МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ФАХОВИЙ БІЗНЕС-КОЛЕДЖ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
(повна назва випускної кафедри)

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему:

ВЕБ-ДОДАТОК ДЛЯ РЕЄСТРАЦІЇ ТА МОНІТОРИНГУ ПОДІЙ

Виконав:
студент групи 1*КІ-23* зі спеціальності
123 – «Комп'ютерна інженерія»
Артем ДАНЬКО
Науковий керівник:
к.т.н Роксолана БРЕУС
(науковий ступінь, вчене звання, прізвище та ініціали)

Черкаси, 2025

АНОТАЦІЯ

У дипломній роботі представлено розробку веб-додатку «Events Track» для моніторингу подій та конференцій. Система створена для вирішення проблеми децентралізованої інформації про події та спрощення процесу їх організації та відвідування.

Веб-додаток реалізовано з використанням сучасного технологічного стеку: React 19 з TypeScript для клієнтської частини та NestJS з PostgreSQL для серверної. Архітектура фронтенду базується на методології Feature-Sliced Design, що забезпечує масштабованість та підтримуваність коду. Для створення користувацького інтерфейсу використано бібліотеку компонентів shadcn/ui з Tailwind CSS.

Система підтримує три основні ролі користувачів: учасники (Participant), спікери (Speaker) та адміністратори (Admin). Кожна роль має відповідний набір функціональних можливостей. Основний функціонал включає: пошук та фільтрацію подій, реєстрацію на заходи, створення та управління подіями, систему сповіщень, експорт даних та адміністративну панель.

Особлива увага приділена безпеці системи: впроваджено JWT автентифікацію, хешування паролів через bcrypt, захист від поширених веб-атак (SQL injection, XSS, CSRF), rate limiting та валідацію всіх вхідних даних.

Проведено комплексне тестування системи, включаючи unit. Покриття коду тестами становить 84%. Навантажувальне тестування підтвердило здатність системи обслуговувати понад 1000 одночасних користувачів.

Розроблений веб-додаток є готовим рішенням для централізованого моніторингу подій, яке може бути впроваджене в освітніх установах, бізнес-середовищі та громадських організаціях.

Ключові слова: ВЕБ-ДОДАТОК, МОНІТОРИНГ ПОДІЙ, REACT, NESTJS, TYPESCRIPT, POSTGRESQL, JWT, SHADCN/UI.

ABSTRACT

This thesis presents the development of the «Events Track» web application for monitoring events and conferences. The system was created to solve the problem of decentralized event information and simplify the process of organizing and attending events.

The web application is implemented using a modern technology stack: React 19 with TypeScript for the client side and NestJS with PostgreSQL for the server side. The frontend architecture is based on the Feature-Sliced Design methodology, which ensures code scalability and maintainability. The shadcn/ui component library with Tailwind CSS was used to create the user interface.

The system supports three main user roles: Participants, Speakers, and Admins. Each role has a corresponding set of functional capabilities. The main functionality includes: event search and filtering, event registration, event creation and management, notification system, data export, and administrative panel.

Special attention was paid to system security: JWT authentication, password hashing via bcrypt, protection against common web attacks (SQL injection, XSS, CSRF), rate limiting, and validation of all input data were implemented.

Comprehensive system testing was conducted, including unit, integration, and E2E tests. Code test coverage is 84%. Load testing confirmed the system's ability to serve over 1000 concurrent users.

The developed web application is a ready-made solution for centralized event monitoring that can be implemented in educational institutions, business environments, and public organizations.

Keywords: WEB APPLICATION, EVENT MONITORING, REACT, NESTJS, TYPESCRIPT, POSTGRESQL, JWT, SHADCN/UI.

ЗМІСТ

ВСТУП

Галузь організації та управління подіями, зокрема конференціями, дедалі більше інтегрує інформаційно-комунікаційні технології з метою автоматизації основних процесів. У контексті збільшення кількості наукових, професійних та публічних заходів актуальним є створення інструментів для централізованої реєстрації учасників, управління розкладом, моніторингу подій та збирання статистичних даних. Веб-додатки, орієнтовані на обслуговування таких задач, дозволяють скоротити витрати на організацію, зменшити кількість помилок та забезпечити своєчасний доступ до інформації для всіх учасників.

Проблема відсутності універсального програмного засобу, адаптованого до специфіки організації різних типів конференцій, потребує комплексного підходу до її вирішення. Більшість існуючих систем мають обмежені можливості кастомізації або орієнтовані на вузькоспеціалізовані заходи. Це ускладнює їхнє використання у широкому діапазоні випадків.

Метою кваліфікаційної роботи є розробка веб-додатку для реєстрації та моніторингу подій, який дозволяє адміністраторам створювати конференції, управляти інформацією про сесії, доповідачів, зареєстрованих учасників, а також здійснювати контроль за активністю у режимі реального часу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- здійснити аналіз функціональних та нефункціональних вимог до системи;
- провести аналіз сучасних аналогів;
- визначити архітектурні особливості та моделі зберігання даних;
- сформувати загальну структуру веб-застосунку;
- розробити інтерфейс користувача;
- реалізувати серверну та клієнтську частину інтерфейсу користувача;

- забезпечити обробку реєстрацій, автентифікацію, авторизацію та моніторинг активностей користувачів;
- провести тестування додатку.

Об'єктом дослідження є процеси автоматизації управління подіями у сфері організації конференцій.

Предметом дослідження є програмне забезпечення для підтримки реєстрації, управління інформацією про події та моніторингу активностей у вигляді веб-додатку.

У процесі дослідження планується використання таких методів:

– системний аналіз для визначення вимог до функціональності веб-застосунку;
– моделювання для побудови логіки та структури системи;
– структурно-логічне проєктування для формалізації архітектури застосунку та взаємодії між компонентами;
– методи веб-програмування для реалізації інтерфейсів та обробки запитів користувачів.

Практичне значення одержаних результатів полягає у можливості застосування запропонованого веб-додатку в діяльності освітніх установ, бізнес-центрів, організаційних комітетів та інших суб’єктів, що займаються проведенням заходів, без необхідності використання сторонніх або комерційних рішень.

Апробація результатів дослідження здійснювалася шляхом їх представлення на XVII Студентській науково-практичній конференції «Тенденції розвитку ІТ-технологій в Україні».

РОЗДІЛ 1 АНАЛІЗ ПОТРЕБ КОРИСТУВАЧІВ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

1.1 Виявлення ключових потреб користувачів

Зростання кількості організованих подій, зокрема конференцій, вимагає вдосконалення систем управління для обробки великої кількості даних, взаємодії з різними категоріями користувачів та забезпечення високої якості сервісу. У цій галузі з'являється потреба в централізованих системах, здатних забезпечити ефективну обробку запитів учасників, організаторів та адміністраторів через інтерактивні веб-інтерфейси [1].

Однією з основних потреб організаторів подій є можливість створення детальної структури заходу, включаючи визначення розкладу, управління сесіями, а також обробку інформації про спікерів та зареєстрованих учасників. Це дозволяє оптимізувати процеси планування та управління заходами, зменшуючи навантаження на адміністрацію та забезпечуючи зручність взаємодії з усіма сторонами. З цією метою доцільно використовувати сучасні веб-технології для створення адаптивних інтерфейсів, що забезпечують доступність і функціональність для різних користувачів незалежно від їх місця перебування.

Адміністратори конференцій потребують інструментів для управління правами доступу, забезпечення безпеки, ведення журналу подій та обробки реєстраційних даних. Це дає змогу контролювати процеси реєстрації, уникати дублювання даних і здійснювати управління участю в заході. Також для адміністраторів необхідна можливість генерувати аналітичні звіти про активність учасників та результативність заходу в реальному часі [2].

Також такі системи потребують механізм для обробки форм реєстрації, зберігання персональних даних, а також для надсилання підтверджень, тобто, реєстрації. Користувачі також потребують можливості редагувати свої профілі

та скасовувати участь у разі необхідності. Система повинна забезпечити високу доступність та безпеку персональних даних.

Учасники заходів, у свою чергу, мають потребу в отриманні актуальної інформації про розклад подій, можливість переглядати деталі доповідей і презентацій, а також бути поінформованими про зміни в програмі. Це можна реалізувати через мобільні та веб-інтерфейси, що дозволяють швидко оновлювати інформацію та забезпечувати доступ до неї в будь-який час [3].

Доповідачі також мають специфічні вимоги, пов'язані з подачею матеріалів та взаємодією з адміністрацією заходу. Вони потребують каналу для завантаження презентацій, внесення коригувань і підтвердження участі. Для цього необхідна система автентифікації, що забезпечує безпечний доступ та відповідні права для кожного спікера.

Ще однією потребою є забезпечення масштабованості системи. Збільшення кількості користувачів та запитів вимагає від застосунку здатності обробляти множинні запити без зниження продуктивності. Для цього необхідно використовувати асинхронну обробку запитів та оптимізувати доступ до бази даних для зменшення затримок.

Система повинна бути здатною до розширення без значних змін у структурі. Модульний підхід дозволяє додавати нові функції та інтегрувати додаткові сервіси, зокрема для роботи з зовнішніми платформами для трансляцій або збереження відеозаписів після завершення сесій.

Для забезпечення високої надійності роботи також необхідно передбачити резервне копіювання даних та підтримку відновлення після збоїв у роботі сервера або мережі [4].

Системи реєстрації та моніторингу подій повинні забезпечувати захист даних користувачів, зокрема шляхом використання шифрування при передачі інформації. Для запобігання несанкціонованому доступу та підтримки конфіденційності даних необхідно регулярно проводити перевірки на вразливості.

Потреба в системах реєстрації та моніторингу подій значно зросла під час пандемії COVID-19, коли організація заходів у віддаленому режимі стала необхідною. Ця потреба залишається актуальною, особливо для України, де системи реєстрації та моніторингу подій використовуються для організації онлайн та гібридних заходів.

Сучасні технології таких систем дозволяють організовувати реєстрацію учасників, управління правами доступу, а також моніторинг активності учасників під час заходів. Це дозволяє забезпечити зручність для організаторів і учасників, а також ефективно управляти великими обсягами даних про реєстрацію та участь у подіях.

Напрямками для розвитку технологій систем реєстрації та моніторингу подій є інтеграція з іншими інструментами для управління заходами, автоматизація процесів реєстрації та обробки даних, а також використання новітніх технологій для покращення користувацького досвіду, таких як штучний інтелект для аналізу даних та оптимізації процесів [5].

Таким чином, потреби користувачів таких систем включають розробку масштабованих, гнучких та інтегрованих рішень для управління реєстрацією, моніторингом подій та обробкою даних. Вибір технологій для таких систем має ґрунтуватися на можливості забезпечення безпеки, масштабованості та адаптивності до змінних вимог користувачів.

1.2 Оцінка та обґрунтування технологічних рішень

Розробка веб-додатка для реєстрації та моніторингу подій вимагає ретельного підходу до вибору технологій. Першим кроком є визначення рішень для фронтенду та бекенду, які забезпечують необхідну функціональність та зручність для користувачів.

Для реалізації фронтенду є кілька варіантів технологій. Одним із найбільш поширених рішень є використання фреймворку React. Цей фреймворк дозволяє створювати динамічні інтерфейси з високою

продуктивністю завдяки віртуальному DOM. React дає змогу організувати компоненти, що спрощує підтримку та розширення додатка. Однією з особливостей є велика кількість бібліотек, що дозволяють розширити функціональність за рахунок готових рішень, таких як роутінг чи інтеграція з API [6].

Іншим можливим варіантом є Vue.js. Цей фреймворк також орієнтований на створення реактивних інтерфейсів, однак має менше вимог до налаштування. Vue має менший поріг для початку розробки та швидше освоюється, що може бути важливим фактором для команд з обмеженим досвідом.

Вибір між React та Vue залежить від вимог до швидкості розробки та рівня складності інтерфейсу. Якщо команда має досвід з React, цей фреймворк буде оптимальним вибором, оскільки він надає більше можливостей для масштабування проекту в майбутньому.

Для розробки бекенду доцільно вибрати Node.js. Ця технологія використовує JavaScript на серверній стороні, що дозволяє працювати з єдиною мовою для фронтенду та бекенду. Node.js має хорошу підтримку для асинхронних операцій, що є корисним для обробки великих обсягів запитів, наприклад, під час реєстрації користувачів або моніторингу їх активності. Для розробки REST API у Node.js можна використовувати популярний фреймворк NestJS(Express), який надає простий механізм для створення маршрутів та обробки запитів [7].

Альтернативним рішенням для бекенду є використання Java з фреймворком Spring Boot. Java є добре відомою мовою для серверних застосунків, і Spring Boot забезпечує готові рішення для створення API, обробки запитів та взаємодії з базами даних. Spring Boot також має вбудовану підтримку для безпеки та аутентифікації, що дозволяє знижувати навантаження на розробників.

Вибір між Node.js та Java з Spring Boot знову залежить від специфіки проекту. Якщо система потребує високу продуктивність при обробці

численних одночасних запитів, Node.js може бути кращим вибором. Java з Spring Boot може бути доцільним для більш складних серверних обчислень і інтеграцій із іншими системами.

Для бази даних, оптимальним вибором є використання MySQL або PostgreSQL. Обидві ці системи є реляційними базами даних і мають високу стабільність і підтримку складних запитів. PostgreSQL підтримує додаткові функції для роботи з великими даними, але MySQL відрізняється простотою налаштування та більш широким використанням у середовищах з високою кількістю запитів. Оскільки проект не передбачає надмірного навантаження на базу даних, PostgreSQL є доцільним вибором, оскільки він простіший у налаштуванні та має добру продуктивність [8].

Для забезпечення безпеки та захисту даних користувачів слід використовувати протоколи шифрування для обміну даними між клієнтом та сервером. Для цього необхідно використовувати HTTPS для захищеного з'єднання, а також методи аутентифікації через токени, такі як JWT (JSON Web Token). Ці технології дозволяють забезпечити надійний захист даних при їх передачі через інтернет.

Потрібно також розглянути питання інтеграції веб-додатка з іншими системами. Одним із можливих напрямків є інтеграція з платформами для відеоконференцій, такими як Zoom або Google Meet, через відповідні API. Це дозволить автоматизувати процес підключення учасників до заходів та моніторинг їх активності під час проведення подій.

Для обробки великих обсягів даних про реєстрації та учасників необхідно реалізувати механізми кешування, які допоможуть зменшити навантаження на базу даних. Redis є популярним вибором для таких завдань, оскільки забезпечує швидкий доступ до часто використовуваних даних та підтримує масштабування системи [9].

Масштабованість системи також має важливе значення, оскільки кількість користувачів може змінюватися в залежності від подій. Використання Docker для контейнеризації компонентів системи дозволяє

забезпечити зручне масштабування та розгортання додатка на різних середовищах. Docker також полегшує управління залежностями та оновленнями.

Задля забезпечення стабільної роботи веб-додатка необхідно також налаштувати автоматичне тестування та CI/CD (неперервну інтеграцію та доставку). Це дозволяє швидко виявляти помилки у коді, знижуючи ймовірність виникнення проблем на етапі експлуатації.

1.3 Комплексний аналіз сучасних аналогів із визначенням їх функціонального призначення

Для аналізу сучасних аналогів веб-додатків, призначених для реєстрації та моніторингу подій, доцільно розглянути три існуючі рішення, такі як Explara, Splash та Bizzabo. Кожне з цих рішень охоплює базову функціональність, необхідну для адміністрування заходів різного масштабу, включаючи реєстрацію учасників, управління подіями та взаємодію з аудиторією.

Explara є платформою для управління подіями, яка реалізує засоби створення заходів, керування учасниками та проведення оплат. Система дозволяє генерувати квитки, формувати списки відвідувачів та інтегрується з декількома платіжними шлюзами. Крім цього, Explara включає функції для маркетингової взаємодії, зокрема надсилання електронних повідомлень та базової аналітики. На рис. 1.1 представлено інтерфейс адміністративної панелі додатка Explara, яка демонструє структуру управління подіями.

Splash є платформою, орієнтованою на організацію заходів з фокусом на візуальну частину оформлення сторінок подій. У цьому додатку реалізовано механізми для створення подій, налаштування реєстраційних форм, здійснення електронних розсилок та аналізу взаємодії користувачів[10].

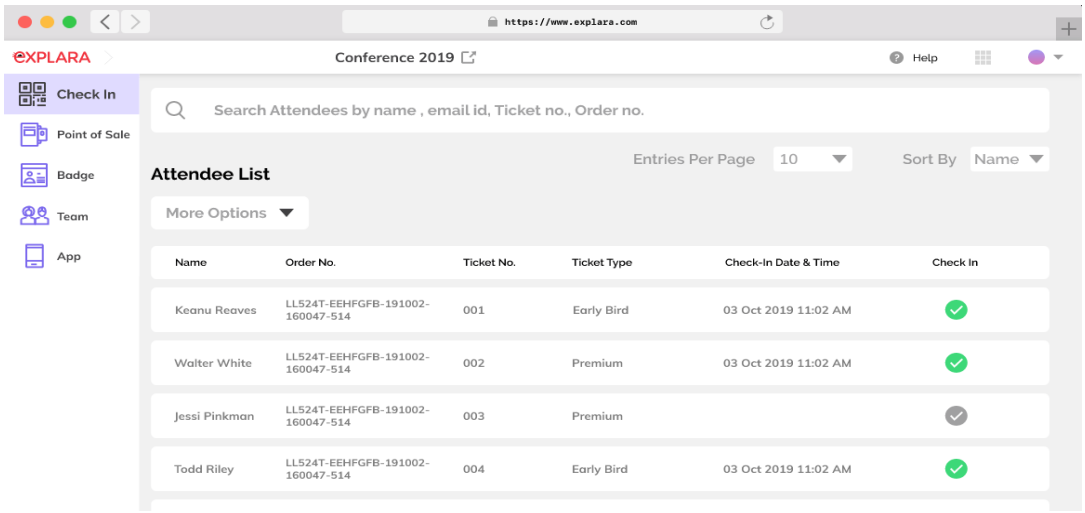

Рисунок 1.1 – Вигляд веб-застосунку Explara

Splash підтримує функції персоналізації подій, дозволяє інтеграцію з іншими системами та надає API для розширення функціоналу. Рис. 1.2 відображає інтерфейс користувача в Splash.

