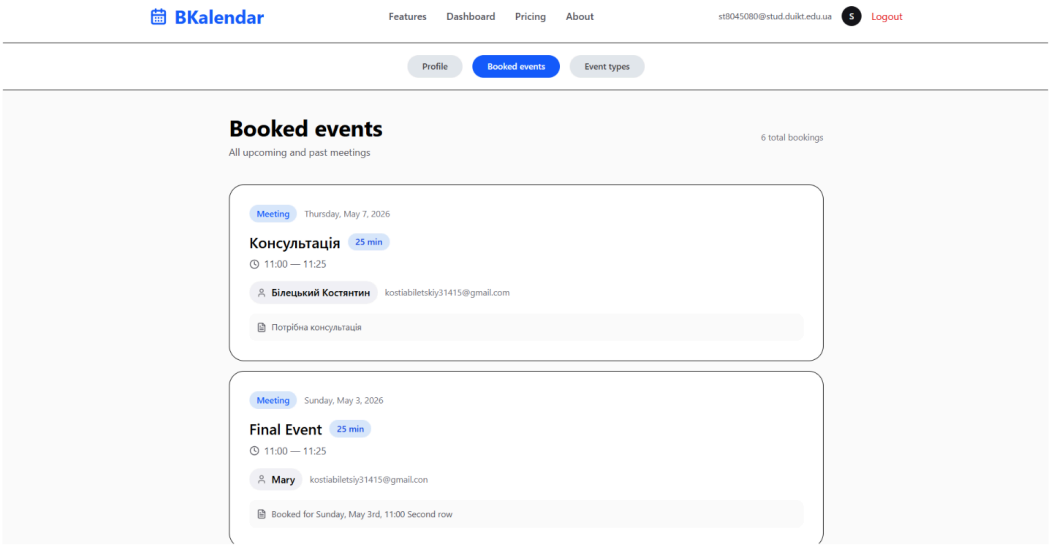
11

Програмна реалізація. Інтерфейс - Landing



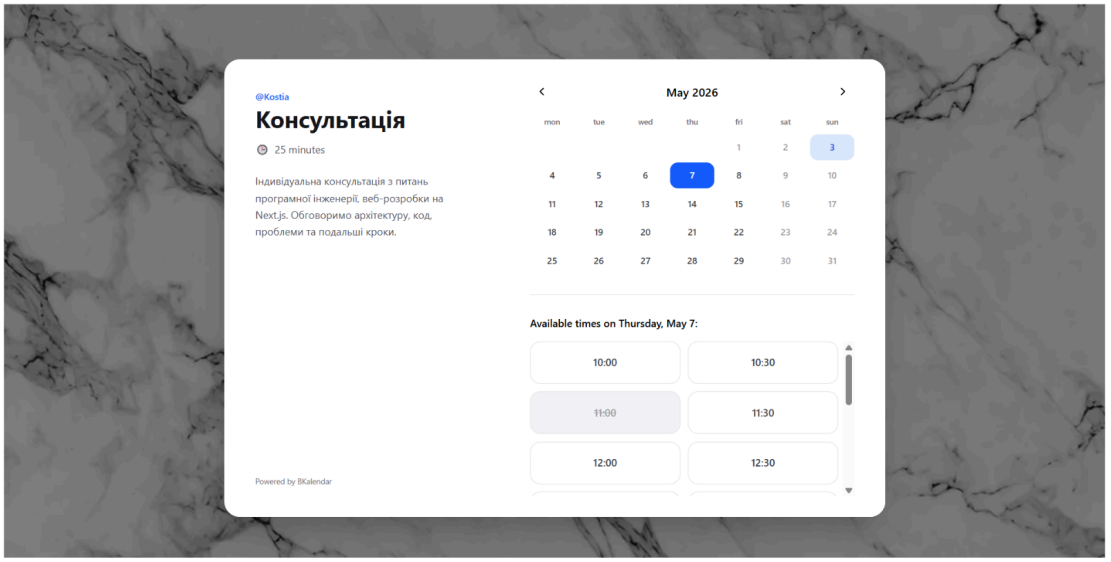
12

Програмна реалізація. Інтерфейс - Dashboard



13

Програмна реалізація. Інтерфейс - Booking (Guest)



14

Програмна реалізація. Сервіс

Серверна частина BKalendar повністю реалізована за допомогою API Routes у Next.js 16. Вона відповідає за всю бізнес-логіку, обробку запитів, взаємодію з базою даних та інтеграцію з платформою Nylas.

Основні модулі сервісу:

1. Авторизація користувачів та управління сесіями (/api/auth, /api/oauth/exchange)
2. Робота з профілем користувача (/api/profile)
3. CRUD-операції з типами подій (/api/event-types)
4. Перевірка зайнятих слотів (/api/busy)
5. Створення бронювання з автоматичним створенням Google Meet (/api/bookings)

15

Тестування системи

Для перевірки працездатності системи було проведено комплексне функціональне тестування на емуляторі та реальному пристрої. Перевірено основні сценарії використання: авторизацію, створення та редагування типів подій, бронювання зустрічей, блокування зайнятих слотів та роботу інтерактивного календаря.

