БАКАЛАВРСЬКА

РОБОТА

БР.ІП – 18.00.00.000 ІПЗ

Група ІП-23-1К

Гринкевич Іван

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Гринкевич Іван Ярославович

(прізвище, ім'я, по батькові)

УДК 004.738.5
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розробка ресурсу модерації та планування розважальних івентів
(назва роботи)

Інженерія програмного забезпечення
(назва освітньої програми)

121– Інженерія програмного забезпечення
(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Гринкевич І.Я.
(підпис, ініціали та прізвище здобувача)

Науковий керівник Романишин Тарас Любомирович, к.т.н., доцент
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри

доц. Бандура В.В.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою ІПЗ
доц. В.В. Бандура
“ ” 2025 р.

ЗАВДАННЯ

НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Гринкевичу Івану Ярославовичу
(прізвище, ім’я, по-батькові)

1. Тема проєкту (роботи) “Розробка ресурсу модерації та планування розважальних івентів”

керівник проєкту (роботи) Романишин Т.Л., к.т.н., доцент

затверджені наказом закладу вищої освіти від **“** 28 ” квітня 2025 р. № 264/7

2. *Строк подання студентом проєкту (роботи)* 10 червня 2025 р.

3. *Вихідні дані до проєкту (роботи)* Результати і матеріали отримані під час проходження переддипломної практики

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

1. Огляд існуючих рішень та теоретичних основ проєктування інструментів аналізу безпеки програмного коду
2. Аналіз вимог та постановка задачі
3. Проєктування системи
4. Реалізація проєкту
5. Тестування та впровадження

5. *Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)*

1. Use-case діаграма користувачів (рис. 2.1, ст. 25)
2. Діаграма компонентів (рис. 3.1, ст. 32)
3. Діаграма розгортання (рис.3.2, ст. 33)
4. Діаграма класів (рис.3.3, ст. 35)
5. Діаграма послідовностей (рис.3.5, ст. 39)

1. ***Консультанти розділів проєкту (роботи)***

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***

2. ***Дата видачі завдання*** ***28 лютого***

Керівник _____ _____
(підпис) (розшифровка підпису)

Завдання прийняв до виконання _____ _____
(підпис) (розшифровка підпису)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	***Назва етапів дипломного проєкту (роботи)***	***Строк виконання етапів проєкту***	***Примітка***
1	Видача завдання	28.02.2025	Виконав
2	Збір матеріалу, підготовка теоретичної частини	12.03.2025	Виконав
3	Розробка програмної частини	24.05.2025	Виконав
4	Оформлення пояснювальної записки і графічної частини	28.05.2025	Виконав
5	Відгук керівника дипломного проєкту	30.05.2025	Виконав
6	Рецензування і нормоконтроль	03.06.2025	Виконав
7	Здачі на перевірку готового дипломного проєкту завідувачі відділенням для допуску до захисту	07.06.2025	Виконав

Студент – дипломник _____ _____
(підпис) (розшифровка підпису)

Керівник роботи _____ _____
(підпис) (розшифровка підпису)

АНОТАЦІЯ

Бакалаврська робота містить 73 сторінку, 22 рисунки, 2 таблиці, список використаних джерел із 34 найменувань.

Метою роботи: розробка веб-застосунку для управління подіями, який забезпечує ефективну взаємодію між користувачами, організаторами та адміністраторами завдяки використанню сучасних технологій, зручного інтерфейсу та широкого функціоналу, включаючи систему реєстрації, створення подій, комунікацію в реальному часі та аналітику.

Об'єктом дослідження: процеси розробки та функціонування веб-застосунку для управління подіями, спрямованого на покращення організації, адміністрування подій та комунікації між учасниками.

Предметом дослідження: методи, технології та інструменти створення веб-застосунку для управління подіями, зокрема аналіз вимог користувачів, проєктування архітектури системи, розробка інтерфейсу користувача, структури баз даних, реалізація функціоналу месенджера, управління доступами, тестування, а також оцінка ефективності впровадженої системи.

Результати дослідження: Створено веб-додаток з використанням Angular та Nest.js, який полегшує пошук роботи.

В першому розділі аналізуються тренди та технології, які набувають популярності та обґрунтовується їх доцільність для подальшого застосування в проєкті. Розглядаються сучасні підходи до веб-розробки та комунікації, що забезпечують високу продуктивність і зручність користування.

В другому розділі формулюється завдання проєктування: яким є кінцевий результат, які межі системи, які припущення та обмеження діють. Визначаються ключові функції системи та роль різних типів користувачів.

В третьому розділі описано проєктування системи. Представлено архітектурні рішення, структуру бази даних та основні компоненти застосунку.