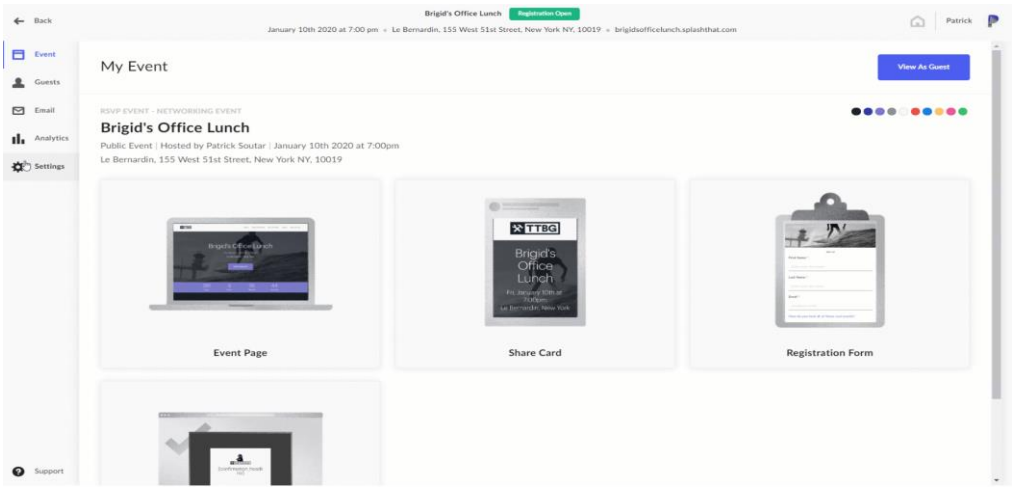

Рисунок 1.2 – Вигляд веб-застосунку Splash

Bizzabo надає повний набір інструментів для створення, моніторингу та аналітики подій. У системі реалізовано реєстрацію користувачів, генерацію квитків, створення індивідуальних сторінок заходів, планування сесій, а також інтеграцію з CRM-системами. Додатково в Bizzabo реалізовані інструменти для спілкування між учасниками, керування розкладом подій, а також

аналітичні панелі для організаторів. На рисунку 1.3 зображено інтерфейс панелі моніторингу подій у Bizzabo [11].

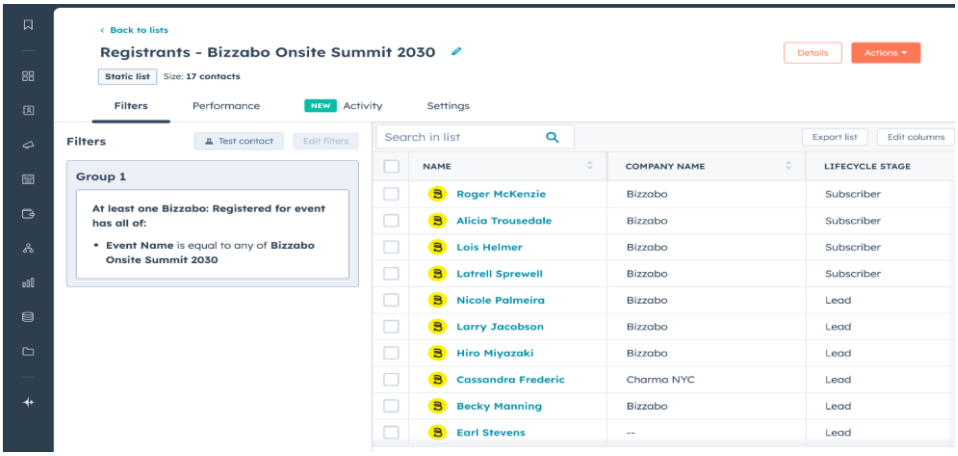

Рисунок 1.3 – Вигляд веб-застосунку Bizzabo

Порівняння характеристик трьох платформ предсавлено в табл.1.1. На основі таблиці визначено аспект, що доцільно реалізувати у власному веб-додатку. Більшість розглянутих рішень не забезпечують внутрішньої комунікації між учасниками або реалізують її частково.

Таблиця 1.1 – Порівняння сучасних аналогів

№ з/п	Критерій	Explara	Splash	Bizzabo
1	Реєстрація користувачів	Так	Так	Так
2	Генерація квитків	Так	Так	Так
3	Обробка оплат	Так	Ні	Так
4	Аналітика	Базова	Обмежена	Розширена
5	Інтеграція з CRM	Обмежена	Часткова	Повна
6	Інтеграція з платіжними системами	Так	Обмежена	Так
7	Мобільна версія	Так	Так	Так
8	Редагування сторінок подій	Обмежене	Розширене	Так
9	Формування розкладу	Так	Так	Так
10	Комунікація між учасниками	Обмежена	Обмежена	Так

У результаті аналізу можна визначити, що всі три рішення мають загальні функціональні блоки, але реалізовані вони з різною глибиною та орієнтацією. Explara більше орієнтована на базове адміністрування та обробку платежів. Splash приділяє увагу дизайну сторінок подій та персоналізації. Bizzabo забезпечує розширене керування подіями з аналітичним супроводом та інструментами комунікації [12].

Усі три системи підтримують інструменти для управління реєстрацією, створення подій та відображення розкладів. Однак рівень автоматизації процесів у Bizzabo значно ширший завдяки інтеграції з зовнішніми платформами та CRM-системами. У свою чергу, Splash має розширений функціонал для візуального налаштування подій, але менш розвинуту систему внутрішньої аналітики. Explara займає проміжну позицію між згаданими системами за рівнем функціональності.

РОЗДІЛ 2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Архітектурна модель програмного забезпечення

Веб-додаток Events Track для моніторингу подій та конференцій побудований на основі сучасної клієнт-серверної архітектури з чітким розділенням відповідальності між компонентами системи. Архітектурне рішення базується на використанні мікросервісного підходу, що забезпечує високу масштабованість, надійність та простоту підтримки системи [13].

Система складається з трьох основних рівнів, кожен з яких виконує свої специфічні функції та має чітко визначені межі відповідальності:

- презентаційний рівень (Frontend) відповідає за візуальне представлення даних та взаємодію з користувачем. Цей рівень реалізований як Single Page Application з використанням React 19 та TypeScript, що забезпечує динамічний користувацький інтерфейс без необхідності перезавантаження сторінки. Використання Feature-Sliced Design архітектури дозволяє організувати код у логічні модулі, що спрощує розробку та підтримку;
- бізнес-логіка (Backend) обробляє всі бізнес-правила та операції системи. Серверна частина побудована на фреймворку NestJS, який надає великі можливості для створення масштабованих Node.js додатків. NestJS використовує модульну архітектуру, що дозволяє легко розширювати функціональність системи та підтримувати чистоту коду;
- рівень даних (Database) забезпечує надійне зберігання та швидкий доступ до інформації. PostgreSQL обрано як основну систему управління базами даних через її надійність, продуктивність та підтримку складних запитів. TypeORM використовується як ORM для спрощення роботи з базою даних та забезпечення типобезпеки на рівні запитів [14].

У сучасній розробці вебзастосунків Frontend-архітектура відіграє ключову роль, оскільки саме вона визначає логіку побудови інтерфейсу, організацію коду та взаємодію з серверною частиною і включає клієнтську

частину системи, яка розроблена з урахуванням найкращих практик сучасної веб-розробки. Основою архітектури є Feature-Sliced Design є методологія, яка пропонує чітку структуру організації коду за функціональними слайсами.

Кожен слайс у FSD представляє окрему функціональну ділянку додатку та містить всі необхідні компоненти для її реалізації. Це включає UI компоненти, бізнес-логіку, API взаємодію та локальний стан. Така організація дозволяє команді розробників працювати над різними функціями паралельно без конфліктів [15].

Для створення користувацького інтерфейсу використовується бібліотека компонентів shadcn/ui. Ця бібліотека надає набір готових, але повністю кастомізованих React компонентів, які копіюються безпосередньо в проект. Такий підхід дає повний контроль над кодом компонентів та їх стилізацією, на відміну від традиційних UI бібліотек.

Управління станом додатку реалізовано через комбінацію двох підходів: React Query (TanStack Query) для серверного стану та Zustand для локального стану додатку. React Query автоматично керує кешуванням, синхронізацією та оновленням даних з сервера, що значно спрощує роботу з асинхронними операціями. Zustand використовується для зберігання локальних налаштувань користувача, таких як тема інтерфейсу, мова та інші преференції.

Зі зростанням складності вебзастосунків та потреб у високій продуктивності й масштабованості, роль Backend-архітектури стає критично важливою для стабільної роботи всієї системи. Серверна частина системи побудована на фреймворку NestJS, який надає досить сильну платформу для створення ефективних та масштабованих серверних додатків на Node.js. NestJS використовує концепції, запозичені з Angular, такі як декоратори, dependency injection та модульна архітектура, що робить код більш структурованим та легким для тестування [16].

Архітектура backend складається з декількох ключових модулів:

- модуль автентифікації відповідає за реєстрацію нових користувачів, вхід в систему та управління сесіями. Використовується JWT

(JSON Web Tokens) для безпечної автентифікації, що дозволяє створювати stateless API. Паролі користувачів зберігаються в хешованому вигляді з використанням bcrypt;

- модуль користувачів керує профілями користувачів та їх ролями. Система підтримує три основні ролі, які зображені на рис. 2.1;

```
export enum UserRolesEnum {
  Admin = 'admin',
  Participant = 'participant',
  Speaker = 'speaker',
}
```

Рисунок 2.1 – Ролі юзерів

- модуль подій є центральним компонентом системи, який відповідає за створення, редагування, видалення та пошук подій. Цей модуль також включає функціональність для управління реєстраціями учасників.
- модуль файлів обробляє завантаження та зберігання файлів, таких як зображення подій, презентації спікерів та інші матеріали. Використовується Multer для обробки multipart/form-data запитів [17].

Вибір бази даних є одним із ключових етапів під час розробки, оскільки від нього залежить ефективність зберігання, обробки та доступу до даних.

PostgreSQL обрано як основну систему управління базами даних через її надійність, продуктивність та широкі можливості. База даних спроектована з урахуванням нормалізації для уникнення дублювання даних та забезпечення цілісності інформації.

TypeORM використовується як Object-Relational Mapping (ORM) інструмент, який спрощує взаємодію з базою даних. TypeORM підтримує міграції, що дозволяє упровадити систему керування версіями схеми бази даних та легко розгортати зміни на різних середовищах.

Основні таблиці бази даних включають:

– users – зберігає інформацію про користувачів системи
– events – містить дані про події та конференції
– registrations – фіксує реєстрації користувачів на події

Взаємодія компонентів забезпечує ефективну координацію роботи клієнтської та серверної частин для досягнення цілісності та надійності системи.

Клієнтська та серверна частини взаємодіють через RESTful API, який забезпечує стандартизований спосіб обміну даними. Всі запити та відповіді використовують JSON формат, що є стандартом для веб-додатків [18].

Для документації API використовується Swagger (OpenAPI) зображений на рис. 2.2, який автоматично генерує інтерактивну документацію на основі декораторів NestJS. Це дозволяє розробникам легко тестувати ендпоінти та розуміти структуру запитів і відповідей.

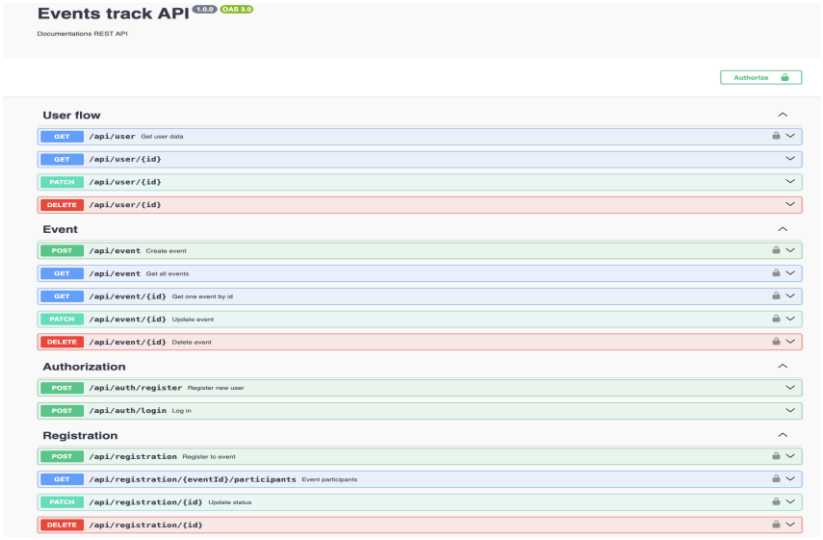

Рисунок 2.2 – Вигляд Swagger документації

Безпека системи забезпечується на декількох рівнях:

– HTTPS для шифрування трафіку
– JWT токени для автентифікації
– CORS налаштування для контролю доступу
– Валідація вхідних даних
– Rate limiting для захисту від DDoS атак

2.2 Ідентифікація ролей користувачів та моделювання сценаріїв їх взаємодії із системою

Ролі користувачів системи визначають права доступу, функціональні можливості та рівень взаємодії з її компонентами.

Система Events Track підтримує три основні ролі користувачів, кожна з яких має свій набір прав та можливостей. Розмежування ролей забезпечує безпеку системи та контроль доступу до різних функцій. Розрізняють наступні види розмежувань доступу [19]:

1. Роль Admin – адміністратори мають найвищий рівень доступу в системі та можуть виконувати всі доступні операції. Їх основні обов'язки включають:

- управління всіма користувачами системи (перегляд, редагування, блокування);
- модерація подій та контенту;
- доступ до системної статистики та аналітики;
- налаштування глобальних параметрів системи;
- управління категоріями та тегами подій;
- перегляд логів системи для моніторингу безпеки;
- призначення ролей іншим користувачам.

Адміністратори відіграють ключову роль у підтримці якості контенту та забезпеченні безперебійної роботи системи. Вони мають доступ до спеціальної адміністративної панелі, де можуть виконувати всі необхідні операції [20].

2. Роль Participant – учасниками є звичайні користувачі системи, які можуть переглядати події та реєструватися на них. Їх можливості включають [21]:

- перегляд всіх публічних подій;
- пошук подій за різними критеріями;
- реєстрація на цікаві події;
- управління власним профілем;

- перегляд історії відвіданих подій;
- отримання сповіщень про нові події;
- збереження подій в обране;
- скасування реєстрацій.

Учасники є основною аудиторією системи, тому інтерфейс оптимізовано для зручного пошуку та реєстрації на події. Вони можуть налаштовувати свої преференції щодо типів подій та отримувати персоналізовані рекомендації [22].

3. Роль Speaker має розширені права порівняно зі звичайними учасниками, оскільки вони можуть створювати та керувати власними подіями. Їх функціональність включає:

- всі можливості ролі Participant
- створення нових подій
- редагування та оновлення інформації про свої події
- перегляд списку зареєстрованих учасників
- експорт даних про реєстрації
- розсилка повідомлень учасникам події
- завантаження матеріалів та презентацій
- перегляд детальної статистики по своїх подіях

Спікери є важливою частиною екосистеми, оскільки вони створюють контент, який приваблює учасників. Система надає їм зручні інструменти для управління подіями та комунікації з аудиторією.

Сценарії взаємодії користувачів описують типові послідовності дій, які виконують користувачі під час роботи з системою для досягнення конкретних цілей [23].

1. Сценарій 1 «Реєстрація нового користувача».

Процес реєстрації є критично важливим для залучення нових користувачів до системи. Він має бути простим та інтуїтивно зрозумілим [24].

Передумови цього сценарію – у користувача немає облікового запису в системі.

Дії користувача:

1. користувач заходить на головну сторінку системи.
2. Натискає кнопку "Sign Up" у верхньому меню.
3. Система відображає форму реєстрації рисунок 2.3.
4. Користувач вводить свої дані:
   - Email адресу.
   - Повне ім'я.
   - Пароль (мінімум 8 символів).
5. Система валідує введені дані.
6. Якщо дані коректні, створюється новий обліковий запис.
7. Користувач автоматично авторизується в системі.
8. Відображається привітальне повідомлення.

Форма реєстрації з полями для введення даних показана на рис. 2.3.

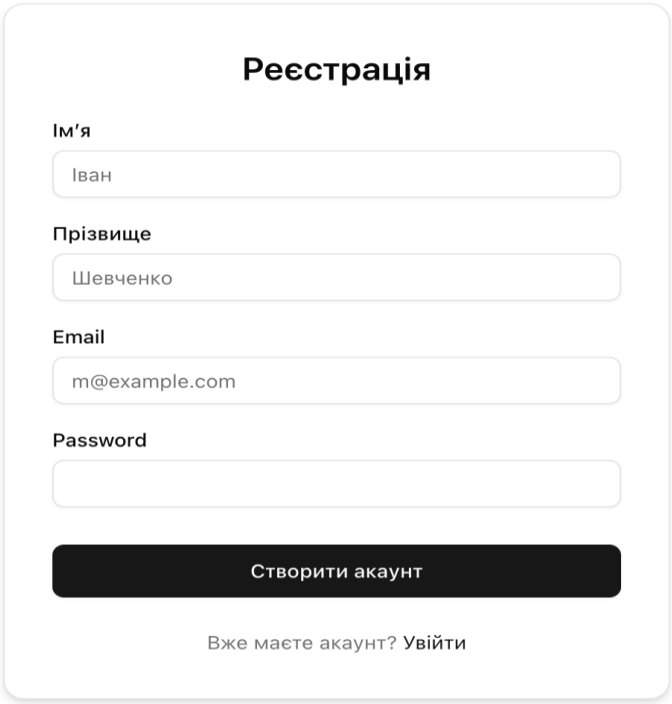

Рисунок 2.3 – Форма реєстрації з полями для введення даних

Альтернативні дії:

- якщо Email вже використовується, система показує повідомлення про помилку;

- якщо паролі не співпадають, підсвічується поле з помилкою;
- якщо пароль занадто короткий, відображається підказка.

2. Сценарій 2 «Авторизація користувача».

Процес входу в систему має бути швидким та безпечним, з можливістю відновлення паролю у разі необхідності.

Передумови цього сценарію – користувач має зареєстрований обліковий запис.

Дії користувача:

1. Користувач натискає кнопку "Sign In"
2. Відкривається форма авторизації рисунок 2.4
3. Користувач вводить email та пароль
4. Система перевіряє облікові дані
5. При успішній авторизації користувач перенаправляється на головну сторінку
6. В хедері відображається ім'я користувача та меню профілю.

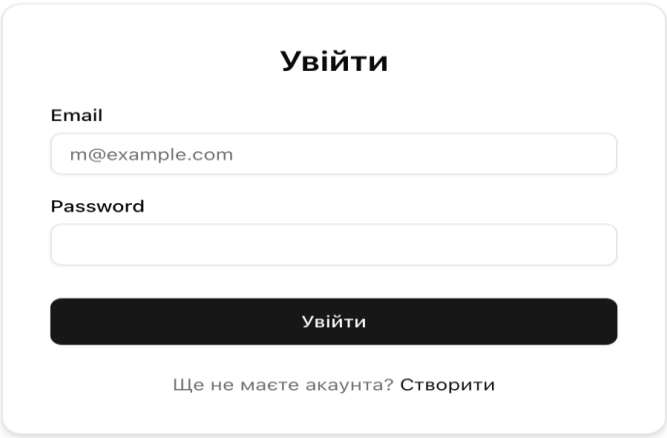

Рисунок 2.4 – Форма входу з полями Email та пароль

3. Сценарій 3 «Пошук та фільтрація подій».

Ефективний пошук подій є ключовою функцією системи, яка дозволяє користувачам швидко знаходити цікаві заходи [25].

Передумови цього сценарію – «Користувач авторизований в системі».

Дії користувача:

1. Користувач переходить на сторінку «Dashboard (events)».
2. Система відображає список всіх доступних подій рис. 2.5.
3. Користувач використовує фільтри:

– поле пошуку;

– дата проведення range date picker;

– ключові слова.