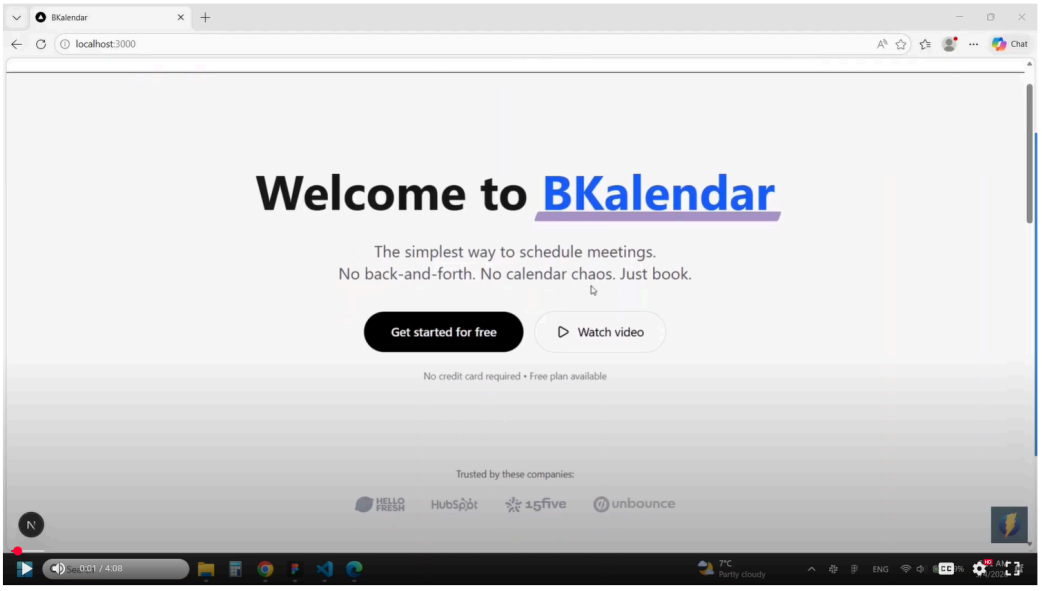
Результати тестування наведено в таблиці:

Всі ключові тест-кейси пройдено успішно. Система демонструє стабільну роботу.

Назва тест-кейсу	Очікуваний результат	Фактичний результат	Статус
Авторизація через Google	Успішний редирект у кабінет	Успішний редирект	Пройдено
Створення типу події	Тип події з'являється у списку	Тип події створено	Пройдено
Редагування типу події	Зміни зберігаються та відображаються	Зміни збережено	Пройдено
Видалення типу події	Тип події видаляється зі списку	Тип події видалено	Пройдено
Бронювання зустрічі (як гість)	Зустріч створюється + надсилається посилання	Зустріч створено	Пройдено
Перевірка блокування зайнятих слотів	Зайняті слоти відображаються сірим кольором та не є активними	Слоти заблоковано	Пройдено

16

Демонстрація роботи додатка



17

Апробація результатів дослідження

Апробація результатів та публікацій:

У Основні положення і результати бакалаврської роботи доповідались на наступних науково-практичних конференціях, що проходили на базі Державного університету інформаційно-комунікаційних технологій:

1. Білецький К.С., Герцюк М.М. Визначення методу планування та резервування зустрічей у цифровій платформі на базі Next.js. // Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». 20.04.2026
2. Білецький К.С., Герцюк М.М. Інтеграція Nylas API для синхронізації календарів та автоматизації резервування зустрічей у цифровій платформі на базі Next.js. // Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація». 25.04.2026

Подальші дослідження можуть бути спрямовані на розширення функціоналу: командні зустрічі, аналітика бронювань чи інтеграція із Zoom.

Висновки

18

1. Проведено аналіз існуючих систем онлайн-бронювання (Calendly, YouCanBook.me, Google Calendar) та виявлено їх основні переваги та недоліки.
2. Сформовано вимоги до системи та спроектовано трирівневу архітектуру на основі Next.js 16 та Nylas.
3. Розроблено структуру бази даних для зберігання типів подій, профілів користувачів та заброньованих зустрічей.
4. Спроектовано архітектуру програмного забезпечення з використанням трирівневої архітектури (Presentation Layer, Business Logic Layer, Data Access Layer).
5. Реалізовано функціонал створення типів подій та гнучкого налаштування доступності по днях тижня.
6. Розроблено механізм автоматичного визначення та блокування зайнятих слотів з інтеграцією до календаря користувача через платформу Nylas.
7. Створено інтерактивний календар з вибором часу та автоматичним блокуванням зайнятих слотів.
8. Реалізовано систему профілю користувача з можливістю встановлення унікального імені (username) та генерації персональних посилань для бронювання.
9. Проведено функціональне та нефункціональне тестування розробленого веб-сервісу, яке підтвердило відповідність системи всім вимогам.
10. Оцінено ефективність системи за показниками продуктивності, безпеки та адаптивності.

Дякую за увагу!