В четвертому розділі розглянуто всі ключові елементи практичної реалізації проєкту. Наведено приклади коду, реалізацію інтерфейсу та логіки взаємодії користувачів.

У п’ятому розділі розглянуто процес впровадження системи у експлуатацію. Описані етапи тестування, налаштування та підтримки веб-застосунку в робочому середовищі.

Висновок: Розробка веб-додатку успішно реалізована, забезпечуючи відповідність сучасним вимогам ринку праці та високій якості програмного продукту.

КЛЮЧОВІ СЛОВА: УПРАВЛІННЯ ПОДІЯМИ, ВЕБ-ЗАСТОСУНОК, ANGULAR, NEST.JS, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, МОДЕЛЮВАННЯ, ТЕСТУВАННЯ, КОМУНІКАЦІЯ, КОНТРОЛЬ ВЕРСІЙ, РЕЄСТРАЦІЯ КОРИСТУВАЧІВ, МЕСЕНДЖЕР. ЦЕ ВЕЛИКИМИ ЛІТЕРАМИ НАПИС

ANNOTATION

The bachelor's thesis contains 73 pages, 22 illustrations, 2 tables, and a list of 34 references.

The purpose of the work is the development of a web application for event management that ensures effective interaction between users, organizers, and administrators through the use of modern technologies, a user-friendly interface, and extensive functionality, including registration systems, event creation, real-time communication, and analytics.

The object of the study is the processes of development and operation of a web application for event management aimed at improving event organization, administration, and communication among participants.

The subject of the study includes methods, technologies, and tools for creating a web application for event management, specifically the analysis of user requirements, system architecture design, user interface development, database structure, implementation of messenger functionality, access control, testing, and evaluation of the effectiveness of the implemented system.

The research results include the creation of a web application using Angular and Nest.js that facilitates event management.

The first chapter analyzes trends and technologies that are gaining popularity and justifies their applicability for further use in the project. Modern approaches to web development and communication, which ensure high performance and user convenience, are also considered.

The second chapter formulates the design task: the final result, system boundaries, assumptions, and constraints. Key system functions and roles of different user types are defined.

The third chapter describes the system design. Architectural decisions, database structure, and main components of the application are presented.

The fourth chapter covers all key elements of the practical implementation of the project. Code examples, interface implementation, and user interaction logic are provided.

The fifth chapter discusses the process of deploying the system into operation. The stages of testing, configuration, and maintenance of the web application in the working environment are described.

Conclusion: The development of the web application has been successfully carried out, ensuring compliance with current labor market requirements and high-quality software standards.

KEYWORDS: EVENT MANAGEMENT, WEB APPLICATION, ANGULAR, NEST.JS, USER INTERFACE, MODELING, TESTING, COMMUNICATION, VERSION CONTROL, USER REGISTRATION, MESSENGER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СУБД – Система управління базами даних

UML – Unified Modelling Language

ООП – Об'єктно-орієнтоване програмування

MS SQL – Microsoft Structured Query Language

JWT – JSON Web Token

CRUD-операції – Create, Read, Update, Delete операції

SPA – Single Page Application

HTTPS – HyperText Transfer Protocol Secure

UI – User Interface

UX – User Experience

RxJS – Reactive Extensions for JavaScript

CLI – Command Line Interface

IDE – Integrated Development Environment

ВСТУП

У сучасних умовах цифрової трансформації суспільства стрімко зростає потреба в інноваційних рішеннях для організації та управління подіями різного масштабу – від невеликих зустрічей до масштабних конференцій. Швидкий ритм життя, розповсюдження віддаленої роботи, соціальні та економічні виклики, включаючи пандемію та військові дії, змушують організаторів подій шукати нові інструменти для ефективної комунікації, планування й адміністрування заходів. Це підкреслює необхідність створення зручних, гнучких та функціональних платформ, здатних підтримувати всі етапи взаємодії між організаторами, учасниками та адміністраторами подій.

Актуальність теми полягає в потребі створення сучасного застосунку для управління подіями, який би об'єднував функціонал реєстрації, планування, адміністрування та комунікації, водночас забезпечуючи інтуїтивний інтерфейс, налаштування зовнішнього вигляду та підтримку багатомовності. Попри наявність ряду існуючих рішень (наприклад, Eventbrite, Meetup, Airmeet), більшість із них мають обмежений функціонал для локалізованих потреб, не враховують вимоги безпеки, гнучкість доступів чи зручність організації групових чатів. Це створює передумови для розробки нової платформи, яка відповідала б сучасним викликам і пропонувала комплексний підхід до управління подіями.

Метою цієї роботи є розробка програмного забезпечення менеджменту подій, що забезпечуватиме зручне створення, управління та адміністрування подій, а також взаємодію між користувачами через інтегрований месенджер. Особлива увага приділяється реалізації ролей з різними рівнями доступу, інструментам комунікації, оплаті подій та аналітиці.