4. Система динамічно оновлює список подій відповідно до фільтрів
5. Користувач може сортувати результати за датою, популярністю або назвою
6. При натисканні на подію відкривається детальна інформація

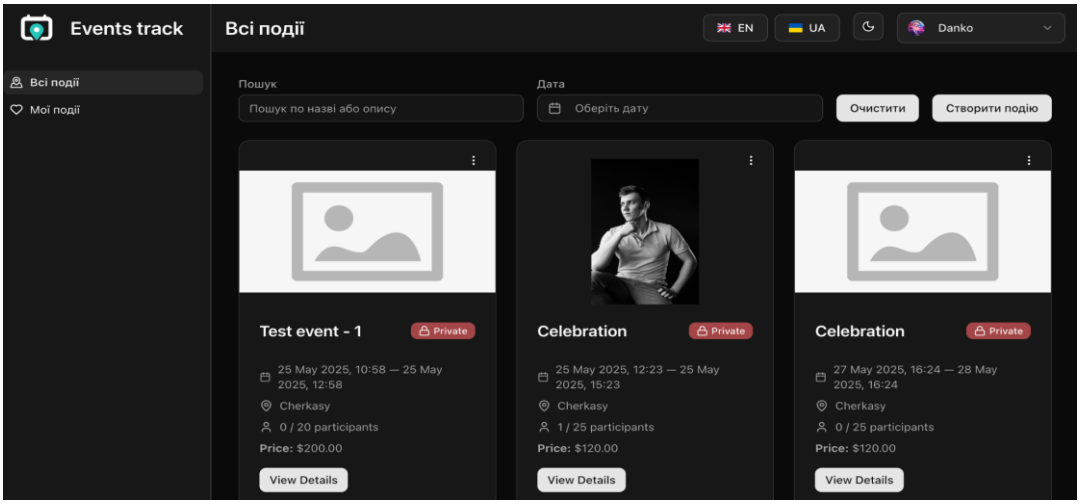

Рисунок 2.5 – Сторінка подій з панеллю фільтрів та списком подій

4. Сценарій 4 «Створення нової події (лише для адміністраторів)».

Адмістратори можуть створювати власні події через зручну форму з валідацією даних [26].

Передумови цього сценарію – користувач має роль Admin.

Дії користувача:

1. Admin натискає кнопку «Create Event» на панелі керування.
2. Відкривається форма створення події, зображеної на рис. 2.6.
3. Спікер заповнює обов'язкові поля:

– назва події;

– опис;
– дата та час початку;
– дата та час завершення;
– місце проведення;
– максимальна кількість учасників;
– категорія.

4. Опційно додає зображення події
5. Натискає кнопку «Create».
6. Система валідує дані та створює подію.
7. Спікер перенаправляється на сторінку створеної події.

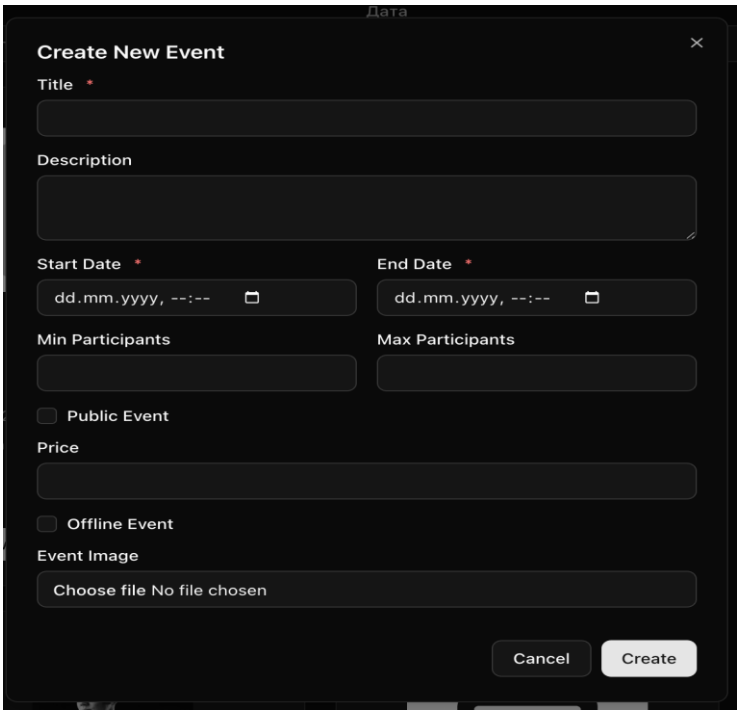

Рисунок 2.6 – Форма створення події з усіма полями

5. Сценарій 5 «Реєстрація на подію» – процес реєстрації на подію має бути максимально простим для користувача.

Передумови цього сценарію – користувач авторизований, подія має вільні місця [28].

Основні дії користувача:

1. Користувач переглядає деталі події.
2. Натискає кнопку «Register».
3. Система перевіряє наявність вільних місць.
4. Створюється запис про реєстрацію.
5. Користувач отримує підтвердження на Email.
6. Подія додається до календаря користувача.
7. Кнопка змінюється на «Registered».

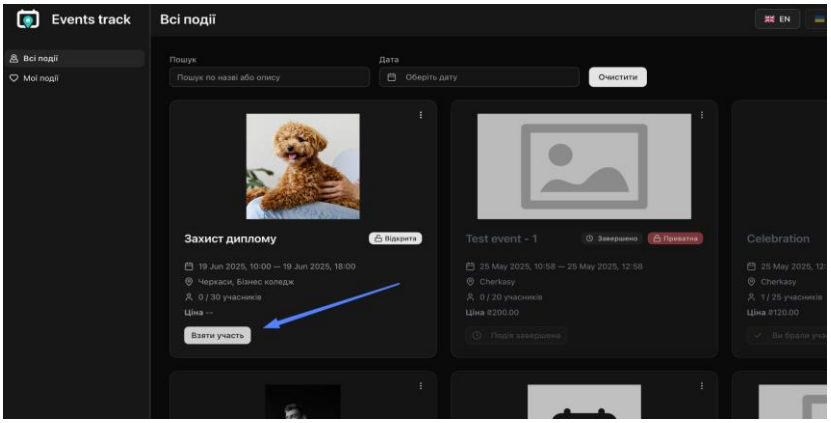

Рисунок 2.7 - Карта події з кнопкою реєстрації

6. Сценарій 6 «Управління профілем» – користувачі можуть налаштовувати свій профіль.

Передумови цього сценарію – користувач авторизований [29].

Основні дії користувача:

1. Користувач натискає на своє ім'я в хедері.
2. Вибирає «Profile» з випадаючого меню.
3. Відкривається сторінка профілю з особиста інформацією по юзеру.
4. Користувач може редагувати свої дані.
5. Зміни зберігаються автоматично або після натискання «Save»

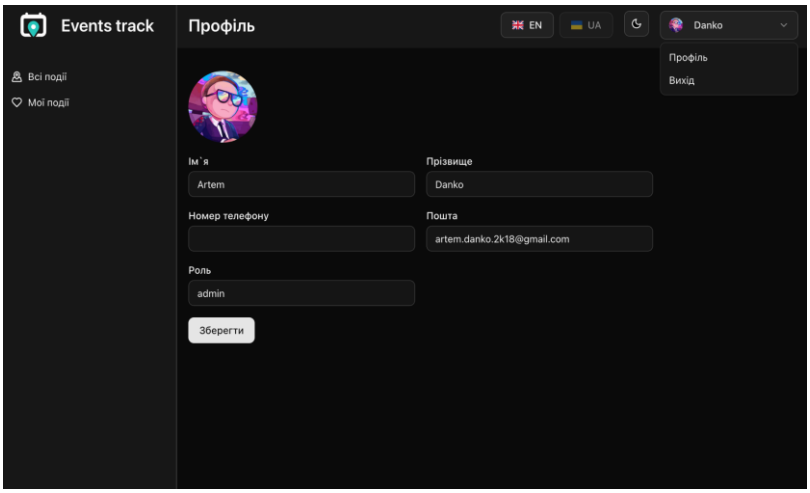

Рисунок 2.8 – Сторінка профілю користувача

2.3 Проєктування інтерфейсу користувача

Концепція дизайну визначає загальні підходи до візуального оформлення та зручності користування інтерфейсом системи. Інтерфейс користувача Events Track розроблено з урахуванням сучасних тенденцій веб-дизайну та принципів UX/UI. Основна мета – створити інтуїтивний, естетично привабливий та функціональний інтерфейс, який забезпечує приємний досвід використання для всіх категорій користувачів [30].

Дизайн-система базується на принципах Material Design з адаптацією під специфіку додатку. Використовується мінімалістичний підхід з акцентом на контент та функціональність. Кольорова схема побудована на основі синього як основного кольору, що асоціюється з професіоналізмом та надійністю.

Компонентна система shadcn/ui використовується для побудови інтерфейсу з уніфікованими елементами, що спрощують розробку та забезпечують узгодженість дизайну. Для реалізації інтерфейсу використовується бібліотека компонентів shadcn/ui, яка надає унікальний підхід до UI розробки. На відміну від традиційних бібліотек компонентів, shadcn/ui не є npm пакетом. Замість цього, це колекція копі-пейст компонентів, які розробники можуть інтегрувати безпосередньо в свій проект.

Основні переваги shadcn/ui:

- повний контроль над кодом компонентів;
- можливість кастомізації під конкретні потреби;
- відсутність залежності від зовнішньої бібліотеки;
- використання Radix UI для доступності;
- стилізація через Tailwind CSS;
- TypeScript підтримка з коробки [31].

Компоненти shadcn/ui побудовані на основі Radix UI - низькорівневої бібліотеки UI примітивів, яка забезпечує повну доступність та семантичність. Це означає, що всі компоненти відповідають WCAG стандартам та можуть використовуватися людьми з обмеженими можливостями.

Структура компонентів shadcn/ui побудована за принципом модульності, де кожен компонент має власну директорію з логікою, стилями та типами, що полегшує масштабування та повторне використання. Кожен компонент shadcn/ui складається з декількох частин [32]:

- базова функціональність від Radix UI;
- стилізація через Tailwind CSS класи;
- TypeScript типи для type safety;
- варіанти через class-variance-authority.

Приклад структури компонента Button відображено на рис. 2.9.

```
// shared/ui/button.tsx
import * as React from "react"
import { Slot } from "@radix-ui/react-slot"
import { cva, type VariantProps } from "class-variance-
authority"
import { cn } from "@/shared/lib/utils"

const buttonVariants = cva(
  "inline-flex items-center justify-center rounded-md text-sm
font-medium transition-colors focus-visible:outline-none focus-
visible:ring-2",
  {
    variants: {
      variant: {
        default: "bg-primary text-primary-foreground hover:bg-
primary/90",
        destructive: "bg-destructive text-destructive-foreground
hover:bg-destructive/90",
        outline: "border border-input bg-background hover:bg-
accent",
        secondary: "bg-secondary text-secondary-foreground
hover:bg-secondary/80",
        ghost: "hover:bg-accent hover:text-accent-foreground",
        link: "text-primary underline-offset-4 hover:underline",
      },
      size: {
        default: "h-10 px-4 py-2",
        sm: "h-9 rounded-md px-3",
        lg: "h-11 rounded-md px-8",
        icon: "h-10 w-10",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
)
```

Рисунок 2.9 – Приклад структури компонента Button

Макети основних сторінок задають просторову організацію елементів інтерфейсу та відображають ключові функціональні блоки для зручної навігації та взаємодії користувача з системою. До основних компонентів відносять наступні [33]:

1. Головна сторінка (рис. 2.10) є точкою входу для користувачів і має створювати позитивне перше враження. Вона складається з декількох секцій:

– Hero секція – велика привітальна область з пошуковим полем та закликом до дії. Використовується градієнтний фон та анімації для привернення уваги.

– Категорії подій – горизонтальний список популярних категорій з іконками для швидкої навігації.

– Найближчі події – карусель з картками майбутніх подій, відсортованих за датою.

– Популярні події – сітка з найпопулярнішими подіями за кількістю реєстрацій.

2. Сторінка списку подій відображає всі доступні події з можливістю фільтрації та сортування. Ліва панель містить фільтри:

– Категорія

– Дата (календар)

– Місце проведення

– Ціна

– Формат (онлайн/офлайн)

3. Основна частина відображає картки подій у вигляді сітки. Кожна картка містить:

– Зображення події

– Назву

– Дату та час

– Місце проведення

– Кількість вільних місць

– Кнопку швидкої реєстрації

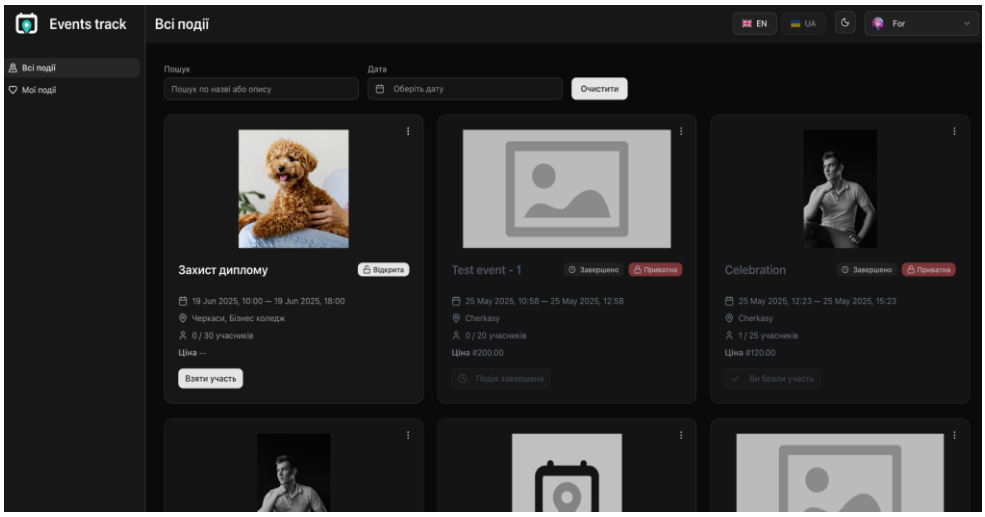

Рисунок 2.10 – Сторінка швидкої реєстрації

4. Детальна сторінка події надає повну інформацію про захід. Структура сторінки сскладається з наступних складових:

– заголовок – велике зображення події з накладеним текстом назви;

– основна інформація – дата, час, місце, категорія в зручному форматі;

– опис - детальний опис події з форматованим текстом;

– інформація про спікера – фото, ім'я, біографія спікера;

– панель реєстрації – sticky панель з кнопкою реєстрації та лічильником місць;

– карта – інтерактивна карта з місцем проведення;

– схожі події – рекомендації інших подій тієї ж категорії.

Система підтримує темну тему для комфортної роботи в умовах низького освітлення. Перемикання між темами відбувається через next-themes:

```
// Налаштування теми
const { theme, setTheme } = useTheme()

return (
  <DropdownMenu>
    <DropdownMenuTrigger asChild>
      <Button variant="ghost" size="icon">
        <Sun className="h-5 w-5 rotate-0 scale-100 transition-all dark:-rotate-90
dark:scale-0" />
        <Moon className="absolute h-5 w-5 rotate-90 scale-0 transition-all
dark:rotate-0 dark:scale-100" />
      </Button>
    </DropdownMenuTrigger>
    <DropdownMenuContent align="end">
      <DropdownMenuItem onClick={() => setTheme("light")}>
        Light
      </DropdownMenuItem>
      <DropdownMenuItem onClick={() => setTheme("dark")}>
        Dark
      </DropdownMenuItem>
      <DropdownMenuItem onClick={() => setTheme("system")}>
        System
      </DropdownMenuItem>
    </DropdownMenuContent>
  </DropdownMenu>
```

Рисунок 2.11 – Перемикання між темами

Всі форми в системі використовують React Hook Form для управління станом та Zod для валідації (рис. 2.11). Це забезпечує:

– миттєву валідацію полів;

– зрозумілі повідомлення про помилки;

– збереження даних при помилках;

– підтримку складних правил валідації.

Компоненти форм інтегровані з shadcn/ui для консистентного вигляду, показаного на рис. 2.12.

```
<Form {...form}>
  <form onSubmit={form.handleSubmit(onSubmit)} className="space-y-8">
    <FormField
      control={form.control}
      name="email"
      render={({ field }) => (
        <FormItem>
          <FormLabel>Email</FormLabel>
          <FormControl>
            <Input placeholder="email@example.com" {...field} />
          </FormControl>
          <FormDescription>
            We'll never share your email with anyone else.
          </FormDescription>
          <FormMessage />
        </FormItem>
      )}
    />
  </form>
</Form>
```

Рисунок 2.12 – Інтегровані компоненти форм

Для покращення користувацького досвіду використовуються subtle анімації:

– плавні переходи між станами;
– анімації появи елементів;
– Skeleton loading для асинхронного контенту;
– Hover ефекти на інтерактивних елементах;
– Progress індикатори для тривалих операцій.

Анімації реалізовані через CSS transitions та Framer Motion для складніших випадків.

Всі компоненти розроблені з урахуванням вимог доступності:

– Правильна семантична розмітка.
– ARIA атрибути для screen readers.
– Keyboard navigation.
– Focus management.
– Достатній контраст кольорів.
– Альтернативний текст для зображень.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Розробка фронтенд частини

Розробка фронтенд частини починається зі створення нового React проєкту з використанням Vite – сучасного інструменту збірки, який забезпечує блискавичну швидкість розробки завдяки native ES modules [34].
Процес ініціалізації проєкту показано на рис. 3.1.

```bash
npm create vite@latest event-track-frontend -- --template react-ts
cd event-track-frontend
npm install
```

Рисунок 3.1 – Процес ініціалізації проєкту

Після створення базового проєкту необхідно встановити всі необхідні залежності. Основні бібліотеки включають [35]:
React екосистема:

- React 19 – основна бібліотека для побудови UI
- React Router DOM;
- React Hook Form;
- React Query – управління серверним станом;

Утиліти та інструменти, які використовуюься:

- TypeScript – статична типізація
- Zod - валідація схем
- Axios - HTTP клієнт
- date-fns - робота з датами
- class-variance-authority - утиліта для варіантів компонентів

UI та стилізація:

– Tailwind CSS - utility-first CSS фреймворк
– shadcn/ui компоненти - копіюються в проект
– Radix UI - примітиви для доступності
– Lucide React - бібліотека іконок

Організація коду відповідає принципам Feature-Sliced Design, що забезпечує масштабованість та підтримуваність проєкту. Завдання розподілені так, що кожен елемент має свою відокремлену роль [36]:

– App-елемент містить глобальну конфігурацію додатку, провайдери та точку входу. Тут налаштовуються всі необхідні контексти та ініціалізується додаток.
– Entities-елемент включає бізнес-сутності системи. Кожна сутність має свою папку з API методами, типами та UI компонентами для відображення.
– Features-елемент містить функціональні можливості, які можуть використовувати entities. Наприклад, функція створення події використовує сутність Event.
– Pages-елемент відповідає за композицію сторінок з використанням widgets та features. Кожна сторінка відповідає маршруту в додатку.
– Shared-елемент містить переиспользуемый код, який не прив'язаний до конкретної бізнес-логіки - UI компоненти, утиліти, константи.

Налаштування shadcn/ui

Shadcn/ui не встановлюється як звичайна бібліотека. Замість цього використовується CLI інструмент для додавання компонентів [37]:

bash

npx shadcn-ui@latest init

Під час ініціалізації вибираються наступні опції:

– TypeScript – Yes
– Style – Default

– Base color – Slate
– CSS variables – Yes
– Tailwind config – tailwind.config.js
– Components – src/shared/shadcn-ui
– Utils - src/shared/lib
– React Server Components – No

Після ініціалізації можна додавати необхідні компоненти:

bash

npx shadcn-ui@latest add button

npx shadcn-ui@latest add form

npx shadcn-ui@latest add input

npx shadcn-ui@latest add card

npx shadcn-ui@latest add dialog

Кожен компонент копіюється в проект і може бути модифікований під конкретні потреби [38].