ДОДАТОК Б. КОД ОСНОВНИХ МОДУЛІВ

1. Моделі бази даних

1.1 EventTypes.ts

```
import mongoose, {model, Schema} from 'mongoose';
import { BookingTimes } from '@/libs/types';

const FromToSchema = new mongoose.Schema({
    from: String,
    to: String,
});

type EventType = {
    email: string;
    title: string;
    description: string;
    length: number;
    bookingTimes: BookingTimes;
}

const BookingSchema = new mongoose.Schema({
        monday: FromToSchema,
        tuesday: FromToSchema,
        wednesday: FromToSchema,
        thursday: FromToSchema,
        friday: FromToSchema,
        saturday: FromToSchema,
        sunday: FromToSchema,
});

const EventTypeSchema = new Schema({
    email: String,
    title: String,
    description: String,
    length: Number,
    bookingTimes: BookingSchema,
```

```
  uri: { type: String, unique: true },
}, {
  timestamps: true,
})

export const EventTypeModel = mongoose.models?.EventType || mongoose.model<EventType>('EventType',
EventTypeSchema);
```

1.2 Profile.ts

```
import mongoose, { Schema } from 'mongoose';

type Profile = {
 email: string;
 username: string;
 grantId?: string;
};

const ProfileSchema = new Schema({
 email: { type: String, required: true, unique: true },
 username: { type: String, required: true, unique: true },
 grantId: String,
}, {
 timestamps: true,
});

export const ProfileModel = mongoose.models?.Profile || mongoose.model<Profile>('Profile', ProfileSchema);
```

1.3 Booking.ts

```
import mongoose, { model, Schema, Types } from 'mongoose';

interface IBooking extends mongoose.Document {
 guestName: string;
 guestEmail: string;
 guestNotes?: string;
 when: Date;
 eventTypeId: Types.ObjectId;
 eventTitle: string;
 eventLength: number;
 username: string;
 bookingUri: string;
```

```
  meetingLink?: string;
}

const BookingSchema = new Schema<IBooking>({
  guestName: { type: String, required: true },
  guestEmail: { type: String, required: true },
  guestNotes: { type: String },
  when: { type: Date, required: true },
  eventTypeId: { type: Schema.Types.ObjectId, ref: 'EventType', required: true },
  eventTitle: { type: String, required: true },
  eventLength: { type: Number, required: true },
  username: { type: String, required: true },
  bookingUri: { type: String, required: true },
  meetingLink: { type: String },
}, {
  timestamps: true,
});

export const BookingModel = mongoose.models?.Booking || model<IBooking>('Booking', BookingSchema);
```