Основними завданнями дослідження є: аналіз існуючих рішень у сфері управління подіями; виявлення потреб потенційних користувачів платформи; розробка архітектури застосунку; впровадження основного функціоналу з використанням Angular і Nest.js; реалізація засобів комунікації (чати); забезпечення авторизації та ролей доступу; тестування та оптимізація системи; підготовка

рекомендацій щодо подальшого розвитку.

Об’єктом дослідження виступає процес проєктування та реалізації програмного забезпечення для управління подіями, що охоплює повний життєвий цикл розробки – від аналізу вимог до впровадження.

Предметом дослідження є технології, методи та інструменти, що використовуються для створення вебзастосунків, орієнтованих на багаторівневу взаємодію користувачів, зокрема засоби розмежування доступу, обробки даних, інтерактивного інтерфейсу, обміну повідомленнями, аналітики та масштабування. Також розглядаються аспекти моделювання бізнес-процесів організації подій, взаємодії між учасниками, безпеки та UX-дизайну.

Методи дослідження: аналіз і порівняння функціоналу аналогічних систем, проектування бази даних, створення прототипів, реалізація клієнтської та серверної частин, модульне й інтеграційне тестування, верифікація функцій месенджера, аналіз навантаження та оцінка продуктивності.

Наукова новизна роботи полягає в розробці комплексної веб-платформи для управління подіями, яка інтегрує функціонал створення подій, адміністрування, реєстрації учасників, системи ролей доступу та вбудованого месенджера у межах єдиного інтерфейсу. Вперше реалізовано рішення, що орієнтоване не лише на базову організацію подій, а й на забезпечення гнучкої взаємодії між учасниками в реальному часі з урахуванням сучасних вимог до безпеки, масштабованості, мультимовності та персоналізації інтерфейсу.

Результатом роботи стане платформа менеджменту подій, що дозволятиме створювати та адмініструвати події з детальними описами, реєструвати користувачів із різними ролями, забезпечувати зручний інтерфейс та функціонал для спілкування між учасниками. Передбачається, що ця система спростить організацію подій навіть у складних умовах, забезпечуючи ефективну комунікацію та прозоре адміністрування, що особливо важливо в сучасному нестабільному середовищі.

Бакалаврська робота містить 73 сторінки, 22 рисунки, 2 таблиці і 34 бібліографічних джерел.

РОЗДІЛ 1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕОРЕТИЧНИХ ОСНОВ ПРОЄКТУВАННЯ ІНСТРУМЕНТІВ АНАЛІЗУ БЕЗПЕКИ ПРОГРАМНОГО КОДУ

1.1 Основні поняття та визначення

У межах розробки інформаційної системи для управління подіями доцільно розглянути базові поняття, які є ключовими для розуміння предметної області, функціональності платформи та технологій, що використовуються в процесі її реалізації.

Подія – це заздалегідь запланована активність, яка має конкретну дату, час, місце проведення (фізичне або віртуальне) та учасників. Події можуть бути публічними або приватними, платними чи безкоштовними, і мати різний ступінь взаємодії між організатором та учасниками. У контексті електронних сервісів подія є одиницею даних, яка має власні мета-дані та належить певному користувачу.

Система управління подіями – це програмне забезпечення, що забезпечує повний життєвий цикл події: її створення, редагування, публікацію, реєстрацію учасників, обробку оплат (за необхідності), комунікацію між сторонами, аналітику та адміністрування. Основна мета таких систем – спростити організаційні процеси та покращити взаємодію між усіма учасниками подій.

Користувач системи – особа, яка взаємодіє з платформою через інтерфейс, відповідно до своєї ролі. У запропонованій системі передбачено кілька ролей:

– неавторизований користувач – має доступ до перегляду подій, створення облікового запису, базових налаштувань інтерфейсу;

– авторизований користувач – має змогу керувати власним обліковим записом, створювати і редагувати події, записуватись на події, вести переписку в чатах;

– адміністратор – має розширені права керування всіма подіями, обліковими записами інших користувачів, доступ до статистики та аналітики.

Месенджер – складова частина системи, що дозволяє користувачам

обмінюватися повідомленнями в режимі реального часу. В контексті проєкту реалізується підтримка як особистих, так і групових чатів з можливістю редагування та видалення повідомлень, відповіді на конкретні повідомлення, додавання/видалення учасників, пошуку за контактами.

Авторизація та аутентифікація – процеси перевірки особи користувача, який входить у систему. Аутентифікація підтверджує, що користувач є тим, за кого себе видає (наприклад, шляхом введення логіна та пароля), тоді як авторизація визначає рівень його доступу до функціоналу системи.