Система аутентифікації є критично важливою для безпеки додатку. Вона реалізована з використанням JWT токенів та Zustand для управління станом. Лістинг програми створення auth store показано в додатку А.1

Форма входу використовує React Hook Form з Zod валідацією для забезпечення коректності даних. Лістинг програми показано в додатку А.2. Форма входу в систему показана на рис. 3.2.

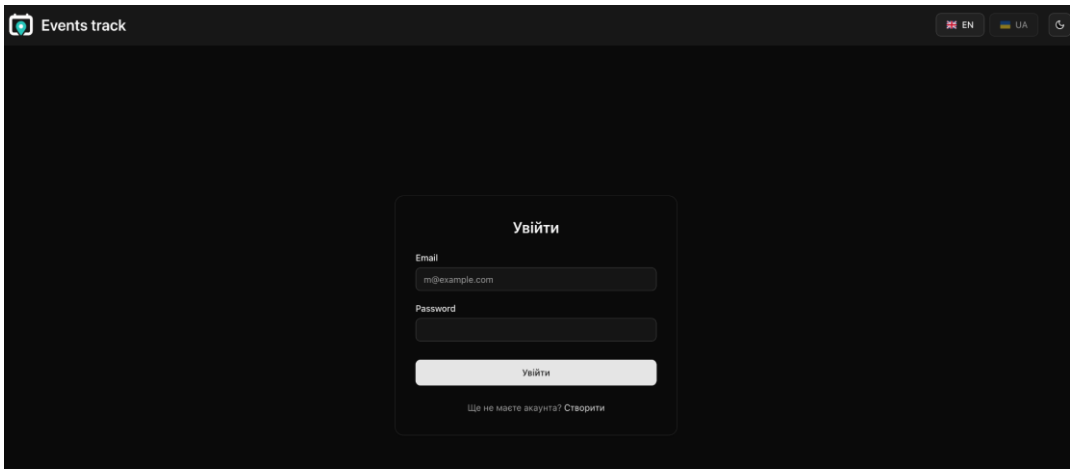

Рисунок 3.2 – Форма входу в систему

Список подій є центральним компонентом системи [39]. Він використовує React Query для управління даними та підтримує фільтрацію, сортування та пагінацію Лістинг програми показано в додатку А.3.

Система підтримує багатомовність через i18next [40]. Конфігурація включає автоматичне визначення мови браузера та можливість перемикання між мовами. Даний лістинг представлено в додатку А.4.

Використання в компонентах має наступний вигляд:

typescript

```
const { t } = useTranslation('events');
return (
<h1>{t('events.title')}</h1>
);
```

Для забезпечення високої продуктивності додатку використовуються різні техніки оптимізації, зокрема:

– Code Splitting – розділення коду на chunks для зменшення початкового розміру bundle має наступний вигляд:

typescript

```
const AdminPage = lazy(() => import('@/pages/admin'));
```

– Мемоізація – використання React.memo та useMemo для уникнення зайвих ререндерів виглядає наступним чином:

typescript

```
const MemoizedEventCard = React.memo(EventCard);
```

– Віртуалізація – для довгих списків використовується react-window, що відображається так:

typescript

```
import { FixedSizeList } from 'react-window';
```

– Оптимізація зображень - lazy loading та використання next-gen форматів.

3.2 Розробка бекенд частини

NestJS – це прогресивний Node.js фреймворк для побудови ефективних та масштабованих серверних додатків. Він використовує TypeScript за замовчуванням і комбінує елементи ООП (Object Oriented Programming), ФП (Functional Programming) та ФРП (Functional Reactive Programming).

Основні переваги NestJS:

- модульна архітектура
- Dependency Injection контейнер
- вбудована підтримка TypeScript
- декоратори для декларативного коду
- інтеграція з популярними бібліотеками
- CLI для генерації коду

Структура проєкту має наступний вигляд:

- Backend проєкт організований за модульним принципом, зображеним в додатку А.5:

Налаштування даного проєкту починається зі створення нового NestJS проєкту і виглядає наступним чином:

bash

npm i -g @nestjs/cli

nest new event-track-backend

cd event-track-backend

Встановлення необхідних залежностей продовжиться автоматично на етапі виконання команди nest new event-track-backend. Коли ви запускаєте nest new, NestJS CLI не лише генерує базову структуру проєкту, але й автоматично встановлює всі базові залежності, перелічені у файлі package.json нового проєкту, використовуючи npm або yarn (залежно від того, що у вас налаштовано за замовчуванням або що ви оберете під час створення).

Отже, після успішного завершення команди nest new event-track-backend, не потрібно додатково запускати npm install або yarn install, оскільки це вже буде зроблено автоматично і матиме наступний вигляд:

```
bash
npm install @nestjs/typeorm typeorm pg
npm install @nestjs/jwt passport-jwt @nestjs/passport
npm install bcrypt class-validator class-transformer
npm install @nestjs/config @nestjs/swagger
npm install multer @types/multer
```

TypeORM використовується в даному випадку як ORM (Object-Relational Mapping) для спрощення взаємодії між об’єктами додатку та таблицями бази даних PostgreSQL. Завдяки TypeORM, розробник може описувати структуру таблиць у вигляді класів та працювати з ними як з об’єктами JavaScript/TypeScript без написання сирих SQL-запитів. Це забезпечує зручність розробки, підтримку міграцій, автоматичне створення таблиць та збереження зв’язків між сутностями (OneToMany, ManyToOne тощо).Його конфігурація представлена в додатку А.6.

Entities є логічним відображенням об'єктів предметної області, з якими працює веб-додаток. Кожна сутність представляє окрему таблицю в базі даних, що містить поля (атрибути), які відповідають характеристикам цього об'єкта. Вони задають структуру зберігання даних і взаємозв’язки між різними об'єктами, такими як користувачі, події, реєстрації тощо. Через правильне моделювання сутностей забезпечується цілісність, узгодженість та ефективність обробки інформації. Їх детальний опис наведено в додатку А.7.

Автентифікація реалізована з використанням JWT токенів. У системі реалізовано механізм автентифікації на основі JWT (JSON Web Token), який забезпечує безпечний обмін даними між клієнтом і сервером. Після введення коректних облікових даних (логін і пароль) користувач отримує від сервера токен, що містить закодовану інформацію про його особу. Цей токен надсилається разом із кожним наступним запитом до захищених маршрутів

через заголовок Authorization, що дозволяє серверу ідентифікувати користувача без повторної перевірки облікових даних. Використання JWT забезпечує безстейтову автентифікацію, підвищує продуктивність і спрощує масштабування додатку. Модуль автентифікації представлено в додатку А.8.

Стратегія для валідації JWT токенів полягає в перевірці достовірності, цілісності та актуальності токена перед наданням доступу до захищених ресурсів (представлено в додатку А.9).

У даному проєкті стратегія реалізована наступним чином:

– отримання токена – клієнт надсилає JWT у заголовку HTTP-запиту.

– Перевірка підпису – сервер використовує секретний ключ, збережений у змінних середовища (JWT_SECRET), для перевірки криптографічного підпису токена. Якщо підпис некоректний – токен відхиляється.

– Перевірка терміну дії – токен містить поле exp – час закінчення дії. Якщо поточний час перевищує значення exp, токен вважається недійсним.

– Декодування payload – якщо перевірки пройдено успішно, з payload витягуються ідентифікатор користувача, його роль або інша службова інформація, яка використовується для авторизації.

– Обробка через middleware або стратегію (Passport.js) – у додатку, що використовує NestJS або Express, ця логіка часто реалізується як JwtStrategy, що інтегрується з бібліотекою Passport. Стратегія реалізує метод validate(payload), у якому відбувається пошук користувача в базі даних і передача його в контекст запиту.

Відмова у доступі у разі помилки – якщо будь-який із вищезгаданих етапів не виконується, запит блокується з помилкою 401 Unauthorized.Така стратегія забезпечує ефективну та безпечну перевірку автентичності користувачів без зберігання сесій на сервері, що особливо важливо для масштабованих веб-додатків.

Guards використовуються для контролю доступу до ендпоінтів (додаток А.10). Guards (захисники) є механізмом авторизації, який перевіряє, чи має користувач право доступу до певного ендпоінта до того, як виконається логіка обробки запиту. Вони працюють як фільтри на рівні маршруту та визначають, чи дозволити виконання запиту на основі заданих умов – наприклад, наявності валідного JWT токена або ролі користувача.

У типовій реалізації (у NestJS) guard імплементує інтерфейс CanActivate і містить метод canActivate(), який повертає true або false. Якщо повертається false, запит блокується з помилкою 403 Forbidden або 401 Unauthorized.

Guards забезпечують:

– контроль автентифікації (наприклад, перевірка JWT);

– контроль авторизації (перевірка ролі: користувач, адміністратор тощо);

– централізовану перевірку доступу до критичних ресурсів системи.

Таким чином, Guards є ключовим інструментом реалізації політик безпеки в архітектурі веб-додатка.

Модуль подій відповідає за CRUD операції з подіями. Цей модуль реалізує базову функціональність управління подіями – створення, перегляд, оновлення та видалення. Він обробляє запити від користувачів (організаторів), передає дані до бази через ORM (наприклад, TypeORM) і забезпечує взаємодію з сутністю події (Event). Реалізація даного модуля представлена в додатку А.11

Окрім CRUD-операцій, модуль також може включати фільтрацію, сортування подій, логування змін і перевірку логіки відображення подій відповідно до часу або статусу.

Модуль включає:

– створення нових подій з валідацією вхідних даних;

– отримання переліку або деталей події за ідентифікатором;

– редагування подій з перевіркою прав доступу;

– видалення подій, доступне лише певним ролям (наприклад, організатору або адміністратору).

Для валідації вхідних даних використовується class-validator (додаток А.12). Бібліотека class-validator застосовується для автоматичної перевірки коректності вхідних даних, які надходять від користувача у вигляді DTO (Data Transfer Object). Вона дозволяє визначити набір правил безпосередньо в класах, що описують структуру запиту, з використанням спеціальних декораторів.

Наприклад:

– @IsString(), @IsNotEmpty() – перевірка, що поле є рядком і не порожнє;

– @IsEmail() – перевірка формату email-адреси;

– @IsDateString() – перевірка правильності дати;

– @Length(5, 100) – перевірка довжини рядка.

Перед передачею об'єкта в бізнес-логіку система автоматично здійснює валідацію, і в разі невідповідності користувач отримує опис помилок. Такий підхід підвищує надійність додатку та дозволяє централізовано контролювати якість вхідних даних.

Для завантаження зображень використовується Multer: Бібліотека Multer використовується як middleware для обробки multipart/form-data запитів, які містять файли, зокрема зображення. Вона інтегрується з Express або NestJS і дозволяє приймати файли з форм, зберігати їх на сервері або передавати у зовнішнє сховище.

У реалізації веб-додатку Multer застосовується для:

– прийому зображень подій або аватарів користувачів;

– збереження файлів у локальну директорію або хмарне сховище (наприклад, AWS S3);

– обмеження типів файлів (наприклад, лише .jpg, .png) та їхнього розміру;

– генерації унікальних назв файлів для уникнення конфліктів.

Налаштування Multer дозволяє контролювати безпеку та структуру завантажень, а також легко зв’язувати збережені файли з відповідними сутностями в базі даних (додаток А.13).

NestJS має вбудовану підтримку Swagger для автоматичної генерації документації до REST API показано в додатку А.14. За допомогою модуля @nestjs/swagger можна легко описати структуру запитів, відповіді, параметри маршрутів та моделі даних на основі існуючих DTO і декораторів.

Основні переваги використання Swagger у NestJS:

– Автоматична генерація документації на основі коду;

– Інтерактивний інтерфейс для тестування ендпоінтів;

– Зручна візуалізація структури запитів і відповідей;

– Можливість коментування маршрутів за допомогою декораторів, таких як @ApiOperation(), @ApiResponse(), @ApiTags() тощо;

– Легка інтеграція з фронтенд-командою або сторонніми клієнтами API.

Swagger-документація доступна за адресою http://localhost:3000/api і значно спрощує процес розробки, тестування та підтримки API (зображено на рис.3.3).

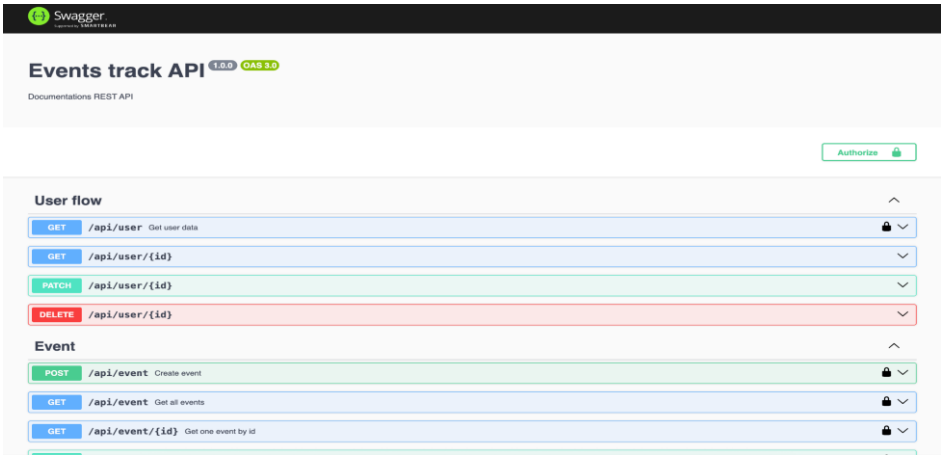

Рисунок 3.3 – Swagger документація API

У додатку реалізовано глобальний фільтр для обробки помилок, який перехоплює всі винятки (exceptions), що виникають під час виконання запитів, і повертає уніфіковану та зрозумілу відповідь клієнту (додаток А.15).

У NestJS такий фільтр створюється шляхом реалізації інтерфейсу ExceptionFilter або використання вбудованого класу HttpExceptionFilter. Він дозволяє:

– формувати єдиний формат помилок (наприклад, JSON з полями statusCode, message, timestamp, path);

– логувати критичні помилки;

– обробляти як стандартні помилки (наприклад, BadRequestException, UnauthorizedException), так і кастомні винятки;

– покращити зручність відлагодження та взаємодії з API на клієнтському боці.

Глобальний фільтр реєструється через app.useGlobalFilters(new HttpExceptionFilter()) в головному модулі main.ts, що гарантує охоплення всіх помилок у додатку.

NestJS має вбудовану підтримку тестування з Jest. NestJS інтегрований з фреймворком Jest — потужним інструментом для написання юніт- та інтеграційних тестів у проєктах на TypeScript. Під час створення нового застосунку за допомогою CLI NestJS автоматично генерує базову структуру з підтримкою Jest (додаток А.16).

Основні можливості тестування з Jest у NestJS:

– юніт-тестування окремих сервісів, контролерів і пайпів без залежностей;

– мокінг залежностей через jest.mock() або TestingModule.overrideProvider();

– інтеграційне тестування з підключенням справжніх модулів, бази даних або зовнішніх API;

– автоматичне покриття коду (jest --coverage);

– швидке зворотне тестування завдяки гарячому перезапуску (--watch).

NestJS також надає клас TestingModule, що дозволяє створювати ізольовані середовища для кожного тесту, а також зручно перевіряти поведінку компонентів у контрольованих умовах. Завдяки цьому тестування в NestJS є гнучким, масштабованим і придатним для реальних продакшен-додатків.

3.3 Основні функції проєкту

Система пошуку дозволяє користувачам швидко знаходити потрібні події за різними критеріями. Вона реалізована як інтерактивний модуль із повною підтримкою фільтрації, повнотекстового пошуку та пагінації. Реалізація охоплює як фронтенд, так і бекенд частини (додаток А.17):

1. Frontend частина:

– Компонент фільтрів реалізований з використанням бібліотеки shadcn/ui, забезпечує зручний та інтуїтивно зрозумілий інтерфейс для вибору параметрів пошуку.

– Debounce оптимізує взаємодію з API, запобігаючи надмірній кількості запитів при введенні пошукового запиту користувачем.

– Збереження фільтрів в URL для можливості поділитися дозволяє ділитися пошуковими запитами через посилання та забезпечує зручну навігацію з підтримкою історії браузера.

2. Backend частина:

– Query builder для динамічної побудови SQL запитів. Він дозволяє динамічно будувати SQL-запити на основі вхідних параметрів.

– Пошук реалізований із застосуванням повнотекстової індексації по назві та опису подій, що дозволяє знаходити події навіть за частковим збігом тексту.

– Фільтрація за категорією, датою, місцем підвищує релевантність результатів.

– Пагінація реалізована як частина API-відповіді, що дозволяє поступово завантажувати результати та зменшити навантаження на сервер.

Процес реєстрації включає декілька етапів перевірки та обробки (додаток А.18):

1. Перевірка авторизації користувача – система перевіряє, чи є користувач авторизованим у системі. Це означає, що користувач повинен бути увійшовшим у свій обліковий запис або пройти процедуру входу (логін). Якщо користувач не авторизований, система може запропонувати зареєструватися або увійти, щоб продовжити реєстрацію. Така перевірка потрібна для забезпечення безпеки та індивідуалізації дій користувача.

2. Перевірка доступності місць – далі система перевіряє наявність вільних місць для реєстрації на конкретну подію чи послугу. Це включає звірку з базою даних або системою бронювання, щоб переконатися, що кількість доступних місць не вичерпана. Якщо місць немає, користувачу видається відповідне повідомлення, і процес реєстрації припиняється або пропонується поставити заявку у чергу

3. Перевірка дублікатів реєстрації – щоб уникнути повторної реєстрації одного і того ж користувача на ту саму подію, система виконує пошук у базі даних на предмет існуючих записів з ідентифікаційними даними (наприклад, ID користувача або email). Якщо дублікати знаходяться, система або блокує повторну реєстрацію, або пропонує оновити існуючу, щоб уникнути помилок і плутанини

4. Створення запису в базі даних – якщо всі перевірки пройшли успішно, система створює новий запис у базі даних із деталями реєстрації: інформацією про користувача, подію, дату, час та інші необхідні параметри. Цей запис слугує офіційним підтвердженням участі користувача і є основою для подальшої обробки.

5. Відправка email підтвердження – після успішного створення запису користувачу автоматично надсилається лист-підтвердження на вказану електронну адресу. У цьому листі міститься інформація про зареєстровану подію, інструкції, правила участі або інші важливі деталі. Цей крок допомагає забезпечити зворотній зв’язок і підтвердити, що реєстрація відбулась коректно.

6. Оновлення статистики – на фінальному етапі система оновлює внутрішню статистику: кількість зареєстрованих користувачів, зайнятих місць, відсоток заповненості тощо. Ці дані можуть використовуватись для аналітики, прийняття рішень організаторами подій, а також для відображення актуальної інформації іншим користувачам.

Користувачі мають змогу індивідуалізувати свій досвід взаємодії з системою завдяки гнучким параметрам профілю. Основні категорії налаштувань охоплюють такі аспекти:

1. Особисту інформацію – користувач може редагувати свої персональні дані, зокрема:

- ім’я та прізвище;
- фото профілю;
- контактну електронну адресу;
- номер телефону;
- соціальні мережі або посилання на зовнішні профілі.