2. Основні клієнтські компоненти

2.1 EventTypeForm.tsx

```
'use client';
import { useState, useEffect } from "react";
import { useRouter } from "next/navigation";
import { BookingTimes, WeekdayName } from "@/libs/types";

const weekdays: WeekdayName[] = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday'];

type FormData = {
  title: string;
  description: string;
  length: number;
  bookingTimes: BookingTimes;
  activeDays: Record<WeekdayName, boolean>;
};
```

```
type EventTypeFormProps = {
  email: string;
  doc?: any;
  isEdit?: boolean;
};

export default function EventTypeForm({ email, doc, isEdit = false }: EventTypeFormProps) {
  const router = useRouter();

  const [formData, setFormData] = useState<FormData>(() => {
    const defaultBookingTimes: BookingTimes = {
      monday: { from: "09:00", to: "17:00" },
      tuesday: { from: "09:00", to: "17:00" },
      wednesday: { from: "09:00", to: "17:00" },
      thursday: { from: "09:00", to: "17:00" },
      friday: { from: "09:00", to: "17:00" },
      saturday: { from: "10:00", to: "14:00" },
      sunday: { from: "10:00", to: "14:00" },
    };

    if (doc?.bookingTimes) {
      return {
        title: doc.title || "",
        description: doc.description || "",
        length: doc.length || 30,
        bookingTimes: { ...defaultBookingTimes, ...doc.bookingTimes },
        activeDays: {
          monday: !!doc.bookingTimes.monday,
          tuesday: !!doc.bookingTimes.tuesday,
          wednesday: !!doc.bookingTimes.wednesday,
          thursday: !!doc.bookingTimes.thursday,
          friday: !!doc.bookingTimes.friday,
          saturday: !!doc.bookingTimes.saturday,
          sunday: !!doc.bookingTimes.sunday,
        },
      };
    }

    return {
      title: "",
      description: "",
      length: 30,
```

```
    bookingTimes: defaultBookingTimes,
    activeDays: {
      monday: true, tuesday: true, wednesday: true, thursday: true,
      friday: true, saturday: true, sunday: true,
    },
  };
});

const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();

  const activeBookingTimes: Partial<BookingTimes> = {};
  Object.keys(formData.bookingTimes).forEach((day) => {
    const weekday = day as WeekdayName;
    if (formData.activeDays[weekday]) {
      activeBookingTimes[weekday] = formData.bookingTimes[weekday];
    }
  });


  const generatedUri = formData.title
    .toLowerCase()
    .trim()
    .replace(/[^a-z0-9]+/g, '-')
    .replace(/-+/g, '-')
    .replace(/^-|-$/g, '') || 'event';

  const payload = {
    email,
    title: formData.title,
    description: formData.description,
    length: formData.length,
    bookingTimes: activeBookingTimes,
    uri: generatedUri,
  };

  try {
    const url = isEdit && doc?._id
      ? `/api/event-types/${doc._id}`
      : '/api/event-types';

    const method = isEdit && doc?._id ? 'PUT' : 'POST';
```

```
    const res = await fetch(url, {
      method,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload),
    });

    const result = await res.json();

    if (result.success || result._id) {
      alert(isEdit ? "✅ Updated successfully!" : "✅ Saved successfully!");
      router.push('/dashboard/event-types');
      router.refresh();
    } else {
      alert("❌ Failed to save");
    }
  } catch (err) {
    console.error(err);
    alert("❌ Network error");
  }
};

const toggleDay = (day: WeekdayName) => {
  setFormData(prev => ({
    ...prev,
    activeDays: { ...prev.activeDays, [day]: !prev.activeDays[day] },
  }));
};

const updateTime = (day: WeekdayName, field: 'from' | 'to', value: string) => {
  setFormData(prev => ({
    ...prev,
    bookingTimes: {
      ...prev.bookingTimes,
      [day]: { ...prev.bookingTimes[day], [field]: value },
    },
  }));
};

const handleCancel = () => router.push('/dashboard/event-types');

return (
```

```
<div className="bg-zinc-100 rounded-3xl p-8 max-w-4xl mx-auto">
 <h2 className="text-2xl font-semibold mb-8">
  {isEdit ? "Edit event type" : "Create new event type:"}
 </h2>

 <form onSubmit={handleSubmit} className="space-y-8">
  <div className="grid grid-cols-2 gap-8">
   <div>
    <label className="block text-sm font-medium text-zinc-600 mb-2">TITLE</label>
    <input
     type="text"
     value={formData.title}
     onChange={(e) => setFormData({ ...formData, title: e.target.value })}
     className="w-full p-3 rounded-xl border border-zinc-300 focus:outline-none focus:border-blue-600"
     required
    />
   </div>

   <div>
    <label className="block text-sm font-medium text-zinc-600 mb-2">EVENT LENGTH (MINUTES)</label>
    <input
     type="number"
     value={formData.length}
     onChange={(e) => setFormData({ ...formData, length: parseInt(e.target.value) || 30 })}
     className="w-full p-3 rounded-xl border border-zinc-300 focus:outline-none focus:border-blue-600"
     required
    />
   </div>
  </div>

  <div>
   <label className="block text-sm font-medium text-zinc-600 mb-2">DESCRIPTION</label>
   <textarea
    value={formData.description}
    onChange={(e) => setFormData({ ...formData, description: e.target.value })}
    className="w-full p-3 rounded-xl border border-zinc-300 focus:outline-none focus:border-blue-600 h-24 resize-y"
   />
  </div>

  <div>
   <label className="block text-sm font-medium text-zinc-600 mb-4">AVAILABILITY</label>
   <div className="space-y-4">
```

```
          {weekdays.map((day) => (
           <div key={day} className="flex items-center gap-4">
            <label className="flex items-center gap-2 w-32 cursor-pointer">
             <input
              type="checkbox"
              checked={formData.activeDays[day]}
              onChange={() => toggleDay(day)}
              className="w-4 h-4 accent-blue-600"
             />
             <span className="capitalize font-medium text-zinc-700">{day}</span>
            </label>

            <div className="flex items-center gap-3 flex-1">
             <input
              type="time"
              value={formData.bookingTimes[day]?.from || "09:00"}
              onChange={(e) => updateTime(day, 'from', e.target.value)}
              disabled={!formData.activeDays[day]}
              className="p-2 border border-zinc-300 rounded-lg disabled:opacity-50 disabled:cursor-not-allowed"
             />
             <span className="text-zinc-400">-</span>
             <input
              type="time"
              value={formData.bookingTimes[day]?.to || "17:00"}
              onChange={(e) => updateTime(day, 'to', e.target.value)}
              disabled={!formData.activeDays[day]}
              className="p-2 border border-zinc-300 rounded-lg disabled:opacity-50 disabled:cursor-not-allowed"
             />
            </div>
           </div>
          ))}
         </div>
        </div>

        <div className="flex justify-center gap-4 pt-6">
         <button
          type="button"
          onClick={handleCancel}
          className="px-8 py-3 border border-zinc-400 text-zinc-700 rounded-full font-medium hover:bg-zinc-100
transition-colors"
         >
          Cancel
```

```
        </button>

        <button
          type="submit"
          className="bg-blue-600 hover:bg-blue-700 text-white px-10 py-3 rounded-full font-medium transition-colors"
        >
          {isEdit ? "Update" : "Save"}
        </button>
      </div>
    </form>
  </div>
 );
}
```