Front-end – це клієнтська частина вебзастосунку, яка відповідає за візуальне представлення даних, інтерактивність і взаємодію з користувачем. У проєкті використовується Angular [4] – сучасний фреймворк для створення односторінкових застосунків з багатим функціоналом.

Back-end – серверна частина застосунку, що забезпечує обробку логіки, управління базами даних, обробку запитів від клієнта, реалізацію авторизації та взаємодію між компонентами. Реалізується на базі Nest.js [5] – прогресивного фреймворку на Node.js [12] для побудови масштабованих серверних додатків.

База даних – структурований набір інформації, що зберігає дані про користувачів, події, повідомлення, ролі та інші об’єкти системи. У межах проєкту передбачається використання реляційної бази даних (наприклад, MS SQL [7]), яка забезпечує ефективне зберігання та доступ до даних.

Інтерфейс користувача (UI) – це сукупність елементів, за допомогою яких користувач взаємодіє із системою: кнопки, форми, меню, діалогові вікна тощо. Важливими аспектами є простота, інтуїтивність та адаптивність дизайну.

UX (User Experience) – досвід користувача, що включає зручність, логіку та задоволення від взаємодії з платформою. У розробці «EventManager» UX-дизайн відіграє важливу роль для залучення та утримання користувачів.

API (Application Programming Interface) – інтерфейс програмування додатків, який дозволяє клієнтській частині та стороннім сервісам взаємодіяти із серверною логікою через стандартизовані запити.

Роль користувача – набір дозволених дій у системі, що визначає рівень

доступу до функцій. В проєкті розрізняють ролі «гостя», «користувача» та «адміністратора».

Сеанс (Session) – період активної взаємодії користувача з системою, що починається після успішної авторизації та завершується виходом або таймаутом.

Вебсокети (WebSockets) – технологія, що забезпечує двонаправлену комунікацію в реальному часі між клієнтом і сервером, що необхідно для роботи месенджера.

Локалізація (i18n) – процес адаптації інтерфейсу платформи під різні мови та регіональні налаштування для покращення доступності.

Теми оформлення (Themes) – варіанти візуального дизайну інтерфейсу, які користувач може обирати для персоналізації зовнішнього вигляду застосунку.

Таким чином, наведені поняття формують термінологічну базу, необхідну для подальшого розгляду методів реалізації, аналізу існуючих підходів і проєктування системи управління подіями з розширеними можливостями.

1.2 Аналіз сучасного стану питання

Сучасна цифрова епоха сформувала нові підходи до організації подій, де важливу роль відіграють онлайн-платформи для управління заходами різного рівня – від освітніх і корпоративних до культурно-розважальних та соціальних. Ці сервіси дозволяють автоматизувати процеси планування, реєстрації, комунікації, моніторингу активності учасників, що значно знижує навантаження на організаторів. Розвиток даної галузі зумовлений як технічним прогресом, так і потребою у швидкій та зручній взаємодії між організаторами й аудиторією.

На ринку вже присутні відомі рішення, такі як Eventbrite, Meetup, Aventri, Hopin, Whova та інші. Вони пропонують базовий функціонал, включаючи створення подій, обробку реєстрацій, оплату, аналітику, іноді – інтеграцію з соціальними мережами або засобами відеоконференцій. Однак більшість із них орієнтовані на англомовну аудиторію та мають обмежені можливості для гнучкого налаштування інтерфейсу, локалізації та індивідуального розширення

функціональності.

Ще одним важливим аспектом є наявність якісної системи комунікації між учасниками. У багатьох платформах цей функціонал або відсутній, або реалізований у вигляді простих коментарів без підтримки повноцінного месенджера. Це обмежує гнучкість взаємодії між учасниками та ускладнює обговорення деталей заходів, особливо в групових або масштабних подіях.

З технічної точки зору, більшість сучасних платформ реалізовані як вебзастосунки, що забезпечує доступ із будь-якого пристрою з інтернет-з'єднанням. Часто використовуються фреймворки Angular, React або Vue на фронтенді та Node.js, Django або Laravel на бекенді. З точки зору архітектури, актуальними є мікросервісні рішення, хмарне зберігання даних, RESTful API або GraphQL API, що забезпечує масштабованість і гнучкість.

Щодо рівня доступу, рольова модель зазвичай поділяє користувачів на організаторів, учасників та адміністраторів. Але в деяких системах відсутній чіткий поділ прав, що створює ризики в адмініструванні та безпеці.

Особливу актуальність у сучасних умовах набуває адаптивність і безперервність бізнес-процесів, зокрема під час кризових ситуацій – війни, пандемії, техногенних катастроф. Саме тому ефективні системи управління подіями мають враховувати стійкість до навантажень, підтримку кількох мов, зручну навігацію та швидку обробку запитів.