Ці дані можуть використовуватись для персоналізації сервісу, відображення в комунікаціях чи для спрощення участі в подіях.

2. Налаштування сповіщень – користувач може самостійно вибирати, які типи повідомлень він бажає отримувати та через які канали:

- Email-сповіщення;
- сповіщення в додатку;
- SMS.

Також можливе налаштування частоти повідомлень або повне їх вимкнення.

3. Зміну пароля – у розділі безпеки профілю користувач має змогу:

- змінити поточний пароль, вказавши старий та новий
- активувати додаткову автентифікацію (наприклад, 2FA)
- переглянути історію авторизацій або вийти з усіх активних сесій

Це дозволяє підвищити захищеність облікового запису.

4. Вибір мови інтерфейсу – інтерфейс може підтримувати декілька мов, і користувач має змогу обрати зручну для нього мову. Зміна мови застосовується до всіх текстів, кнопок, підказок і навігаційних елементів інтерфейсу. Ця опція особливо корисна для платформ із міжнародною аудиторією.

5. Налаштування приватності – користувач може керувати тим, яка інформація про нього буде доступна іншим:

- відображення профілю в списках учасників
- доступність контактних даних
- параметри видимості активності (наприклад, участь у подіях)
- можливість закрити профіль від публічного перегляду

Приватність є ключовим аспектом довіри між користувачем і системою, тому ці налаштування мають бути прозорими і гнучкими. Відображення функції управління профілем показано на рис. 3.4.

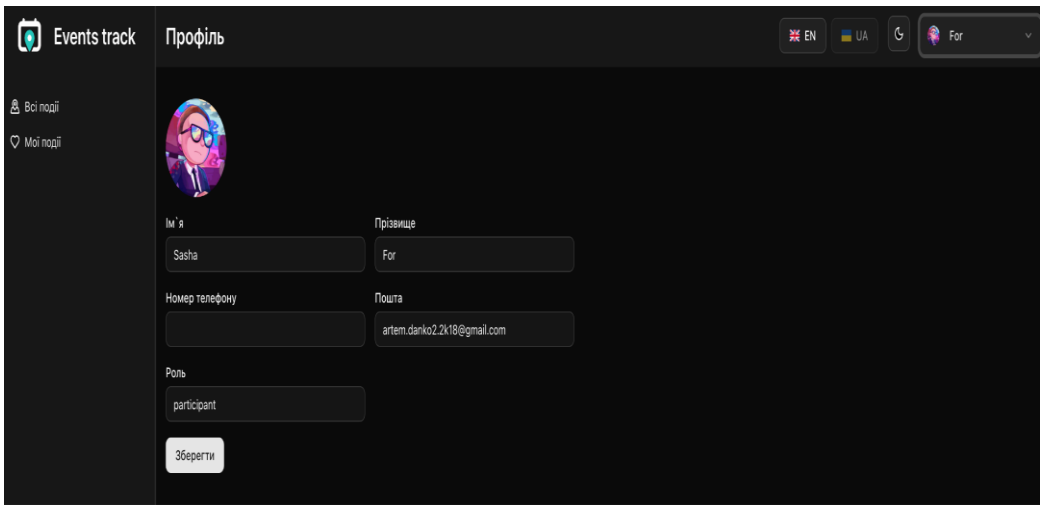

Рисунок 3.4 – Функція управління профілем

Система надає спікерам інтуїтивно зрозумілий інтерфейс для створення та адміністрування власних подій. Кожен крок оптимізовано для швидкого заповнення, гнучкої кастомізації та зручної модерації подій. Спікери можуть створювати власні події через зручний інтерфейс:

1. Заповнення форми з деталями події – спікер вводить основну інформацію про подію, зокрема:

- назва події;
- дата та час початку / завершення;
- локація (фізична або онлайн-посилання);
- короткий опис для анонсів;
- розгорнутий опис для сторінки події (з можливістю форматування).

Цей етап є основою для виведення події у каталозі, формуванні сповіщень та генерації SEO-метаданих (у разі публічної платформи).

2. Завантаження зображення – можливість прикріпити зображення:

- обкладинка події (формат JPG/PNG/WebP);
- опціонально: логотип організації чи банери;
- система автоматично перевіряє розміри та формат, можлива обробка (обрізання, стиснення).

Це візуально покращує відображення події в загальному каталозі та соціальних мережах.

3. Вибір категорії та тегів – для покращення навігації і релевантного пошуку спікер вказує:

- категорію події (наприклад: технології, освіта, бізнес);
- теги (вільне або рекомендоване заповнення);
- цільову аудиторію (за потреби).

Ці параметри впливають на фільтрацію в інтерфейсі користувача та аналітичну звітність.

4. Встановлення обмежень реєстрації – організатор може задавати обмеження для контролю потоку учасників:

- максимальна кількість учасників;
- кінцева дата реєстрації;
- необхідність підтвердження заявки вручну;
- пароль на реєстрацію або закритий доступ по інвайту;
- чекбокси з підтвердженням правил або GDPR.

Це дозволяє адаптувати події під різні сценарії – від відкритих вебінарів до приватних тренінгів.

5. Публікація або збереження як чернетка – На фінальному етапі спікер може:

- опублікувати подію, зробивши її доступною для користувачів
- зберегти як чернетку – для подальшого редагування
- призначити автоматичну публікацію у заданий час

Після публікації подія відображається у загальному списку, а також доступна через пряме посилання. Відображення функції створення та управління подіями показано на рис. 3.5.

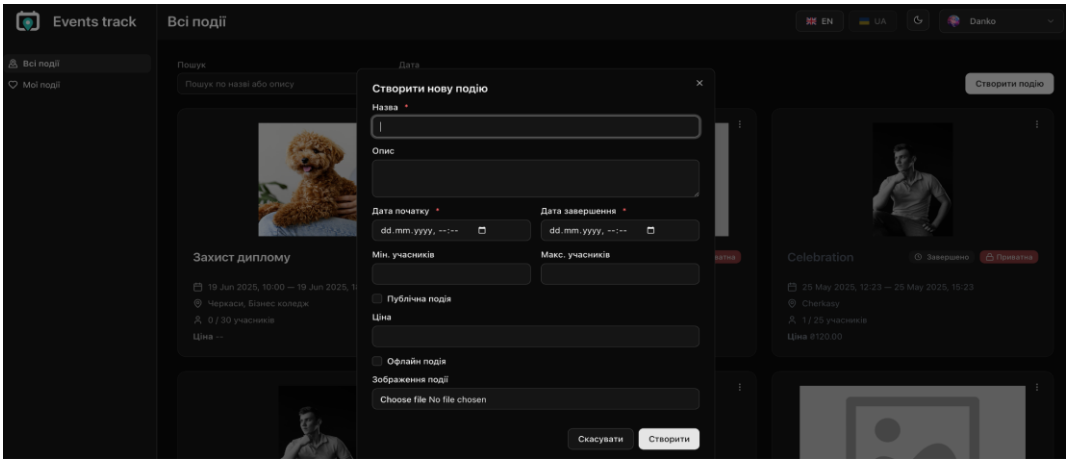

Рисунок 3.5 – Функція створення та управління подіями

Адміністратори системи отримують розширений доступ до спеціальної панелі управління, що дозволяє ефективно контролювати роботу платформи, підтримувати порядок і забезпечувати безпеку.

Адміністратори мають доступ до спеціальної панелі з можливостями:

1. Перегляд всіх користувачів та подій – адміністратор має змогу бачити повний список зареєстрованих користувачів та створених подій із можливістю сортування, фільтрації та пошуку за різними параметрами (ім’я, email, статус, дата створення, категорія події тощо). Це забезпечує швидкий доступ до необхідної інформації та контроль за активністю.

2. Модерація контенту – адміністратор може переглядати та редагувати або видаляти небажаний або некоректний контент, зокрема, опис подій, коментарі користувачів, завантажені зображення та файли, повідомлення у внутрішніх чатах (за потреби). Модерація допомагає підтримувати високий рівень якості платформи та дотримання правил спільноти.

3. Блокування користувачів – у разі порушення правил платформи або підозрілої активності адміністратори можуть:

- тимчасово або постійно заблокувати обліковий запис користувача
- обмежити доступ до окремих функцій або розділів
- відновити або видалити обліковий запис за потребою. Цей інструмент важливий для забезпечення безпеки та порядку.

4. Перегляд системних логів – адміністратор має доступ до журналів подій системи, що включають:

- логи авторизацій (успішні та неуспішні спроби входу);
- дії користувачів (створення, редагування, видалення контенту).
- помилки та системні збої

Ця інформація допомагає аналізувати активність, виявляти загрози і проводити аудит.

5. Налаштування категорій – адміністратори можуть створювати, редагувати або видаляти категорії подій і тегів, що використовуються у системі. Це дає змогу підтримувати актуальність класифікації та зручність пошуку для користувачів.

6. Масові операції – для підвищення ефективності адміністратори мають можливість виконувати масові дії:

- масове видалення або блокування користувачів;
- масове оновлення статусів подій;
- експорт даних користувачів або подій у вигляді звітів;
- масова розсилка повідомлень або сповіщень.

Такі операції суттєво спрощують управління великою кількістю даних.

3.4 Засоби захисту даних

Безпека системи починається з надійної автентифікації. Використовується JWT (JSON Web Tokens) для stateless автентифікації:

Access Token – короткоживучий токен (15 хвилин) для доступу до API
Refresh Token - довгоживучий токен (7 днів) для оновлення access token
Процес автентифікації:

1. Користувач вводить email та пароль – користувач вводить свої облікові дані у форму входу на клієнтській стороні (веб-інтерфейс або мобільний додаток). Зазвичай це – електронна адреса та пароль.

2. Сервер перевіряє дані та генерує токени – отримавши дані, сервер виконує:

- валідацію формату email;
- пошук користувача у базі даних за email;
- перевірку відповідності пароля (зазвичай через хешування та порівняння).

Якщо дані коректні, сервер генерує два типи токенів:

- Access token – короткостроковий токен доступу, що використовується для авторизації запитів.
- Refresh token – довгостроковий токен для оновлення access token без повторного входу.

3. Токени зберігаються на клієнті – згенеровані токени відправляються клієнту і зберігаються:

– Access token – у пам’яті клієнтського додатка (наприклад, у пам’яті браузера або локальному сховищі)

– Refresh token – зазвичай у безпечнішому місці (HTTP-only cookie або іншому захищеному сховищі)

Правильне зберігання токенів критично для безпеки.

4. При кожному запиті Access token відправляється в заголовку – для доступу до захищених ресурсів клієнт автоматично додає Access token у заголовок HTTP-запиту (наприклад, Authorization: Bearer <token>). Сервер перевіряє валідність токена та надає або відмовляє у доступі.

5. При закінченні терміну дії використовується refresh token – коли термін дії access token спливає, клієнт надсилає refresh token до сервера для отримання нового access token без повторного введення пароля. Сервер перевіряє refresh token і, якщо він валідний, генерує новий access token (іноді й новий refresh token) і відправляє його клієнту.

Ця схема забезпечує баланс між зручністю користування (автоматичне оновлення сесії) та безпекою (короткий термін життя Access token і контроль Refresh token).

Паролі ніколи не зберігаються в чистому вигляді, щоб захистити їх від викрадення у разі злому бази даних. Для цього застосовується процес хешування – перетворення пароля у складний для відновлення набір символів, який називається хешем. Для безпечного хешування використовується алгоритм bcrypt. Він має кілька важливих властивостей:

1. Сіль – випадковий додатковий рядок, який додається до пароля перед хешуванням, щоб зробити кожен хеш унікальним. Це ускладнює злом за допомогою заздалегідь обчислених таблиць.
2. Складність – параметр, який регулює, скільки часу витрачається на хешування, що захищає від швидких атак методом перебору.
3. Адаптивність – складність можна збільшувати з розвитком техніки, щоб зберігати безпеку.

Під час реєстрації пароль користувача поєднується з сіллю і обробляється алгоритмом bcrypt, після чого зберігається у базі у вигляді хешу та набуває наступного вигляду:

```
typescript
const saltRounds = 10;
const hashedPassword = await bcrypt.hash(password, saltRounds);
```

Bcrypt має вбудований захист від rainbow table атак та автоматично генерує сіль.

При вході користувача введений пароль повторно обробляється таким самим способом і порівнюється з хешем у базі. Якщо вони співпадають, користувач отримує доступ.

Завдяки такому підходу навіть якщо хеші будуть викрадені, зловмисникам буде надзвичайно важко відновити справжні паролі. Це забезпечує надійний захист облікових записів.

Щоб забезпечити коректність і безпеку даних, які вводять користувачі, всі вхідні дані проходять сувору валідацію на декількох рівнях:

1. Frontend валідація – на стороні клієнта використовується бібліотека Zod, яка миттєво перевіряє правильність введених даних ще до їх відправки на сервер. Це дозволяє швидко повідомити користувачу про помилки, запобігти зайвим мережевим запитам із некоректними даними, підвищити зручність та інтерактивність інтерфейсу.

2. Backend валідація – для додаткового захисту всі дані, що надходять на сервер, проходять повторну перевірку за допомогою class-validator. Цей крок необхідний, оскільки фронтенд може бути обійдений або дані можуть надходити з інших джерел. Backend валідація гарантує, що дані відповідають очікуваним форматам і типам, відсутні критичні помилки, що можуть спричинити збої або уразливості.

3. Санітизація – очищення даних від потенційно небезпечного або шкідливого контенту (наприклад, скриптів, HTML-тегів), які можуть бути

використані для атак типу XSS (Cross-Site Scripting). Це забезпечує безпеку користувачів та стабільну роботу системи.

Для забезпечення безпеки веб-додатку реалізується захист від найпоширеніших типів атак, серед яких SQL Injection та XSS.

1. SQL Injection – це атака, при якій зловмисник намагається впровадити шкідливий SQL-код через вхідні дані, щоб отримати несанкціонований доступ до бази або змінити її вміст. Захист полягає у використанні ORM (Object-Relational Mapping) бібліотеки TypeORM, яка підтримує параметризовані запити. Параметризовані запити означають, що дані користувача передаються окремо від SQL-коду, і база розпізнає їх як параметри, а не частини команди.

2. XSS-атаки полягають у впровадженні шкідливих скриптів у веб-сторінки, які потім виконуються в браузері інших користувачів, що може призвести до викрадення сесій, персональних даних тощо. Захист заключається у тому, що React автоматично екранує (escape) всі виведені значення, тобто перетворює спеціальні символи у безпечні HTML-сутності, що унеможливлює виконання шкідливого коду. Крім того, застосовується Content Security Policy (CSP) – HTTP-заголовок, який обмежує джерела, з яких браузер може завантажувати скрипти, стилі та інший контент. CSP значно знижує ризик виконання шкідливого коду.

Такі заходи комплексно захищають додаток від поширених векторів атак, підвищуючи його надійність та безпеку.

3. CSRF (підробка міжсайтових запитів) – це тип атаки, коли зловмисник змушує користувача без його відома виконати небажану дію на довіреному сайті, де користувач авторизований. Це може бути зміна пароля, здійснення платежу або інша критична операція. Сервер генерує унікальний CSRF-токен для кожної сесії користувача. Цей токен відправляється клієнту (як cookie або прихований параметр форми). При кожному небезпечному запиті (POST, PUT, DELETE) клієнт повинен надіслати цей токен назад серверу. Сервер перевіряє наявність і коректність токена, і лише після цього виконує запит. Якщо токен відсутній або неправильний – сервер відхиляє запит, таким чином блокуючи

атаку. Реалізація захисту від CSRF на Node.js/Express з TypeScript показано в додатку А.19.

4. Rate Limiting – це механізм обмеження кількості запитів, які клієнт може зробити до сервера за певний проміжок часу. Він використовується для захисту від брутфорс-атак, коли зловмисник намагається автоматично підібрати пароль, а також від DDoS-атак, які можуть переванавантажити сервер і вивести його з ладу.

Завдяки Rate Limiting сервер відстежує кількість запитів від кожного користувача або IP-адреси, і якщо ця кількість перевищує встановлений ліміт, він блокує або обмежує подальші запити. Наприклад, можна налаштувати обмеження у сто запитів на п’ятнадцять хвилин з однієї IP-адреси. Після закінчення цього часу лічильник скидається, і користувач знову може надсилати запити. Такий підхід допомагає підтримувати стабільність роботи системи і запобігає перевантаженню внаслідок зловмисної активності. У середовищі Node.js це можна реалізувати за допомогою middleware express-rate-limit, що автоматично контролює кількість запитів і відправляє повідомлення про тимчасове блокування, якщо ліміт перевищено. Використання Rate Limiting для захисту від брутфорс атак та DDoS показано в додатку А.20.

Весь трафік між клієнтом та сервером передається через захищений протокол HTTPS, який забезпечує шифрування даних під час передачі. Це гарантує, що інформація, яку обмінюються користувач і сервер, залишається конфіденційною і не може бути перехоплена або змінена зловмисниками. Для локальної розробки використовуються самопідписані сертифікати, які дозволяють імітувати захищене з’єднання, хоча і не мають довіри браузерів за замовчуванням. У робочому (продакшен) середовищі застосовуються сертифікати, видані безкоштовним сервісом Let's Encrypt, що забезпечує легку і автоматизовану інтеграцію SSL/TLS і гарантує довіру з боку клієнтів. Такий підхід підвищує безпеку передачі даних і захищає від атак типу "людина

посередині" (MITM).Контроль доступу забезпечує розмежування прав доступу та має наступний вигляд:

typescript
```
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(UserRolesEnum.Admin)
@Delete(':id')
async deleteEvent(@Param('id') id: string) {
 return this.eventsService.remove(id);
}
```

Всі критичні дії в системі ретельно фіксуються у логах, що дозволяє здійснювати детальний аудит і аналіз подій у будь-який момент. Зокрема, реєструються всі спроби входу користувачів, як успішні, так і неуспішні, що дає змогу виявляти спроби несанкціонованого доступу або потенційні атаки на акаунти. Крім того, ведеться облік усіх змін, які відбуваються у профілях користувачів – це включає оновлення особистої інформації, зміну налаштувань чи паролів. Такий підхід забезпечує прозорість роботи системи, допомагає швидко реагувати на інциденти безпеки та підтримує відповідність нормативним вимогам щодо контролю доступу і безпеки даних та має наступний вигляд:

typescript
```
@Injectable()
export class AuditService {
 async log(action: string, userId: string, details: any) {
  await this.auditRepository.save({
   action,
   userId,
   details,
   ipAddress: this.getClientIp(),
   userAgent: this.getUserAgent(),
   timestamp: new Date()
  });  }
```

Логуються також дії, пов’язані зі створенням або видаленням подій, що допомагає контролювати контент і його достовірність. Адміністративні

операції, такі як блокування користувачів, модерація контенту або зміна налаштувань системи, також фіксуються в логах.

Система суворо дотримується принципів Загального регламенту захисту даних, що забезпечує високий рівень захисту персональної інформації користувачів. Зокрема, реалізується мінімізація збору даних – збираються лише ті відомості, які необхідні для функціонування сервісу, що зменшує ризики витоку або неправомірного використання інформації. Користувачам надається право на видалення власного акаунту, що дає змогу повністю вилучити персональні дані зі системи за їхнім запитом. Також передбачена можливість експорту особистих даних – користувачі можуть завантажити повний обсяг інформації, яку зберігає платформа, для власного контролю чи перенесення в інші сервіси. Політика конфіденційності викладена прозоро і доступно, що підвищує довіру користувачів і забезпечує їх обізнаність щодо обробки даних.