2.2 TimePicker.tsx

```
'use client';
import { useState, useEffect } from "react";
import { ChevronLeft, ChevronRight } from "lucide-react";
import { format, startOfMonth, endOfMonth, eachDayOfInterval, addMonths, subMonths, isToday, isSameDay, isPast,
startOfDay, endOfDay, addMinutes, isAfter, isBefore, isEqual } from "date-fns";
import { BookingTimes, WeekdayName } from "@/libs/types";
import { weekdaysNames, weekdaysShortNames } from "@/libs/shared";
import Link from "next/link";
import axios from "axios";
import PulseLoader from "react-spinners/PulseLoader";

type TimePickerProps = {
 bookingTimes: BookingTimes;
 username: string;
 bookingUri: string;
 length: number;
};

type BusySlot = {
 startTime: number;
 endTime: number;
};

export default function TimePicker({ bookingTimes, username, bookingUri, length }: TimePickerProps) {
 const [currentDate, setCurrentDate] = useState(new Date());
 const [selectedDate, setSelectedDate] = useState<Date | null>(null);
```

```
const [busySlots, setBusySlots] = useState<BusySlot[]>([]);
const [busySlotsLoaded, setBusySlotsLoaded] = useState(false);

const monthStart = startOfMonth(currentDate);
const monthEnd = endOfMonth(currentDate);
const daysInMonth = eachDayOfInterval({ start: monthStart, end: monthEnd });

const startDayOffset = monthStart.getDay() === 0 ? 6 : monthStart.getDay() - 1;

const getWeekdayName = (date: Date): WeekdayName => {
  const dayOfWeek = date.getDay() === 0 ? 6 : date.getDay() - 1;
  return weekdaysNames[dayOfWeek];
};

const selectedDayName = selectedDate ? getWeekdayName(selectedDate) : null;
const currentBooking = selectedDayName ? bookingTimes[selectedDayName] : null;

useEffect(() => {
  if (selectedDate) {
    setBusySlots([]);
    setBusySlotsLoaded(false);

    const params = new URLSearchParams();
    params.set('username', username);
    params.set('from', startOfDay(selectedDate).toISOString());
    params.set('to', endOfDay(selectedDate).toISOString());

    axios
      .get(`/api/busy?${params.toString()}`)
      .then((response) => {
        setBusySlots(response.data);
        setBusySlotsLoaded(true);
      })
      .catch((err) => {
        console.error("Failed to load busy slots", err);
        setBusySlotsLoaded(true);
      });
  }
}, [selectedDate, username]);

const generateTimeSlots = (from: string, to: string, duration: number) => {
  const slots: string[] = [];
```

```
  let [hour, minute] = from.split(':').map(Number);
  const [endHour, endMinute] = to.split(':').map(Number);

  while (true) {
    const slotEndHour = hour + Math.floor((minute + duration) / 60);
    const slotEndMinute = (minute + duration) % 60;

    if (slotEndHour > endHour || (slotEndHour === endHour && slotEndMinute > endMinute)) {
      break;
    }

    slots.push(`${hour.toString().padStart(2, '0')}:${minute.toString().padStart(2, '0')}`);

    minute += 30;
    if (minute >= 60) {
      minute = 0;
      hour++;
    }
    if (hour >= 24) break;
  }
  return slots;
};

const timeSlots = currentBooking
  ? generateTimeSlots(currentBooking.from, currentBooking.to, length)
  : [];

const isSlotBusy = (time: string) => {
  if (!selectedDate || !busySlotsLoaded) return false;

  const [hour, minute] = time.split(':').map(Number);
  const slotStart = new Date(selectedDate);
  slotStart.setHours(hour, minute, 0, 0);

  const slotEnd = addMinutes(slotStart, length);

  return busySlots.some((busy) => {
    const busyFrom = new Date(busy.startTime);
    const busyTo = new Date(busy.endTime);

    return (
      (isAfter(slotStart, busyFrom) && isBefore(slotStart, busyTo)) ||
```

```
      (isAfter(slotEnd, busyFrom) && isBefore(slotEnd, busyTo)) ||
      isEqual(slotStart, busyFrom) ||
      isEqual(slotEnd, busyTo)
    );
  });
};

return (
  <div className="space-y-8">
    <div className="flex items-center justify-between mb-4">
      <button
        onClick={() => setCurrentDate(subMonths(currentDate, 1))}
        className="p-2 hover:bg-zinc-100 rounded-lg transition-colors"
      >
        <ChevronLeft size={20} />
      </button>
      <h3 className="font-semibold text-lg">{format(currentDate, 'MMMM yyyy')}</h3>
      <button
        onClick={() => setCurrentDate(addMonths(currentDate, 1))}
        className="p-2 hover:bg-zinc-100 rounded-lg transition-colors"
      >
        <ChevronRight size={20} />
      </button>
    </div>

    <div className="grid grid-cols-7 gap-1 mb-2">
      {weekdaysShortNames.map((day, i) => (
        <div key={i} className="text-center text-xs font-medium text-zinc-500 py-1">
          {day}
        </div>
      ))}
    </div>

    <div className="grid grid-cols-7 gap-1">
      {Array.from({ length: startDayOffset }).map((_, i) => (
        <div key={`empty-${i}`} className="h-10" />
      ))}

      {daysInMonth.map((day, i) => {
        const isSelected = selectedDate ? isSameDay(day, selectedDate) : false;
        const isCurrentDay = isToday(day);
        const weekdayName = getWeekdayName(day);
```

```
    const isPastDay = isPast(startOfDay(day)) && !isToday(day);
    const hasAvailability = !!bookingTimes[weekdayName] && !isPastDay;

    return (
     <button
      key={i}
      onClick={() => hasAvailability && setSelectedDate(day)}
      disabled={!hasAvailability}
      className={`h-10 flex items-center justify-center rounded-xl text-sm font-medium transition-all
       ${isSelected
         ? 'bg-blue-600 text-white'
         : isCurrentDay
          ? 'bg-blue-100 text-blue-700 font-semibold'
          : hasAvailability
           ? 'hover:bg-zinc-100 text-zinc-700'
           : 'text-zinc-400 cursor-not-allowed'
       }`}
     >
      {format(day, 'd')}
     </button>
    );
   })}
  </div>

  <div className="mt-8 border-t border-zinc-200 pt-8">
   {selectedDate ? (
    <>
     <h3 className="font-semibold mb-4">
      Available times on {format(selectedDate, 'EEEE, MMMM d')}:
     </h3>

     {!busySlotsLoaded ? (
      <div className="flex justify-center py-12">
       <PulseLoader
        color="#3B82F6"
        size={12}
        speedMultiplier={0.8}
       />
      </div>
     ) : timeSlots.length > 0 ? (
      <div className="max-h-60 overflow-y-auto pr-2 grid grid-cols-2 gap-3">
       {timeSlots.map((time) => {
```

```
            const busy = isSlotBusy(time);
            return (
              <Link
                key={time}
                href={busy ? "#" : `/${username}/${bookingUri}/${format(selectedDate!, 'yyyy-MM-dd')}T${time}`}
                onClick={(e) => busy && e.preventDefault()}
                className={`border rounded-2xl py-5 text-base font-medium transition-all active:scale-95 block text-center
                  ${busy
                    ? 'border-zinc-200 bg-zinc-100 text-zinc-400 cursor-not-allowed line-through'
                    : 'border-zinc-200 hover:border-blue-600 hover:bg-blue-50 text-zinc-700'
                  }`}
              >
                {time}
              </Link>
            );
          })}
        </div>
      ) : (
        <p className="text-zinc-500 py-8 text-center">No availability on this day</p>
      )}
    </>
  ) : (
    <p className="text-zinc-500 py-8 text-center">Select a day to see available times</p>
  )}
 </div>
</div>
);
}
```