Підсумовуючи, можна зазначити, що хоча існує чимало платформ для організації подій, жодна з них не охоплює всі потреби одночасно: рольову модель, гнучку локалізацію, інтегрований месенджер, аналітику та налаштування інтерфейсу. Це створює передумови для розробки нового, більш функціонального рішення, яке відповідатиме сучасним викликам та очікуванням користувачів.

1.3 Класифікація підходів і методів

У процесі розробки сучасних інформаційних систем, зокрема систем управління подіями, використовується широкий спектр підходів і методів, які

охоплюють усі етапи життєвого циклу програмного забезпечення: від аналізу вимог до тестування та впровадження. Для побудови ефективної, гнучкої та масштабованої системи необхідно обґрунтовано обрати підходи як до архітектурного проєктування, так і до реалізації окремих функціональних компонентів.

1.3.1. Методологічні підходи до розробки ПЗ.

Найпоширенішими є каскадна (Waterfall) модель, ітераційна модель, гнучкі методології (Agile, Scrum) та DevOps-підхід. У контексті системи «EventManager» доцільно застосувати Agile-підхід, оскільки він дозволяє розробляти застосунок поетапно, адаптуючись до змінних вимог користувачів, із частим тестуванням і зворотним зв’язком. Це особливо важливо при створенні комплексних систем із багатьма ролями та сценаріями взаємодії.

1.3.2. Підходи до архітектури системи.

Залежно від функціональності та масштабованості платформи, можливо використання монолітної або мікросервісної архітектури. Для «EventManager» доцільно обрати модульну монолітну архітектуру, яка реалізується у Nest.js за рахунок структурованого розділення компонентів (модулі, контролери, сервіси), що полегшує супровід і розширення функцій. У майбутньому можливий перехід до мікросервісної моделі для підвищення масштабованості.

1.3.3. Методи реалізації клієнтської частини.

Для побудови інтерактивного, динамічного і чутливого до дій користувача інтерфейсу застосовується Angular – фреймворк, що підтримує модульність, двостороннє зв’язування даних, використання шаблонів та реактивних форм. Angular дозволяє реалізувати складну рольову модель, підтримку тем, мов інтерфейсу та гнучке керування станом застосунку.

1.3.4. Методи управління доступом та безпекою.

Система має підтримувати механізми аутентифікації та авторизації. Використовуються підходи на основі JWT [21] (JSON Web Tokens) для збереження сесії та перевірки прав доступу.

1.3.5. Методи забезпечення комунікації в системі.

Для реалізації функціоналу месенджера застосовується WebSocket або Socket.IO, що дозволяє забезпечити двосторонній зв’язок між клієнтом і сервером у режимі реального часу. Також можуть використовуватись REST-методи для історії повідомлень і фонових операцій.

1.3.6. Методи зберігання та обробки даних.

Для збереження інформації про події, користувачів, повідомлення, чати та реєстрації використовуються реляційні бази даних, зокрема Microsoft SQL. Методи нормалізації, індексації та оптимізації запитів забезпечують високу продуктивність та консистентність даних.

Таким чином, ефективне поєднання згаданих підходів і методів дозволяє створити систему управління подіями, що буде не лише функціональною, а й надійною, безпечною та зручною для кінцевих користувачів із різними ролями та потребами.

1.4 Огляд існуючих рішень і власних доповнень

На сучасному ринку інформаційних технологій існує чимало платформ для організації подій, кожна з яких має власний набір функцій, переваг та обмежень. Найбільш відомі серед них – Eventbrite, Meetup, Whova, Hopin, а також спеціалізовані рішення для корпоративного використання або віртуальних конференцій.

Eventbrite є однією з найпоширеніших платформ, яка дозволяє створювати події, продавати квитки, отримувати базову статистику та повідомлення електронною поштою. Проте, система має обмежену підтримку кастомізації інтерфейсу, відсутність інтегрованого месенджера та фокус на англомовну аудиторію. Вона більше підходить для відкритих публічних заходів, ніж для внутрішніх або закритих подій.

Meetup орієнтований на спільноти за інтересами. Хоча платформа забезпечує можливість створення подій і комунікацію учасників, її функціональність обмежена переважно неформальними подіями, а інтерфейс має

жорстку структуру, що не дозволяє реалізувати гнучке налаштування системи доступу або розширену аналітику.

Whova та Hopin пропонують рішення для великих конференцій, підтримують відеотрансляції, віртуальні кімнати, а також інтеграцію з календарями та CRM. Їхній функціонал досить потужний, однак ці сервіси орієнтовані на великі корпоративні заходи, мають високу вартість та складну систему адміністрування, що робить їх менш придатними для середніх або невеликих організаторів.