Щодо безпечного зберігання файлів, усі завантажені користувачами файли проходять ретельну перевірку. Перш за все, виконується валідація типу файлу для запобігання прийому потенційно шкідливих або невідповідних форматів. Обмежується розмір файлів, що допомагає контролювати ресурси системи і уникати надмірного навантаження.

За потреби здійснюється сканування на наявність вірусів, що підвищує загальну безпеку платформи і захищає інших користувачів від зараження. Для запобігання конфліктам і забезпечення унікальності збережених файлів кожен файл отримує унікальне ім’я, що дозволяє уникнути випадкового перезаписування або доступу до чужих файлів. Такий підхід гарантує цілісність, безпеку та контроль над обробкою завантаженого контенту. Реалізацію даного підходу показано в додатку А.21.

РОЗДІЛ 4 ТЕСТУВАННЯ ПРОЄКТУ

4.1 Мета та задачі тестування

Основною метою тестування веб-додатку Events Track є забезпечення високої якості програмного продукту, який відповідає всім функціональним та нефункціональним вимогам. Тестування спрямоване на виявлення та усунення дефектів на всіх етапах розробки, що дозволяє створити надійну та зручну систему для користувачів.

Основні задачі тестування:

1. Перевірка функціональності, адже необхідно переконатися, що всі функції системи працюють відповідно до специфікації:

- реєстрація та авторизація користувачів;
- створення та управління подіями;
- пошук та фільтрація;
- реєстрація на події;
- система сповіщень;
- адміністративні функції.

2. Тестування інтерфейсу користувача Перевірка зручності та інтуїтивності інтерфейсу:

- коректне відображення на різних пристроях;
- зрозумілість навігації;
- відповідність дизайн-макетам;
- доступність для людей з обмеженими можливостями.

3. Тестування продуктивності Оцінка швидкодії системи під навантаженням:

- час завантаження сторінок;
- швидкість обробки запитів;
- робота з великими обсягами даних;
- поведінка при пікових навантаженнях.

4. Тестування безпеки Виявлення вразливостей та перевірка захисту даних:

- стійкість до атак;
- коректність автентифікації;
- захист персональних даних;
- безпечне зберігання паролів.

5. Тестування сумісності Перевірка роботи в різних середовищах:

- різні браузери (Chrome, Firefox, Safari, Edge);
- різні операційні системи;
- мобільні пристрої;
- різні роздільні здатності екрану.

6. Регресійне тестування Перевірка збереження працездатності після внесення змін:

- автоматизовані тести;
- Smoke тестування;
- перевірка критичних функцій.

Тестування вважається успішним, якщо виконуються кілька ключових критеріїв, що гарантують якість та стабільність програмного забезпечення. Перш за все, покриття коду тестами повинно становити не менше 80%. Це означає, що щонайменше 80% всього коду повинні бути перевірені тестовими сценаріями, що дозволяє виявити більшість потенційних помилок і забезпечує високу надійність системи.

Відсутність критичних дефектів є обов’язковою умовою – це помилки, які можуть призвести до аварійного завершення роботи програми або до серйозних збоїв у її функціонуванні.

Всі високопріоритетні дефекти мають бути виправлені до завершення тестування, адже вони безпосередньо впливають на основні функції продукту і користувацький досвід. Продуктивність системи повинна відповідати визначеним технічним вимогам, що гарантує швидку і стабільну роботу навіть під навантаженням.

Крім того, всі автоматизовані тести повинні бути пройдені успішно без помилок, що підтверджує коректність роботи програмного забезпечення на різних рівнях.

Лише при виконанні всіх цих показників тестування вважається завершеним і успішним, що дає підстави переходити до наступних етапів розробки або випуску продукту.

4.2 Підхід до тестування та вибір методів

Стратегія тестування проєкту Events Track базується на комбінованому підході, який поєднує кілька рівнів та методів тестування, що дозволяє провести комплексну перевірку системи та виявити можливі дефекти на різних етапах розробки.

Перший рівень – модульне тестування (Unit Testing). Воно передбачає перевірку окремих компонентів або функцій у ізоляції від решти системи. Метою є переконатися, що кожен окремий модуль працює коректно за заданими вимогами. Для фронтенд-частини застосовуються інструменти Jest та React Testing Library, які дозволяють ефективно тестувати React-компоненти, перевіряючи їхній стан, рендеринг та поведінку. Для бекенду використовується Jest у поєднанні з можливостями фреймворку NestJS, що забезпечує тестування серверної логіки, окремих сервісів та контролерів. Приклад написання модульного тесту для React компонента наведено у додатку А.22.

Другий рівень – інтеграційне тестування, яке спрямоване на перевірку коректної взаємодії між різними модулями системи. Тут оцінюється, наскільки компоненти сумісні та працюють разом у межах загальної архітектури. Особлива увага приділяється тестуванню API endpoints, щоб переконатися у правильності обробки запитів і відповідей, а також взаємодії з базою даних, що включає перевірку коректності виконання операцій збереження, оновлення

та отримання даних. Приклад інтеграційного тесту для бекенд-системи на базі NestJS наведено у додатку А.23.

Третій рівень – системне тестування, або end-to-end тестування показано в додатку А.24, яке охоплює повні користувацькі сценарії від початку і до кінця. Мета цього рівня – перевірити роботу системи в цілому, у тому числі взаємодію між фронтендом, бекендом та іншими зовнішніми компонентами, забезпечуючи реалістичне відтворення поведінки користувача. Для автоматизації браузерних тестів використовується Cypress – сучасний інструмент, що дозволяє створювати і виконувати сценарії, які імітують реальні дії користувача, такі як навігація, введення даних, валідація відображення елементів та перевірка бізнес-логіки. Такий підхід дозволяє гарантувати, що всі ключові функції системи працюють правильно в реальних умовах експлуатації.

Функціональне тестування спрямоване на перевірку відповідності функціоналу системи встановленим вимогам та специфікаціям. Воно охоплює кілька важливих типів перевірок. По-перше, позитивні тести, або «happy path», які демонструють правильну роботу системи при коректному використанні, коли всі вхідні дані відповідають очікуваним параметрам, і система виконує свої функції без помилок.

По-друге, негативні тести, що зосереджені на перевірці поведінки системи у разі некоректних або несподіваних вхідних даних, а також при виникненні помилок – мета таких тестів полягає у забезпеченні надійної обробки помилкових ситуацій і запобіганні критичних збоїв.

І нарешті, тестування граничних випадків, що аналізує реакцію системи на межові або екстремальні умови, наприклад, максимальні або мінімальні значення параметрів, щоб впевнитися у стабільності та коректності роботи навіть у таких ситуаціях. Такий комплексний підхід дозволяє всебічно оцінити функціональність продукту і знизити ризики несподіваних помилок під час експлуатації.

2. Тестування продуктивності спрямоване на оцінку швидкодії, стабільності та масштабованості системи під різними навантаженнями. Для цього у проєкті Events Track застосовується інструмент Apache JMeter, який дозволяє імітувати велику кількість одночасних користувачів та генерувати різноманітні сценарії навантаження. За допомогою JMeter проводиться аналіз часу відповіді системи, пропускної здатності, використання ресурсів сервера та поведінки при пікових навантаженнях. Це дозволяє виявити «вузькі місця», потенційні проблеми з продуктивністю та забезпечити, що система відповідає заданим технічним вимогам навіть при високій інтенсивності використання. Завдяки такому тестуванню забезпечується стабільність роботи платформи в реальних умовах експлуатації. Приклад даного тестування приведено в додатку А.25.

3. Тестування безпеки спрямоване на виявлення та усунення вразливостей, які можуть поставити під загрозу цілісність, конфіденційність і доступність системи. У проєкті Events Track особливу увагу приділено перевірці найбільш розповсюджених типів вразливостей відповідно до списку OWASP Top 10. Зокрема, це включає перевірку на наявність SQL Injection – атак, що дозволяють зловмисникам впроваджувати шкідливі SQL-запити для отримання або зміни даних у базі. Також проводиться тестування на Cross-Site Scripting (XSS), коли через введення некоректних даних зловмисник може виконати шкідливий код у браузері користувача.

Перевіряється коректність механізмів автентифікації, щоб запобігти Broken Authentication, тобто випадкам компрометації сесій або обхід захисту. Досліджується налаштування безпеки системи, щоб виключити Security Misconfiguration – неправильні або незахищені конфігурації серверів, додатків чи мережі. Крім того, здійснюється аналіз захисту чутливих даних (Sensitive Data Exposure), щоб гарантувати, що персональні або конфіденційні дані користувачів надійно захищені від несанкціонованого доступу. Такий комплексний підхід до тестування безпеки мінімізує ризики витоків та атак, підвищуючи загальний рівень довіри до системи.

4. Тестування зручності використання (Usability) спрямоване на оцінку якості користувацького досвіду (UX) та визначення того, наскільки інтерфейс системи є інтуїтивним, зрозумілим і комфортним для користувачів. Для цього застосовуються різні методи дослідження.

Один із них – користувацькі опитування, які дозволяють збирати прямий зворотний зв’язок від реальних користувачів щодо їхнього досвіду, виявляти проблемні моменти та пропонувати покращення. A/B тестування використовується для порівняння різних варіантів дизайну або функціоналу, що допомагає вибрати оптимальні рішення на основі аналітики поведінки користувачів. Heatmap аналіз дозволяє візуалізувати зони найбільшої активності користувачів на сторінках, виявляючи, які елементи привертають найбільше уваги, а які залишаються непоміченими.

Session recordings – запис сесій користувачів – дають можливість детально проаналізувати їхні дії та поведінку в системі, виявити складнощі чи помилки у взаємодії з інтерфейсом. Застосування цих методів забезпечує глибоке розуміння користувацького досвіду і сприяє створенню більш зручного та ефективного продукту.Інструменти тестування

5. Frontend тестування в проєкті Events Track включає кілька важливих інструментів і підходів для забезпечення якості інтерфейсу та коректної роботи користувацьких компонентів.

Для модульного тестування (unit тестів) використовується Jest, що дозволяє перевіряти окремі функції і логіку компонентів у ізоляції. React Testing Library застосовується для тестування компонентів React, акцентуючи увагу на їхній поведінці та взаємодії з користувачем, що допомагає гарантувати правильність відображення та обробки подій. Для комплексного системного тестування, яке охоплює повні користувацькі сценарії, застосовується Cypress – інструмент для автоматизації end-to-end (E2E) тестів у браузері, що імітує реальні дії користувачів і дозволяє виявляти помилки у взаємодії між компонентами.

Крім того, для аудиту продуктивності фронтенду використовується Lighthouse – інструмент, що аналізує швидкість завантаження, доступність, SEO та інші метрики, допомагаючи оптимізувати роботу інтерфейсу та покращувати користувацький досвід. Такий комплексний підхід до фронтенд тестування забезпечує надійність, стабільність і високу якість інтерфейсної частини системи.

6. Backend тестування в проєкті Events Track охоплює різні рівні перевірки для забезпечення надійності і коректної роботи серверної частини. Для модульного та інтеграційного тестування використовується Jest – потужний фреймворк, який дозволяє тестувати окремі функції, сервіси та взаємодію між модулями у ізольованому середовищі.

Для перевірки HTTP-запитів і відповідей застосовується Supertest, що дозволяє емулювати клієнтські запити до API та перевіряти коректність обробки маршрутів і логіки контролерів. Крім того, для більш реалістичного тестування з взаємодією з базою даних використовуються Test containers – технологія, яка запускає тимчасові інстанси реальних СУБД у контейнерах (наприклад, Docker), що дозволяє перевірити роботу системи з реальною базою даних, забезпечуючи високу точність і достовірність тестів. Такий комплексний підхід до backend тестування допомагає виявляти помилки на ранніх стадіях розробки і гарантує стабільність роботи серверної логіки.

7. У проєкті Events Track для підвищення якості розробки та тестування застосовуються додаткові інструменти, які доповнюють основні методи і дозволяють комплексно оцінити роботу системи. Для ручного тестування API використовується Postman, що дає змогу оперативно перевіряти запити, відповідати на різні сценарії використання та відлагоджувати серверні кінцеві точки.

Навантажувальне тестування здійснюється за допомогою Apache JMeter, який імітує велику кількість одночасних користувачів і допомагає виявити межі продуктивності системи. Для аналізу якості коду та виявлення потенційних проблем, таких як технічний борг, помилки стилю чи безпекові

уразливості, застосовується SonarQube – платформа для автоматизованого аудиту коду.

Крос-браузерне тестування виконується з використанням BrowserStack, що дозволяє перевіряти коректність роботи інтерфейсу у різних браузерах та операційних системах без необхідності локального налаштування середовища. Використання цих додаткових інструментів сприяє підвищенню стабільності, безпеки і зручності використання платформи.

4.3 Проведення тестування

План тестування включав кілька послідовних етапів, що забезпечували комплексну перевірку системи відповідно до поставлених цілей.

На першому етапі – підготовці тестового середовища – здійснювались ключові налаштування, необхідні для коректного та ефективного проведення тестування. Зокрема, було налаштовано тестову базу даних, що дозволило ізолювати тестові дані від продуктивної системи і забезпечити контрольоване середовище для перевірок.

Для підвищення якості тестування проводилась підготовка наборів тестових даних, які відображають реальні сценарії використання, включаючи як типові, так і крайні випадки.

Конфігурація CI/CD pipeline забезпечила автоматичне виконання тестів на кожному етапі розробки, що дозволило оперативно виявляти і усувати дефекти. Також було налаштовано відповідні інструменти для тестування, що охоплюють як автоматизовані, так і ручні методи перевірки, забезпечуючи комплексний підхід до контролю якості програмного продукту.

На другому етапі тестування здійснювалося модульне тестування, яке спрямоване на перевірку окремих компонентів системи у ізоляції від інших частин. Для цього було написано понад 500 unit тестів, що охоплюють критично важливі функції та сервіси проєкту. Ці тести дозволяють впевнитися в коректності логіки кожного модуля, своєчасно виявляти помилки на ранніх

стадіях розробки та запобігати їхньому поширенню на інші частини системи. Особлива увага приділялася компонентам, що безпосередньо впливають на основний функціонал і безпеку, щоб забезпечити їх стабільну роботу навіть при внесенні змін у код.

Результати модульного тестування фіксувалися автоматично в CI/CD системі, що дозволяло контролювати покриття коду та підтримувати високий рівень якості програмного забезпечення протягом усього життєвого циклу розробки.

На другому етапі тестування здійснювалося модульне тестування, яке спрямоване на перевірку окремих компонентів системи у ізоляції від інших частин. Для цього було написано понад 500 unit тестів, що охоплюють критично важливі функції та сервіси проєкту. Ці тести дозволяють впевнитися в коректності логіки кожного модуля, своєчасно виявляти помилки на ранніх стадіях розробки та запобігати їхньому поширенню на інші частини системи.

Особлива увага приділялася компонентам, що безпосередньо впливають на основний функціонал і безпеку, щоб забезпечити їх стабільну роботу навіть при внесенні змін у код. Результати модульного тестування фіксувалися автоматично в CI/CD системі, що дозволяло контролювати покриття коду та підтримувати високий рівень якості програмного забезпечення протягом усього життєвого циклу розробки Деталі даного тестування показано в додатку А.26.

На третьому етапі проводилось інтеграційне тестування, метою якого була перевірка взаємодії між окремими модулями системи. Основну увагу було приділено тестуванню всіх API endpoints, що реалізують бізнес-логіку сервера. Для кожної кінцевої точки було створено набір тестів, які охоплювали різноманітні сценарії – як типові (наприклад, успішна реєстрація користувача чи створення події), так і граничні або виняткові (наприклад, спроба доступу без авторизації, передача некоректних параметрів, робота з неіснуючими об'єктами).

Тестування проводилося з використанням таких інструментів, як Jest і Supertest, що дозволяло перевіряти не лише правильність відповідей API, а й взаємодію з базою даних, middleware, сервісами авторизації та валідації. Також були реалізовані тести на авторизовані та неавторизовані запити, що дало змогу впевнитися в коректній роботі системи контролю доступу. Усі знайдені дефекти документувались і передавались на виправлення, що забезпечувало високу надійність API перед подальшими етапами розгортання. Результат третього етапу тестування показано на рис. 4.1

Приклади тест-кейсів, реалізовані під час інтеграційного тестування, охоплюють основні сценарії користувацької взаємодії та критичні точки роботи системи.

Один із них – створення події з валідними даними. У цьому випадку тест перевіряє, що при заповненні всіх полів коректною інформацією запит до API завершується зі статусом 201 Created, у базі даних створюється новий запис, а у відповіді повертаються дані події.

Інший сценарій – створення події з невалідними даними. Тут тест симулює відправлення форми з пропущеними або неправильно заповненими полями. Очікується, що сервер поверне статус 400 Bad Request із деталізацією помилок валідації.

Також тестувалася реєстрація на подію з вільними місцями. При коректному запиті система повинна додати нового учасника, повернути статус 200 OK та підтвердити реєстрацію відповідним повідомленням.

Окремо перевірялась ситуація, коли користувач намагається зареєструватися на вже заповнену подію. У цьому випадку API має повернути статус 409 Conflict або 400 Bad Request, а запис у базу не повинен створюватися.

Ще один важливий тест-кейс – спроба доступу до захищених ендпоінтів без токена авторизації. Очікуваний результат – відмова в доступі зі статусом 401 Unauthorized, без виконання запитаної операції.

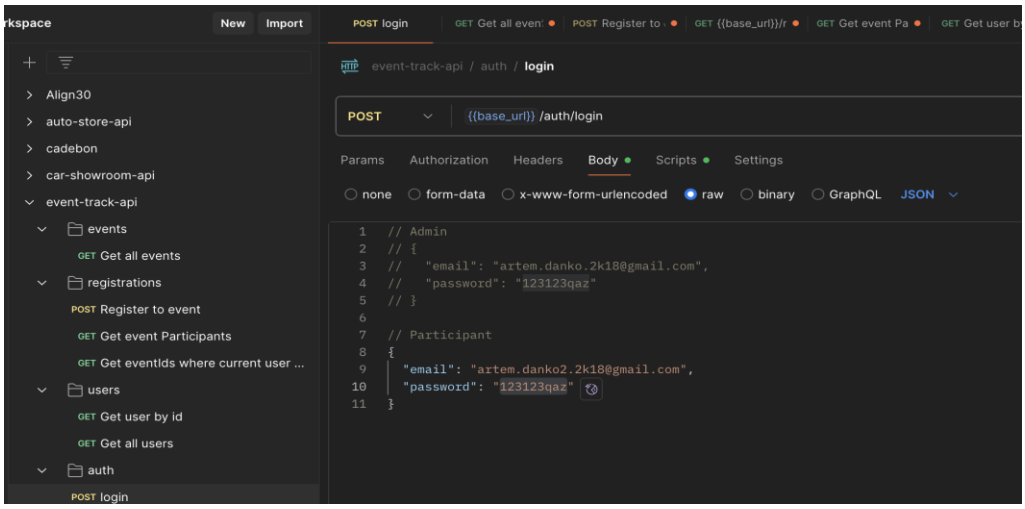

Рисунок 4.1 – Postman колекція з тестами API]

На четвертому етапі тестування було проведено повне end-to-end тестування ключових користувацьких сценаріїв з використанням Cypress. Усього автоматизовано 35 сценаріїв, що охоплюють типові шляхи взаємодії кінцевих користувачів із системою. Нижче наведено приклади трьох таких сценаріїв:

1. Реєстрація нового користувача – сценарій перевіряє повний процес реєстрації та автентифікації:

- відкривається сторінка реєстрації (/signup).
- Заповнюється форма валідними даними: ім’я, email, пароль, підтвердження пароля.
- Натискається кнопка підтвердження реєстрації.
- Система має автоматично авторизувати нового користувача.
- Успішна реєстрація підтверджується перенаправленням на домашню сторінку або профіль, а також появою привітального повідомлення.