2.3 BookingForm.tsx

```
'use client';
import { useState, useEffect } from 'react';
import { format } from 'date-fns';
import axios from 'axios';
import { useRouter } from 'next/navigation';
import { Check } from 'lucide-react';
import Link from 'next/link';

interface BookingFormProps {
 username: string;
 bookingUri: string;
```

```
  bookingTime: Date;
  eventType: any;
}

export default function BookingForm({ username, bookingUri, bookingTime, eventType }: BookingFormProps) {
  const [guestName, setGuestName] = useState('');
  const [guestEmail, setGuestEmail] = useState('');
  const [guestNotes, setGuestNotes] = useState('');
  const [confirmed, setConfirmed] = useState(false);
  const [loading, setLoading] = useState(false);
  const router = useRouter();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!guestName || !guestEmail) return;

    setLoading(true);

    try {
      await axios.post('/api/bookings', {
        guestName,
        guestEmail,
        guestNotes,
        when: bookingTime.toISOString(),
        username,
        bookingUri,
      });
      setConfirmed(true);
    } catch (err) {
      console.error(err);
      alert('Failed to create booking. Please try again.');
    } finally {
      setLoading(false);
    }
  };

  useEffect(() => {
    if (confirmed) {
      const timer = setTimeout(() => {
        router.push('/dashboard/booked-events');
      }, 5000);
```

```
    return () => clearTimeout(timer);
  }
}, [confirmed, router]);

return (
  <>
    <h2 className="text-2xl font-semibold mb-8">
      {format(bookingTime, 'EEEE, MMMM d, HH:mm')}
    </h2>

    {!confirmed ? (
      <form onSubmit={handleSubmit} className="space-y-6">
        <div>
          <label className="block text-sm font-medium mb-2">YOUR NAME</label>
          <input
            type="text"
            value={guestName}
            onChange={(e) => setGuestName(e.target.value)}
            required
            className="w-full border border-zinc-300 rounded-xl px-4 py-3 focus:outline-none focus:border-blue-500"
            placeholder="John Doe"
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-2">YOUR EMAIL</label>
          <input
            type="email"
            value={guestEmail}
            onChange={(e) => setGuestEmail(e.target.value)}
            required
            className="w-full border border-zinc-300 rounded-xl px-4 py-3 focus:outline-none focus:border-blue-500"
            placeholder="you@example.com"
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-2">ANY ADDITIONAL INFO? (optional)</label>
          <textarea
            value={guestNotes}
            onChange={(e) => setGuestNotes(e.target.value)}
            className="w-full border border-zinc-300 rounded-xl px-4 py-3 h-28 focus:outline-none focus:border-blue-500"
```

```
            placeholder="Any relevant information..."
          />
        </div>

        <button
          type="submit"
          disabled={loading}
          className="w-full bg-blue-600 hover:bg-blue-700 text-white py-4 rounded-2xl font-medium transition-colors
disabled:opacity-50"
        >
          {loading ? 'Confirming...' : 'Confirm booking'}
        </button>
      </form>
    ) : (
      <div className="text-center py-16">
        <div className="flex justify-center mb-6">
          <div className="w-20 h-20 bg-green-100 rounded-full flex items-center justify-center">
            <Check className="w-12 h-12 text-green-600" />
          </div>
        </div>

        <h3 className="text-3xl font-bold mb-3">Booking Confirmed!</h3>
        <p className="text-zinc-600 mb-8">You will receive a confirmation email shortly.</p>

        <div className="space-y-4">
          <Link
            href="/dashboard/booked-events"
            className="inline-block bg-blue-600 hover:bg-blue-700 text-white px-8 py-3 rounded-full font-medium
transition-colors"
          >
            Go to Booked Events
          </Link>

          <p className="text-sm text-zinc-500">
            Redirecting automatically in 5 seconds...
          </p>
        </div>
      </div>
    )}
  </>
 );
}
```