На цьому фоні стає очевидною потреба у створенні системи, яка б поєднувала найкращі практики зручності, гнучкості та сучасних технічних рішень – саме такою є концепція «EventManager». Запропонований застосунок має ряд власних доповнень і вдосконалень, які вирізняють його серед аналогів:

– гнучка рольова модель з чітким розмежуванням доступу між неавторизованими користувачами, зареєстрованими учасниками та адміністраторами;

– інтегрований чат-месенджер з підтримкою особистих та групових чатів, редагуванням і видаленням повідомлень, відповідями на окремі репліки – функціонал, якого не вистачає в більшості сучасних платформ;

– підтримка платних і безкоштовних подій, з можливістю реєстрації, онлайн-оплати та управління квитками;

– налаштування інтерфейсу, включаючи вибір теми оформлення та мови, що робить систему доступною для ширшої аудиторії;

– можливість створення, редагування та аналізу власних подій, із доступом до аналітичних даних для користувачів і адміністраторів;

– завдяки використанню сучасного технологічного стеку (Angular + Nest.js), система дозволяє ефективно реалізувати як фронтенд, так і бекенд частину з високою продуктивністю та зручністю для розробки і подальшого масштабування.

Таким чином, запропонована система «EventManager» не тільки враховує кращі практики існуючих рішень, але й розширює їх можливості за рахунок більш

глибокої персоналізації, безпечної комунікації та інтеграції ключових функцій в єдину платформу.

1.5 Висновки по розділу

У першому розділі бакалаврської роботи було здійснено аналіз теоретичних засад, що стосуються створення систем управління подіями. Розглянуто ключові поняття та терміни, пов’язані з організацією подій, адмініструванням користувачів, комунікацією всередині цифрових платформ, а також роллю сучасних вебтехнологій у розробці подібних систем.

Проаналізовано сучасний стан у сфері організації подій, виявлено потребу в більш інтегрованих, персоналізованих та безпечних рішеннях, здатних забезпечити не лише ефективне управління подіями, але й комунікацію між користувачами різного рівня доступу. Існуючі платформи мають обмеження в гнучкості, локалізації, доступності та розширенні функціоналу.

Була проведена класифікація підходів і методів, що використовуються при розробці подібних систем, зокрема в контексті веброзробки, побудови ролей, безпеки та реалізації внутрішніх комунікаційних модулів.

У рамках огляду існуючих рішень розглянуто найпоширеніші платформи, їхні переваги та недоліки, що дало змогу чітко окреслити функціональні прогалини, які може заповнити запропонований застосунок. Основні інноваційні доповнення проєкту – вбудований месенджер, розширені налаштування користувачів, локалізація інтерфейсу – надають переваги, які відсутні в аналогах.

Отже, проведений аналіз підтверджує актуальність та доцільність розробки системи «EventManager», яка відповідає сучасним вимогам до цифрових платформ, спрямованих на організацію подій та інтерактивну взаємодію між користувачами. У наступному розділі буде детально розглянуто вимоги до системи, її архітектурні рішення та особливості реалізації.

РОЗДІЛ 2. АНАЛІЗ ВИМОГ ТА ПОСТАНОВКА ЗАДАЧІ

2.1 Збір та аналіз вимог користувачів

Процес розробки будь-якої складної інформаційної системи починається з ретельного збору та аналізу вимог користувачів. Цей етап є критично важливим, оскільки саме від глибини розуміння потреб цільової аудиторії залежить ефективність майбутньої системи. У випадку з платформою «EventManager», яка передбачає багатофункціональне управління подіями, взаємодію користувачів у реальному часі та адміністративне адміністрування, потреби користувачів є різноманітними, залежно від їхніх ролей у системі.

2.1.1 Ідентифікація основних категорій користувачів

Першочергово, для збору вимог було визначено три основні категорії користувачів:

1. Неавторизовані користувачі – відвідувачі, які ще не створили обліковий запис.
2. Авторизовані користувачі – зареєстровані учасники платформи, які можуть брати участь у подіях, створювати власні події, користуватись месенджером тощо.
3. Адміністратори – користувачі з розширеними правами доступу, які виконують контроль і модерацію всієї системи.

Кожна з категорій має специфічні потреби, тому збір вимог був диференційованим.

2.1.2 Методи збору вимог

Для отримання актуальних, обґрунтованих і репрезентативних вимог було використано кілька методів:

- аналіз аналогічних рішень (Worksup, Eventbrite, Google Calendar, Meetup, Zoom Events, Microsoft Teams Events, Eventual тощо), що дозволило виявити поширені функції, сильні та слабкі сторони конкурентів;
- вивчення типових сценаріїв використання (Use Cases) – визначення

реальних ситуацій, у яких користувач взаємодіє з системою, наприклад: «Створити подію», «Записатися на подію», «Написати повідомлення в груповому чаті», «Переглянути аналітику».