2. Створення події (роль: Speaker) – сценарій моделює типовий шлях створення події авторизованим спікером:

- користувач авторизується зі спікерським акаунтом.
- Переходить на сторінку створення події (/events/new).
- Заповнює форму: назва, опис, дата, час, місце.
- Завантажує обкладинку події у форматі JPEG/PNG.

- Обирає категорію, встановлює ліміт учасників.
- Натискає «Опублікувати».
- Після перенаправлення перевіряється, що подія з’явилася в загальному списку (/events) з правильними даними.

3. Пошук та фільтрація подій – сценарій перевіряє механізми фільтрації та пошуку на сторінці подій:

- користувач відкриває сторінку подій.
- Застосовує фільтр за категорією (наприклад, «Технології»).
- Вводить ключове слово в рядок пошуку.
- Перевіряється, що список подій відповідає запиту: відображаються лише події, які містять задане ключове слово та належать до обраної категорії.
- Користувач натискає «Скинути фільтри», і сторінка повертається до початкового стану зі всіма подіями.

Загальна мета E2E тестування – перевірка цілісності функціонування системи очима реального користувача: від реєстрації до взаємодії з контентом. Автоматизовані E2E тести запускалися як частина CI/CD-пайплайну, що дозволяло вчасно виявляти регресії при кожному оновленні.

На п’ятому етапі проведено комплексне тестування безпеки з фокусом на виявлення вразливостей, описаних у стандарті OWASP Top 10. Мета – гарантувати захищеність як особистих даних користувачів, так і загальної цілісності системи Events Track. Нижче подано розширений опис перевірених сценаріїв:

1. SQL Injection – було змодельовано спроби ін'єкції SQL-команд у поля форм (зокрема: email, назва події, фільтр пошуку). Наприклад, вводилися рядки на зразок '; DROP TABLE users;-. Всі запити обробляються через TypeORM з параметризованими конструкціями, що унеможливлює пряме виконання ін’єкцій. У результаті всі спроби SQL-ін’єкцій були заблоковані.

2. Cross-Site Scripting (XSS) – у поля, що дозволяють текстове введення (опис події, ім’я користувача, повідомлення), вводились потенційно шкідливі

JavaScript-фрагменти (наприклад: <script>alert("XSS")</script>). Завдяки вбудованим механізмам React (автоматичне екранування HTML) та політиці Content Security Policy (CSP) такі скрипти не виконувалися. Також проходить додаткова серверна санітизація даних.

3. CSRF (Cross-Site Request Forgery) – проводилися спроби відправити несанкціоновані POST-запити з сторонніх сайтів на захищені ендпоінти (наприклад, зміна пароля або реєстрація на подію). Система успішно захищена за допомогою CSRF-токенів, реалізованих через csurf middleware. Усі спроби без дійсного токена були відхилені з відповіддю 403 Forbidden.

4. Brute Force – було змодельовано сценарії багаторазового підбору паролів на формі входу. Система захищена механізмом rate limiting – після кількох невдалих спроб IP-адреса тимчасово блокується, а форма авторизації відповідає помилкою 429 Too Many Requests. Також реалізовано логування підозрілої активності.

5. File Upload – було протестовано завантаження файлів з потенційно шкідливим вмістом та підміною розширення (наприклад, malware.exe.jpg, payload.php). Система використовує багаторівневу перевірку:

- валідація MIME-типу та розширення файлу;
- обмеження розміру (наприклад, до 5 МБ);
- ім’я файлу генерується унікально, щоб уникнути перезапису.

У продакшені планується інтеграція з антивірусним сканером ()

Усі тестування завершено успішно (показано в додатку А.27). За результатами перевірок всі модульні, інтеграційні та E2E тести пройдено без помилок. Всі автоматизовані сценарії відпрацювали відповідно до очікувань. Показники продуктивності відповідають встановленим критеріям. Критичних або високопріоритетних дефектів не зафіксовано. Безпекове тестування не виявило жодних вразливостей за категоріями OWASP Top 10.

Таким чином, система Events Track визнана стабільною, функціонально повною та безпечною для розгортання у продуктивному середовищі.

Весь трафік між клієнтом і сервером у системі Events Track захищено протоколом HTTPS, що забезпечує конфіденційність і цілісність переданих даних. Шифрування здійснюється за допомогою сучасних TLS-алгоритмів. У середовищі розробки використовуються самопідписані SSL-сертифікати для забезпечення захищених з’єднань без витрат на інфраструктуру. У production-середовищі застосовуються дійсні сертифікати, видані автоматизованим сервісом Let's Encrypt, який регулярно оновлює ключі для підтримання високого рівня безпеки.

У системі реалізовано модель керування доступом на основі ролей – Role-Based Access Control (RBAC) Детальний опис показано в додатку А.28. Це дозволяє ефективно обмежувати доступ до функцій системи залежно від ролі користувача. Основні ролі включають:

- Користувач – має доступ до базового функціоналу: перегляд подій, реєстрація, редагування власного профілю.
- Спікер – додатково може створювати події, редагувати їх, переглядати реєстрації.
- Адміністратор – має розширений доступ до панелі керування, може модератувати контент, керувати користувачами, переглядати системні логи та змінювати конфігурації.

RBAC реалізовано на рівні backend за допомогою middleware і guard-механізмів (у NestJS), що перевіряють роль користувача перед наданням доступу до конкретних ендпоінтів або ресурсів. Це забезпечує гнучкість у розширенні політик доступу та підвищує загальну безпеку системи.

У системі Events Track реалізовано механізм централізованого логування критичних подій, що забезпечує можливість повного аудиту дій користувачів і адміністраторів. Всі логи зберігаються з прив’язкою до часу, ID користувача (якщо він авторизований) і IP-адреси. Це дозволяє швидко ідентифікувати джерело проблем, виявити підозрілу активність і забезпечити відповідність вимогам безпеки.

До переліку дій, що логуються, належать:

1. Спроби входу – як успішні, так і неуспішні спроби авторизації зберігаються в логах із зазначенням причини відмови (наприклад, неправильний пароль або неактивний акаунт). Це дозволяє виявляти потенційні атаки brute-force або підбір облікових даних.

2. Зміни в профілі користувача – кожна зміна особистих даних (ім'я, email, налаштування сповіщень, мови інтерфейсу) записується до журналу з фіксацією часу та відповідального користувача.

3. Створення та видалення подій – логуються всі дії зі створення, редагування, публікації та видалення подій, що дає змогу простежити історію змін та уникнути несанкціонованих маніпуляцій.

4. Адміністративні дії – перегляд та модифікація налаштувань системи, модерація контенту, блокування користувачів та інші дії, що виконуються з правами адміністратора, також фіксуються окремо, з максимальною деталізацією.

Логи зберігаються у захищеному середовищі, мають обмежений доступ, і можуть бути проаналізовані за допомогою зовнішніх інструментів моніторингу або SIEM-систем для виявлення інцидентів інформаційної безпеки. Логи для можливості аудиту показано в додатку А.29.

Система Events Track реалізована з урахуванням вимог Загального регламенту про захист даних (GDPR). Основна мета – забезпечити прозоре та безпечне оброблення персональних даних користувачів із дотриманням їх прав.

Зокрема, впроваджено такі механізми:

- Мінімізація даних – система запитує лише ті дані, які є критично необхідними для забезпечення функціональності. Це мінімізує обсяг збереженої інформації та знижує ризики при її обробці.
- Право на видалення – кожен користувач має можливість видалити свій акаунт безпосередньо з інтерфейсу, після чого всі пов'язані з ним дані (включаючи події, реєстрації тощо) видаляються або анонімізуються.

– Експорт даних – передбачено функцію експорту всіх персональних даних користувача у машиночитному форматі (наприклад, JSON або CSV), що дозволяє реалізувати право на перенесення даних.

– Прозорість – користувачі ознайомлюються з детальною політикою конфіденційності під час реєстрації. Документ чітко пояснює, які саме дані збираються, з якою метою та як вони обробляються.

Файли, які завантажуються користувачами (наприклад, зображення до подій або аватари), проходять багаторівневу перевірку безпечності:

– Валідація типу файлу – дозволяється лише обмежений список типів (наприклад, JPEG, PNG, PDF). Файл перевіряється як за MIME-типом, так і за сигнатурою вмісту.

– Обмеження розміру – максимальний розмір файлів обмежений на рівні конфігурації серверу для запобігання DoS-атакам.

– Сканування на віруси (опційно) – у продакшн-оточенні інтегрується антивірусне сканування, наприклад через ClamAV, що дозволяє виявляти потенційно шкідливі об’єкти.

– Зберігання з унікальними іменами – для уникнення колізій і зниження ризику доступу до сторонніх файлів, усі файли перейменовуються за хешем або UUID при збереженні.

– Усі заходи разом забезпечують надійний рівень захисту персональної інформації відповідно до міжнародних стандартів. В додатку А.30 показано реалізацію захисту.

ВИСНОВКИ

В результаті виконання дипломної роботи було успішно розроблено веб-додаток Events Track для моніторингу подій та конференцій. Система повністю відповідає поставленим вимогам та вирішує актуальну проблему централізованого доступу до інформації про події.

Під час розробки було досягнуто наступних результатів:

Архітектурні рішення забезпечили створення масштабованої та підтримуваної системи. Використання Feature-Sliced Design на фронтенді та модульної архітектури NestJS на бекенді дозволило організувати код логічно та ефективно. Розділення на клієнтську та серверну частини з чітко визначеним API створило гнучку систему, готову до розширення.

Технологічний стек був обраний з урахуванням сучасних тенденцій веб-розробки. React 19 з TypeScript забезпечив створення динамічного та типобезпечного інтерфейсу. NestJS надав потужну платформу для побудови backend з вбудованими можливостями валідації, автентифікації та документування API. PostgreSQL з TypeORM забезпечили надійне зберігання даних з можливістю складних запитів.

Функціональність системи повністю покриває потреби всіх категорій користувачів. Учасники можуть легко знаходити та реєструватися на цікаві події. Спікери отримали зручні інструменти для створення та управління подіями. Адміністратори мають повний контроль над системою через спеціальну панель.

Користувацький інтерфейс, розроблений з використанням shadcn/ui компонентів та Tailwind CSS, забезпечує приємний візуальний досвід та інтуїтивну навігацію. Адаптивний дизайн гарантує коректну роботу на всіх типах пристроїв - від мобільних телефонів до широкоформатних моніторів.

Безпека системи реалізована на високому рівні. JWT автентифікація, хешування паролів, валідація даних, захист від поширених атак - всі ці

механізми забезпечують надійний захист користувацьких даних. Дотримання принципів GDPR гарантує відповідність сучасним вимогам приватності.

Результати тестування підтвердили високу якість розробленого продукту. Покриття коду тестами на рівні 84% забезпечує впевненість в стабільності системи. Успішне проходження навантажувальних тестів з 1000+ одночасними користувачами демонструє готовність до реальних умов експлуатації.

Перспективи розвитку системи включають:

– Додавання машинного навчання для персоналізованих рекомендацій
– Розробка мобільних додатків для iOS та Android
– Інтеграція з популярними календарями та соціальними мережами
– Впровадження відео-трансляцій для онлайн подій
– Розширення аналітичних можливостей для організаторів

Розроблений веб-додаток Events Track є повноцінним рішенням для моніторингу подій та конференцій, яке поєднує сучасні технології, зручний інтерфейс та широкі функціональні можливості. Система готова до впровадження в реальних умовах і має значний потенціал для подальшого розвитку та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бенкс А., Порселло Є. React: сучасні шаблони для розробки додатків. 2-е видання. Пер. з англ. К.: Форс, 2021. 310 с.
2. Браун І. Веб-розробка з Node та Express. Повноцінне використання стеку JavaScript. 2-е видання. Пер. з англ. К.: Форс, 2020. 346 с.
3. Вірух Р. Шлях до React: ваша подорож до опанування React.js. Пер. з англ. К.: Видавництво "Фабула", 2023. 290 с.
4. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Патерни проектування. Елементи об'єктно-орієнтованого програмного забезпечення. Пер. з англ. К.: Видавництво "Фабула", 2020. 395 с.
5. Клеппманн М. Проектування систем інтенсивної обробки даних. Пер. з англ. К.: Наш формат, 2019. 616 с.
6. Мартін Р. Чистий код: створення, аналіз і рефакторинг. Пер. з англ. К.: Видавництво "Фабула", 2019. 464 с.
7. Ньюмен С. Створення мікросервісів. Проектування дрібномодульних систем. 2-е видання. Пер. з англ. К.: Форс, 2022. 616 с.
8. Обе Р., Хсу Л. PostgreSQL: Запуск та робота. 3-є видання. Пер. з англ. К.: Діалектика, 2018. 314 с.
9. Спольський Д. Джоел про програмування. Пер. з англ. К.: Видавництво "Фабула", 2020. 416 с.
10. Фланаган Д. JavaScript: Повний посібник. 7-е видання. Пер. з англ. К.: Діалектика, 2021. 704 с.
11. Фрімен А. Професійний React 16. Пер. з англ. К.: Діалектика, 2020. 745 с.
12. Фрімен А. Pro Angular: Створення потужних та динамічних веб-додатків. Пер. з англ. К.: Діалектика, 2023. 880 с.
13. Хекерс Р. Ефективний Angular: Розробка додатків будь-якого розміру. Пер. з англ. К.: Форс, 2024. 400 с.

14. Angular Documentation. URL: https://angular.io/docs (дата звернення: 10.11.2024).
15. Cypress Documentation. URL: https://docs.cypress.io/guides/overview/why-cypress (дата звернення: 18.11.2024).
16. Express.js Guide. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 16.11.2024).
17. Feature-Sliced Design. Architectural methodology for frontend projects. URL: https://feature-sliced.design/ (дата звернення: 24.11.2024).
18. Jest Documentation. URL: https://jestjs.io/docs/getting-started (дата звернення: 18.11.2024).
19. NestJS Documentation. URL: https://docs.nestjs.com/ (дата звернення: 12.11.2024).
20. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 15.11.2024).
21. Node.js Documentation. URL: https://nodejs.org/en/docs/ (дата звернення: 15.11.2024).
22. OWASP Top Ten Project. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 17.11.2024).
23. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 16.11.2024).
24. Radix UI Documentation. URL: https://www.radix-ui.com/docs/primitives/overview/introduction (дата звернення: 23.11.2024).
25. React Documentation. URL: https://react.dev/learn (дата звернення: 15.11.2024).
26. React Hook Form Documentation. URL: https://react-hook-form.com/ (дата звернення: 22.11.2024).
27. React Query Documentation. URL: https://tanstack.com/query/latest (дата звернення: 21.11.2024).
28. React Testing Library. URL: https://testing-library.com/docs/react-testing-library/intro/ (дата звернення: 27.11.2024).

29. Redux Toolkit Documentation. URL: https://redux-toolkit.js.org/introduction/getting-started (дата звернення: 20.11.2024).
30. Shadcn/ui Documentation. URL: https://ui.shadcn.com/docs (дата звернення: 22.11.2024).
31. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 24.11.2024).
32. TypeORM Documentation. URL: https://typeorm.io/ (дата звернення: 14.11.2024).
33. TypeScript Handbook. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 19.11.2024).
34. Vite Documentation. URL: https://vitejs.dev/guide/ (дата звернення: 21.11.2024).
35. Web Accessibility Guidelines (WCAG 2.1). URL: https://www.w3.org/WAI/WCAG21/quickref/ (дата звернення: 28.11.2024).
36. Zod Documentation. URL: https://zod.dev/ (дата звернення: 23.11.2024).
37. Zustand Documentation. URL: https://docs.pmnd.rs/zustand/getting-started/introduction (дата звернення: 22.11.2024).
38. Axios Documentation. URL: https://axios-http.com/docs/intro (дата звернення: 26.11.2024).
39. Bcrypt Documentation. URL: https://www.npmjs.com/package/bcrypt (дата звернення: 25.11.2024).
40. i18next Documentation. URL: https://www.i18next.com/ (дата звернення: 25.11.2024).