3. Основні API Routes

3.1 bookings/route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import mongoose from 'mongoose';
import { BookingModel } from '@/models/Booking';
import { EventTypeModel } from '@/models/EventTypes';
import { ProfileModel } from '@/models/Profile';
import { nylas } from '@/libs/nylas';
import { addMinutes } from 'date-fns';

export async function POST(req: NextRequest) {
 try {
   const { guestName, guestEmail, guestNotes, when, username, bookingUri } = await req.json();

   if (!guestName || !guestEmail || !when || !username || !bookingUri) {
     return NextResponse.json({ error: 'Missing required fields' }, { status: 400 });
   }

   const { setServers } = await import("node:dns/promises");
   setServers(["1.1.1.1", "8.8.8.8"]);

   await mongoose.connect(process.env.MONGODB_URI!, {
     serverSelectionTimeoutMS: 8000
   });

   const profileDoc = await ProfileModel.findOne({ username });
   if (!profileDoc?.grantId) {
     return NextResponse.json({ error: 'No calendar connected' }, { status: 400 });
   }

   const etDoc = await EventTypeModel.findOne({
     email: profileDoc.email,
     uri: bookingUri
   });

   if (!etDoc) {
     return NextResponse.json({ error: 'Event type not found' }, { status: 404 });
   }

   const startDate = new Date(when);
```

```
const endDate = addMinutes(startDate, etDoc.length || 30);

const nylasResponse = await nylas.events.create({
  identifier: profileDoc.grantId,
  requestBody: {
    title: etDoc.title,
    description: etDoc.description || '',
    when: {
      startTime: Math.round(startDate.getTime() / 1000),
      endTime: Math.round(endDate.getTime() / 1000),
    },
    conferencing: {
      autocreate: {},
      provider: 'Google Meet',
    },
    participants: [
      {
        name: guestName,
        email: guestEmail,
        status: 'yes',
      },
    ],
  },
  queryParams: {
    calendarId: 'primary',
  },
});

const conferencing = (nylasResponse.data as any)?.conferencing || {};
const meetingLink =
  conferencing?.details?.url ||
  conferencing?.meetingUrl ||
  conferencing?.url ||
  (conferencing?.details && typeof conferencing.details === 'string' ? conferencing.details : null) ||
  null;

const booking = await BookingModel.create({
  guestName,
  guestEmail,
  guestNotes: guestNotes || '',
  when: new Date(when),
  eventTypeId: etDoc._id,
```

```
      eventTitle: etDoc.title,
      eventLength: etDoc.length || 30,
      username,
      bookingUri,
      meetingLink,
    });

    console.log(`✅ Booking created with link: ${meetingLink}`);

    return NextResponse.json({
      success: true,
      bookingId: booking._id,
      meetingLink
    });

  } catch (error: any) {
    console.error('Booking creation error:', error);
    return NextResponse.json({
      error: error.message || 'Failed to create booking'
    }, { status: 500 });
  }
}
```

3.2 event-types/route.ts

```
import mongoose from "mongoose";
import { NextRequest } from "next/server";
import { EventTypeModel } from "@/models/EventTypes";

export async function POST(req: NextRequest) {
  try {
    const { setServers } = await import("node:dns/promises");
    setServers(["1.1.1.1", "8.8.8.8"]);

    await mongoose.connect(process.env.MONGODB_URI!);

    const data = await req.json();

    if (!data.email || !data.title) {
       return Response.json({ success: false, error: "Missing required fields" }, { status: 400 });
    }
```

```
    const newEventType = await EventTypeModel.create(data);

    console.log("✅ Saved to DB:", newEventType);

    return Response.json({
      success: true,
      message: "Event type saved successfully!",
      id: newEventType._id
    });

  } catch (error: any) {
    console.error("❌ Error in POST:", error);
    return Response.json({
      success: false,
      error: error.message || "Failed to save"
    }, { status: 500 });
  }
}
```

3.3 event-types/[id]/route.ts