2.1.3 Виявлені потреби неавторизованих користувачів

Аналіз показав, що більшість нових користувачів цінують:

- прозору навігацію без обов'язкової реєстрації;
- можливість перегляду подій без створення облікового запису;
- гнучкість налаштувань зовнішнього вигляду та мови інтерфейсу;
- простий і швидкий процес реєстрації.

Це вимагає реалізації адаптивного інтерфейсу з опціями перемикання теми (світла/темна), підтримки багатомовності, а також забезпечення максимальної інформативності навіть без входу в систему.

2.1.4 Вимоги авторизованих користувачів

Ця група виявила найбільшу кількість функціональних очікувань:

- зручне створення подій із можливістю додавання докладної інформації (дата, час, місце, кількість учасників, опис та інше);
- управління подіями: редагування, видалення події;
- реєстрація на події;
- інтеграція зі сторонніми календарями (Google Calendar, Outlook);
- система нагадувань (push-нотифікації, email);
- внутрішній месенджер з підтримкою групових чатів;
- редагування профілю, зміна пароля.

Окрему увагу було звернуто на безпеку даних та стабільність сервісу, оскільки користувачі очікують, що їхня персональна інформація та історія участі в подіях будуть надійно захищені.

2.1.5 Потреби адміністраторів

Адміністративна частина системи потребує розширеного функціоналу:

- перегляд усіх користувачів;
- редагування або видалення будь-якого вмісту;
- перегляд та аналіз активності користувачів (аналітика відвідуваності,

популярності подій, динаміка створення/реєстрацій).

У результаті аналізу було сформульовано чітку ієрархію вимог користувачів. Вони були класифіковані за ступенем важливості:

- критичні – без яких система не може функціонувати (реєстрація, створення подій, базова авторизація);
- бажані – покращують зручність і конкурентоспроможність (месенджер, підтримка тем, платні події);
- додаткові – збільшують цінність для окремих категорій користувачів (аналітика, інтеграція з іншими сервісами, темна тема).

Зібрані дані стали основою для формалізації функціональних і нефункціональних вимог (див. підрозділ 2.2), а також для розробки архітектури системи та її подальшого проектування.

2.2 Формулювання функціональних та нефункціональних вимог

Після проведення збору та аналізу потреб потенційних користувачів системи «EventManager», було сформульовано набір вимог, які визначають функціональну й нефункціональну частину майбутнього застосунку. Чітке формулювання вимог є основою для побудови ефективної, надійної й масштабованої інформаційної системи, яка відповідає очікуванням користувачів та технічним можливостям реалізації.

Функціональні вимоги описують поведінку системи, її основні можливості та реакції на дії користувача. Вони визначають, які функції має виконувати система у відповідь на зовнішні події.

Основні функціональні вимоги проєкту:

- відображення запланованих подій з детальною інформацією;
- реалізація авторизації користувачів із різними рівнями доступу (неавторизований користувач, авторизований користувач, адміністратор);
- неавторизований користувач має змогу:
- створити обліковий запис;

- переглядати події;
- налаштовувати застосунок (змінювати тему, вибирати мову інтерфейсу).

- авторизований користувач має змогу:
  - виконувати всі дії неавторизованого користувача;
  - вийти з облікового запису;
  - редагувати свій обліковий запис;
  - записуватись на події (з можливістю оплати платних подій);
  - створювати події;
  - редагувати власні події.
- функціонал месенджера:
  - пошук користувачів для створення чату;
  - створення, редагування та видалення групових чатів;
  - додавання та видалення учасників у груповому чаті;
  - відправлення, редагування та видалення повідомлень;
  - відповідь на певне повідомлення.
- адміністратор має змогу:
  - виконувати всі дії авторизованого користувача;
  - аналізувати статистику подій та користувачів;
  - редагувати або видаляти всі події;
  - редагувати або видаляти дані інших користувачів.

Відповідно до функціональних вимог створено Use-case діаграму, зображену на рисунку 2.1.