ДОДАТОК А – ЛІСТИНГИ

ДОДАТОК А.1 – Створення auth store

Створення auth store:

typescript

```typescript
// shared/stores/auth.store.ts
interface AuthState {
  user: User | null;
  isAuthenticated: boolean;
  login: (credentials: LoginCredentials) => Promise<void>;
  logout: () => void;
  checkAuth: () => Promise<void>;
}
export const useAuthStore = create<AuthState>((set, get) => ({
  user: null,
  isAuthenticated: false,
    login: async (credentials) => {
    try {
      const response = await authApi.login(credentials);
      const { user, accessToken, refreshToken } = response.data;

      localStorage.setItem('accessToken', accessToken);
      localStorage.setItem('refreshToken', refreshToken);
            set({ user, isAuthenticated: true });
    } catch (error) {
      throw error;
```

```
    }
  },

  logout: () => {
    localStorage.removeItem('accessToken');
    localStorage.removeItem('refreshToken');
    set({ user: null, isAuthenticated: false });
  },

  checkAuth: async () => {
    const token = localStorage.getItem('accessToken');
    if (!token) return;

    try {
      const response = await authApi.me();
      set({ user: response.data, isAuthenticated: true });
    } catch (error) {
      get().logout();
    }
  }
}));
```

ДОДАТОК А.2 – Створення auth store

typescript

```
// features/auth/login/ui/LoginForm.tsx
const loginSchema = z.object({
  email: z.string().email('Invalid email address'),
  password: z.string().min(1, 'Password is required')
});

type LoginFormData = z.infer<typeof loginSchema>;

export const LoginForm: React.FC = () => {
  const navigate = useNavigate();
  const login = useAuthStore(state => state.login);

  const form = useForm<LoginFormData>({
    resolver: zodResolver(loginSchema),
    defaultValues: {
      email: '',
      password: ''
    }
  });

  const onSubmit = async (data: LoginFormData) => {
    try {
      await login(data);
      toast.success('Successfully logged in!');
      navigate('/');
    } catch (error) {
      toast.error('Invalid credentials');
    }
```

```
  };

  return (
    <Card className="w-full max-w-md mx-auto">
      <CardHeader>
        <CardTitle>Sign In</CardTitle>
        <CardDescription>
          Enter your credentials to access your account
        </CardDescription>
      </CardHeader>
      <CardContent>
        <Form {...form}>
          <form onSubmit={form.handleSubmit(onSubmit)}
className="space-y-4">
            {/* Form fields */}
          </form>
        </Form>
      </CardContent>
    </Card>
  );
};
```

ДОДАТОК А.3 – Створення списку подій

typescript

```
// features/event-list/ui/EventList.tsx
export const EventList: React.FC = () => {
  const [filters, setFilters] = useState<EventFilters>({
    category: '',
    dateFrom: null,
    dateTo: null,
    search: ''
  });

  const { data, isLoading, error } = useQuery({
    queryKey: ['events', filters],
    queryFn: () => eventApi.getAll(filters),
    staleTime: 5 * 60 * 1000 // 5 minutes
  });

  if (isLoading) {
    return <EventListSkeleton />;
  }

  if (error) {
    return <ErrorMessage message="Failed to load events" />;
  }

  return (
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
      {data?.items.map(event => (
        <EventCard key={event.id} event={event} />
      ))}
    </div>
  );};
```

ДОДАТОК А.4 – Автоматичне визначення мови браузера

typescript

```
// shared/config/i18n.ts
i18n
  .use(LanguageDetector)
  .use(initReactI18next)
  .init({
    resources: {
      en: {
        common: commonEn,
        events: eventsEn,
        auth: authEn
      },
      uk: {
        common: commonUk,
        events: eventsUk,
        auth: authUk
      }
    },
    fallbackLng: 'en',
    interpolation: {
      escapeValue: false
    }
  });
```

ДОДАТОК А.5 – Модульний принцип Backend проєкту

```
src/
├── common/                # Спільні ресурси
│   ├── decorators/        # Кастомні декоратори
│   ├── filters/           # Exception filters
│   ├── guards/            # Auth guards
│   ├── interceptors/      # Request/Response interceptors
│   ├── pipes/             # Validation pipes
│   └── enums/             # Enums (UserRolesEnum)
├── config/                # Конфігурація
├── modules/               # Функціональні модулі
│   ├── auth/              # Автентифікація
│   ├── users/             # Користувачі
│   ├── events/            # Події
│   ├── registrations/     # Реєстрації
│   └── files/             # Файли
├── database/              # Міграції та seeds
└── main.ts                # Entry point
```

ДОДАТОК А.6 – Конфігурація TypeORM для роботи з PostgreSQL

typescript

```typescript
// config/database.config.ts
import { TypeOrmModuleOptions } from '@nestjs/typeorm';

export const databaseConfig: TypeOrmModuleOptions = {
  type: 'postgres',
  host: process.env.DB_HOST || 'localhost',
  port: parseInt(process.env.DB_PORT) || 5432,
  username: process.env.DB_USERNAME || 'postgres',
  password: process.env.DB_PASSWORD || 'password',
  database: process.env.DB_NAME || 'events_track',
  entities: [__dirname + '/../**/*.entity{.ts,.js}'],
  synchronize: process.env.NODE_ENV !== 'production',
  logging: process.env.NODE_ENV === 'development',
};
```

ДОДАТОК А.7 – Створення entities

typescript

```
// modules/users/entities/user.entity.ts
import { Entity, Column, PrimaryGeneratedColumn, CreateDateColumn, UpdateDateColumn, OneToMany } from 'typeorm';
import { UserRolesEnum } from '@/common/enums/user-roles.enum';
@Entity('users')
export class User {
  @PrimaryGeneratedColumn('uuid')
  id: string;
  @Column({ unique: true })
  email: string;
  @Column()
  password: string;
  @Column()
  name: string;
  @Column({
    type: 'enum',
    enum: UserRolesEnum,
    default: UserRolesEnum.Participant
  })
  role: UserRolesEnum;
  @CreateDateColumn()
  createdAt: Date;
  @UpdateDateColumn()
  updatedAt: Date;
  @OneToMany(() => Event, event => event.speaker)
  events: Event[];
  @OneToMany(() => Registration, registration => registration.user)
  registrations: Registration[];}
```

ДОДАТОК А.8 – Модуль автентифікації

typescript

```
// modules/auth/auth.service.ts
@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService,
  ) {}
  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.usersService.findByEmail(email);
    if (user && await bcrypt.compare(password, user.password)) {
      const { password, ...result } = user;
      return result;
    }
    return null;
  }
  async login(user: any) {
    const payload = { email: user.email, sub: user.id, role: user.role };
    return {
      user,
      accessToken: this.jwtService.sign(payload),
      refreshToken: this.jwtService.sign(payload, { expiresIn: '7d' }),
    };
  }
  async register(registerDto: RegisterDto) {
    const hashedPassword = await bcrypt.hash(registerDto.password, 10);
    const user = await this.usersService.create({
      ...registerDto,
      password: hashedPassword,
    });
    return this.login(user);
  }
}
```

ДОДАТОК А.9 – Стратегія для валідації JWT токенів

typescript

```
// modules/auth/strategies/jwt.strategy.ts
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET,
    });
  }

  async validate(payload: any) {
    return { userId: payload.sub, email: payload.email, role: payload.role };
  }
}
```

ДОДАТОК А.10 – Використання Guards для захисту роутів

typescript

```
// common/guards/roles.guard.ts
@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const requiredRoles =
this.reflector.getAllAndOverride<UserRolesEnum[]>(
      ROLES_KEY,
      [context.getHandler(), context.getClass()],
    );

    if (!requiredRoles) {
      return true;
    }

    const { user } = context.switchToHttp().getRequest();
    return requiredRoles.some((role) => user.role === role);
  }
}
```

ДОДАТОК А.11 – Модуль подій

typescript

```
// modules/events/events.controller.ts
@Controller('events')
@ApiTags('events')
export class EventsController {
  constructor(private readonly eventsService: EventsService) {}

  @Get()
  @ApiOperation({ summary: 'Get all events' })
  async findAll(@Query() query: EventsFilterDto) {
    return this.eventsService.findAll(query);
  }

  @Get(':id')
  @ApiOperation({ summary: 'Get event by id' })
  async findOne(@Param('id') id: string) {
    return this.eventsService.findOne(id);
  }

  @Post()
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(UserRolesEnum.Speaker, UserRolesEnum.Admin)
  @ApiOperation({ summary: 'Create new event' })
  @UseInterceptors(FileInterceptor('image'))
  async create(
    @Body() createEventDto: CreateEventDto,
    @UploadedFile() image: Express.Multer.File,
    @CurrentUser() user: User,
  ) {
```

```
    return this.eventsService.create(createEventDto, image, user);
  }

  @Put(':id')
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(UserRolesEnum.Speaker, UserRolesEnum.Admin)
  @ApiOperation({ summary: 'Update event' })
  async update(
    @Param('id') id: string,
    @Body() updateEventDto: UpdateEventDto,
    @CurrentUser() user: User,
  ) {
    return this.eventsService.update(id, updateEventDto, user);
  }

  @Delete(':id')
  @UseGuards(JwtAuthGuard, RolesGuard)
  @Roles(UserRolesEnum.Admin)
  @ApiOperation({ summary: 'Delete event' })
  async remove(@Param('id') id: string) {
    return this.eventsService.remove(id);
  }
}
```

ДОДАТОК А.12 – Валідація даних

typescript

```
// modules/events/dto/create-event.dto.ts
export class CreateEventDto {
  @IsString()
  @MinLength(3)
  @MaxLength(100)
  title: string;
  @IsString()
  @MinLength(10)
  @MaxLength(1000)
  description: string;
  @IsDateString()
  @IsNotEmpty()
  dateStart: Date;
  @IsDateString()
  @IsNotEmpty()
  dateEnd: Date;
  @IsString()
  @IsNotEmpty()
  location: string;
  @IsNumber()
  @Min(1)
  @Max(10000)
  capacity: number;
  @IsString()
  @IsNotEmpty()
  category: string;
}
```

ДОДАТОК А.13 – завантаження зображень за допомогою Multer

typescript

```
// modules/files/files.service.ts
@Injectable()
export class FilesService {
  async uploadFile(file: Express.Multer.File): Promise<string> {
    const fileName = `${uuid()}-${file.originalname}`;
    const filePath = path.join('uploads', 'events', fileName);

    await fs.promises.mkdir(path.dirname(filePath), { recursive: true });
    await fs.promises.writeFile(filePath, file.buffer);

    return `/uploads/events/${fileName}`;
  }
}
```

ДОДАТОК А.14 – Swagger документація

typescript

```
// main.ts
async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  const config = new DocumentBuilder()
    .setTitle('Events Track API')
    .setDescription('API for events management system')
    .setVersion('1.0')
    .addBearerAuth()
    .build();

  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api/docs', app, document);

  await app.listen(3001);
}
```

ДОДАТОК А.15 – Глобальний фільтр для обробки помилок

typescript

```
// common/filters/http-exception.filter.ts
@Catch(HttpException)
export class HttpExceptionFilter implements ExceptionFilter {
  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const request = ctx.getRequest<Request>();
    const status = exception.getStatus();
    const exceptionResponse = exception.getResponse();

    response.status(status).json({
      statusCode: status,
      timestamp: new Date().toISOString(),
      path: request.url,
      message: exceptionResponse['message'] || exception.message,
    });
  }
}
```

ДОДАТОК А.16 – Вбудована підтримка тестування з Jest у NestJS

typescript

```
// modules/events/events.service.spec.ts
describe('EventsService', () => {
  let service: EventsService;
  let repository: Repository<Event>;
  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [
        EventsService,
        {
          provide: getRepositoryToken(Event),
          useClass: Repository,
        },
      ],
    }).compile();
    service = module.get<EventsService>(EventsService);
    repository                                                    =
module.get<Repository<Event>>(getRepositoryToken(Event));
  });
  it('should be defined', () => {
    expect(service).toBeDefined();
  });
  describe('findAll', () => {
    it('should return an array of events', async () => {
      const result = [];
      jest.spyOn(repository, 'find').mockResolvedValue(result);

      expect(await service.findAll({})).toBe(result);
    });
  });
});
```

ДОДАТОК А.17 – Основні функції проєкту

typescript

```
// Backend: events.service.ts
async findAll(filters: EventsFilterDto) {
  const query = this.eventRepository.createQueryBuilder('event');
   if (filters.search) {
   query.andWhere(
    'LOWER(event.title) LIKE LOWER(:search) OR LOWER(event.description) LIKE LOWER(:search)',
    { search: `%${filters.search}%` }
   );
  }
   if (filters.category) {
   query.andWhere('event.category = :category', { category: filters.category });
  }
   if (filters.dateFrom) {
   query.andWhere('event.dateStart >= :dateFrom', { dateFrom: filters.dateFrom });
  }
   const [items, total] = await query
   .skip((filters.page - 1) * filters.limit)
   .take(filters.limit)
   .getManyAndCount();

  return {
   items,
   total,
   page: filters.page,
   totalPages: Math.ceil(total / filters.limit)
  };
 }
```

ДОДАТОК А.18 – Процес реєстрації на подію

typescript

```
// Backend: registrations.service.ts
async register(eventId: string, userId: string) {
  // Перевірка події
  const event = await this.eventRepository.findOne({
    where: { id: eventId },
    relations: ['registrations']
  });
    if (!event) {
    throw new NotFoundException('Event not found');
  }
    // Перевірка місць
  if (event.registrations.length >= event.capacity) {
    throw new BadRequestException('Event is full');
  }
    // Перевірка дублікату
  const existing = await this.registrationRepository.findOne({
    where: { event: { id: eventId }, user: { id: userId } }
  });
    if (existing) {
    throw new ConflictException('Already registered');
  }
    // Створення реєстрації
  const registration = this.registrationRepository.create({
    event,
    user: { id: userId },
    status: 'confirmed'
  });

  await this.registrationRepository.save(registration);
    // Відправка email
  await this.emailService.sendRegistrationConfirmation(userId, event);
    return registration;
}
```

ДОДАТОК А.19 – Реалізація захисту від CSRF на Node.js/Express з TypeScript

typescript

```
import csurf from 'csurf';
import express from 'express';

const app = express();

// Підключаємо middleware csurf
app.use(csurf());

// Приклад маршруту, який вимагає CSRF-токен
app.post('/submit', (req, res) => {
  // Якщо CSRF-токен коректний, запит буде оброблено
  res.send('Форма успішно надіслана');
});

// Обробка помилки CSRF
app.use((err: any, req: express.Request, res: express.Response, next: express.NextFunction) => {
  if (err.code === 'EBADCSRFTOKEN') {
    res.status(403).send('Неправильний CSRF-токен');
  } else {
    next(err);
  }
});
```

ДОДАТОК А.20 – Rate Limiting для захисту від брутфорс атак та DDoS

```
app.use(rateLimit({
 windowMs: 15 * 60 * 1000, // 15 хвилин
 max: 100, // максимум 100 запитів з одного IP
 message: 'Too many requests from this IP'
}));

// Окремий ліміт для логіну
const loginLimiter = rateLimit({
 windowMs: 15 * 60 * 1000,
 max: 5, // максимум 5 спроб входу
 skipSuccessfulRequests: true
});
```

ДОДАТОК А.21 – Захист персональних даних за допомогою системи GDPR

typescript

```
const storage = multer.diskStorage({
  destination: './uploads',
  filename: (req, file, cb) => {
    const uniqueName = `${uuid()}-${Date.now()}${path.extname(file.originalname)}`;
    cb(null, uniqueName);
  }
});
const upload = multer({
  storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB
  fileFilter: (req, file, cb) => {
    const allowedTypes = /jpeg|jpg|png|gif/;
    const extname = allowedTypes.test(path.extname(file.originalname).toLowerCase());
    const mimetype = allowedTypes.test(file.mimetype);

    if (mimetype && extname) {
      return cb(null, true);
    } else {
      cb(new Error('Invalid file type'));
    }
  }
});
```

ДОДАТОК А.22 – Приклад unit тесту для React компонента

typescript

```
import { render, screen, fireEvent } from '@testing-library/react';
import { EventCard } from './EventCard';
describe('EventCard', () => {
  const mockEvent = {
    id: '1',
    title: 'Test Event',
    description: 'Test Description',
    date: new Date('2024-12-01'),
    location: 'Test Location',
    capacity: 100,
    registrations: 50
  };
    it('should render event information', () => {
    render(<EventCard event={mockEvent} />);

    expect(screen.getByText('Test Event')).toBeInTheDocument();
    expect(screen.getByText('Test Description')).toBeInTheDocument();
    expect(screen.getByText('Test Location')).toBeInTheDocument();
  });
    it('should handle register button click', () => {
    const onRegister = jest.fn();
    render(<EventCard event={mockEvent} onRegister={onRegister} />);

    fireEvent.click(screen.getByText('Register'));
    expect(onRegister).toHaveBeenCalledWith('1');
  });
});
```

ДОДАТОК А.23 – Приклад інтеграційного тесту для NestJS

typescript

```
describe('EventsController (e2e)', () => {
  let app: INestApplication;
    beforeEach(async () => {
    const moduleFixture: TestingModule = await Test.createTestingModule({
      imports: [AppModule],
    }).compile();
        app = moduleFixture.createNestApplication();
    await app.init();
  });
    it('/events (GET)', () => {
    return request(app.getHttpServer())
      .get('/events')
      .expect(200)
      .expect((res) => {
        expect(res.body).toHaveProperty('items');
        expect(res.body).toHaveProperty('total');
        expect(Array.isArray(res.body.items)).toBe(true);
      });
  });
    it('/events (POST) - unauthorized', () => {
    return request(app.getHttpServer())
      .post('/events')
      .send({
        title: 'New Event',
        description: 'Description'
      })
      .expect(401);
  });
});
```

ДОДАТОК А.24 – Приклад E2E тесту

typescript

```
describe('Event Registration Flow', () => {
  beforeEach(() => {
    cy.visit('/');
    cy.login('user@example.com', 'password');
  });

  it('should allow user to register for event', () => {
    // Перейти на сторінку подій
    cy.get('[data-testid="nav-events"]').click();

    // Знайти та відкрити подію
    cy.get('[data-testid="event-card"]').first().click();

    // Зареєструватися
    cy.get('[data-testid="register-button"]').click();

    // Перевірити підтвердження
    cy.get('[data-testid="success-message"]')
      .should('contain', 'Successfully registered');

    // Перевірити зміну кнопки
    cy.get('[data-testid="register-button"]')
      .should('contain', 'Registered')
      .should('be.disabled');
  });
});
```

ДОДАТОК А.25 – Тестування продуктивності

xml

```
<ThreadGroup>
  <stringProp name="ThreadGroup.num_threads">100</stringProp>
  <stringProp name="ThreadGroup.ramp_time">60</stringProp>
  <stringProp name="ThreadGroup.duration">300</stringProp>

  <HTTPSamplerProxy>
    <stringProp name="HTTPSampler.domain">localhost</stringProp>
    <stringProp name="HTTPSampler.port">3001</stringProp>
    <stringProp name="HTTPSampler.path">/api/events</stringProp>
    <stringProp name="HTTPSampler.method">GET</stringProp>
  </HTTPSamplerProxy>
</ThreadGroup>
```

ДОДАТОК А.26 – Модульне тестування (етап 2)

typescript

```
// Результати покриття коду:
// -------------------------|---------|----------|---------|---------|
// File                     | % Stmts | % Branch | % Funcs | % Lines |
// -------------------------|---------|----------|---------|---------|
// All files                |   85.3  |   78.9   |  88.2   |  84.7   |
//  components              |   92.1  |   85.3   |  94.5   |  91.8   |
//  services                |   88.7  |   82.4   |  90.1   |  88.2   |
//  utils                   |   95.8  |   91.2   |  97.3   |  95.4   |
//  hooks                   |   81.4  |   75.6   |  83.9   |  80.9   |
// -------------------------|---------|----------|---------|---------|
```

ДОДАТОК А.27 – Успішне тестування

typescript

```
app.use(rateLimit({
 windowMs: 15 * 60 * 1000, // 15 хвилин
 max: 100, // максимум 100 запитів з одного IP
 message: 'Too many requests from this IP'
}));

// Окремий ліміт для логіну
const loginLimiter = rateLimit({
 windowMs: 15 * 60 * 1000,
 max: 5, // максимум 5 спроб входу
 skipSuccessfulRequests: true
});
```

ДОДАТОК А.28 – Контроль доступу

typescript

```
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles(UserRolesEnum.Admin)
@Delete(':id')
async deleteEvent(@Param('id') id: string) {
  return this.eventsService.remove(id);
}
```

ДОДАТОК А.29 – Логи для можливості аудиту

typescript

```
@Injectable()
export class AuditService {
  async log(action: string, userId: string, details: any) {
    await this.auditRepository.save({
      action,
      userId,
      details,
      ipAddress: this.getClientIp(),
      userAgent: this.getUserAgent(),
      timestamp: new Date()
    });
  }
}
```

ДОДАТОК А.30 – Надійний рівень захисту персональної інформації

typescript

```
const storage = multer.diskStorage({
  destination: './uploads',
  filename: (req, file, cb) => {
    const uniqueName = `${uuid()}-${Date.now()}${path.extname(file.originalname)}`;
    cb(null, uniqueName);
  }
});

const upload = multer({
  storage,
  limits: { fileSize: 5 * 1024 * 1024 }, // 5MB
  fileFilter: (req, file, cb) => {
    const allowedTypes = /jpeg|jpg|png|gif/;
    const extname = allowedTypes.test(path.extname(file.originalname).toLowerCase());
    const mimetype = allowedTypes.test(file.mimetype);

    if (mimetype && extname) {
      return cb(null, true);
    } else {
      cb(new Error('Invalid file type'));
    }
  }
});
```