```
import mongoose from "mongoose";
import { NextRequest } from "next/server";
import { EventTypeModel } from "@/models/EventTypes";

export async function PUT(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  try {
    const { id } = await params;
    await mongoose.connect(process.env.MONGODB_URI!);
    const data = await req.json();

    const updated = await EventTypeModel.findByIdAndUpdate(id, data, { new: true });

    if (!updated) return Response.json({ success: false, error: "Not found" }, { status: 404 });

    return Response.json({ success: true, data: updated });
  } catch (error: any) {
    console.error(error);
    return Response.json({ success: false, error: error.message }, { status: 500 });
  }
}
```

```
export async function DELETE(req: NextRequest, { params }: { params: Promise<{ id: string }> }) {
  try {
    const { id } = await params;
    await mongoose.connect(process.env.MONGODB_URI!);

    const deleted = await EventTypeModel.findByIdAndDelete(id);

    if (!deleted) return Response.json({ success: false, error: "Not found" }, { status: 404 });

    console.log("✅ Deleted event type:", id);
    return Response.json({ success: true });
  } catch (error: any) {
    console.error(error);
    return Response.json({ success: false, error: error.message }, { status: 500 });
  }
}
```

3.4 busy/route.ts

```
import { NextRequest, NextResponse } from 'next/server';
import mongoose from 'mongoose';
import { BookingModel } from '@/models/Booking';

export async function GET(req: NextRequest) {
  const { searchParams } = new URL(req.url);
  const username = searchParams.get('username');
  const from = searchParams.get('from');
  const to = searchParams.get('to');

  if (!username || !from || !to) {
    return NextResponse.json({ error: 'Missing parameters' }, { status: 400 });
  }

  try {
    const { setServers } = await import("node:dns/promises");
    setServers(["1.1.1.1", "8.8.8.8"]);

    await mongoose.connect(process.env.MONGODB_URI!, {
      serverSelectionTimeoutMS: 8000
    });

    const bookings = await BookingModel.find({
```

```
      username,
      when: {
        $gte: new Date(from),
        $lte: new Date(to),
      },
    }).lean();

    const busySlots = bookings.map((booking: any) => ({
      startTime: booking.when.getTime(),
      endTime: new Date(booking.when.getTime() + (booking.eventTypeId ? 30 * 60 * 1000 : 0)).getTime(),
    }));

    return NextResponse.json(busySlots);
  } catch (error) {
    console.error('Busy slots error:', error);
    return NextResponse.json({ error: 'Failed to fetch busy slots' }, { status: 500 });
  }
}
```

3.5 profile/route.ts

```
import mongoose from "mongoose";
import { NextRequest } from "next/server";
import { getSession } from "@/libs/session";
import { ProfileModel } from "@/models/Profile";

export async function GET() {
  const { email } = await getSession();
  if (!email) return Response.json({ error: "Not logged in" }, { status: 401 });

  try {
    const { setServers } = await import("node:dns/promises");
    setServers(["1.1.1.1", "8.8.8.8"]);
    await mongoose.connect(process.env.MONGODB_URI!);

    const profile = await ProfileModel.findOne({ email });
    return Response.json({ username: profile?.username || "" });
  } catch (error: any) {
    console.error(error);
    return Response.json({ error: "Failed to load profile" }, { status: 500 });
  }
}
```

```
export async function PUT(req: NextRequest) {
  const { email, grantId } = await getSession();
  if (!email) return Response.json({ error: "Not logged in" }, { status: 401 });

  try {
    const { username } = await req.json();
    if (!username || username.length < 3) {
      return Response.json({ error: "Username must be at least 3 characters" }, { status: 400 });
    }

    const { setServers } = await import("node:dns/promises");
    setServers(["1.1.1.1", "8.8.8.8"]);
    await mongoose.connect(process.env.MONGODB_URI!);

    const updated = await ProfileModel.findOneAndUpdate(
      { email },
      { email, username, grantId },
      { upsert: true, new: true }
    );

    return Response.json({ success: true, username: updated.username });
  } catch (error: any) {
    console.error(error);
    return Response.json({ error: error.message || "Failed to save profile" }, { status: 500 });
  }
}
```

4. Допоміжні модулі

4.1 nylas.ts

```
import Nylas from "nylas";

export const nylasConfig = {
  clientId: process.env.NYLAS_CLIENT_ID!,
  callbackUri: "http://localhost:3000/api/oauth/exchange",
  apiKey: process.env.NYLAS_API_KEY!,
  apiUri: process.env.NYLAS_API_URI!,
};

export const nylas = new Nylas({
```

```
  apiKey: nylasConfig.apiKey,
  apiUri: nylasConfig.apiUri,
});
```

4.2 session.ts

```
import { cookies } from 'next/headers';

export async function setSession(grantId: string, email: string) {
  const cookieStore = await cookies();

  cookieStore.set('grantId', grantId, {
    httpOnly: true,
    secure: process.env.NODE_ENV === 'production',
    sameSite: 'lax',
    maxAge: 60 * 60 * 24 * 7,
  });

  if (email) {
    cookieStore.set('email', email, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'lax',
      maxAge: 60 * 60 * 24 * 7,
    });
  }
}

export async function getSession() {
  const cookieStore = await cookies();
  const grantId = cookieStore.get('grantId')?.value;
  const email = cookieStore.get('email')?.value;

  return { grantId, email };
}
```