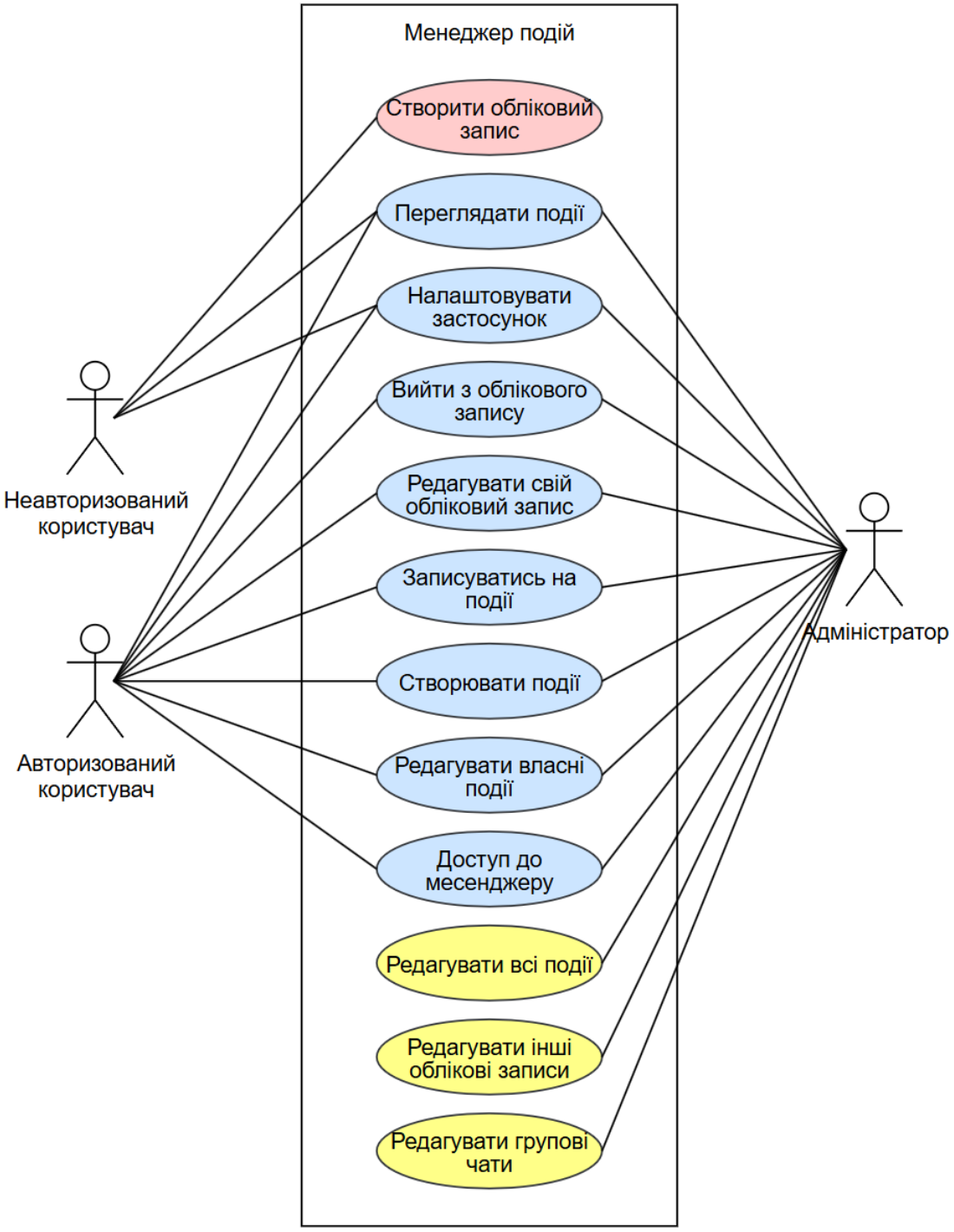

Рисунок 2.1 – Use-case діаграма користувачів

Нефункціональні вимоги описують якісні характеристики системи, які впливають на її експлуатаційні властивості: надійність, продуктивність, безпеку, зручність використання, підтримувані платформи тощо.

1. Продуктивність:

– система повинна обробляти запити користувачів без затримок (до 1 сек

для більшості дій);

– підтримка одночасної роботи не менше 1000 користувачів без зниження продуктивності;

– швидке завантаження сторінок (менше 2 сек).

2. Безпека:

– захист від SQL-ін’єкцій;

– валідація введених даних на клієнтському та серверному рівнях;

– контроль прав доступу згідно з роллю користувача.

3. Масштабованість:

– можливість розширення системи без необхідності кардинальної переробки архітектури;

– легке підключення додаткових модулів (наприклад, аналітика, інтеграція з API соцмереж).

4. Надійність:

– система має витримувати несподівані збої без втрати даних;

– регулярне збереження резервних копій бази даних;

– реалізація механізмів відновлення після збою.

5. Кросплатформеність:

– підтримка сучасних браузерів (Chrome, Firefox, Safari, Edge);

– адаптивний інтерфейс для використання на ПК, планшетах і смартфонах.

6. Зручність інтерфейсу (Usability):

– мінімалістичний, інтуїтивно зрозумілий інтерфейс;

– підтримка мультимовності;

– можливість швидкого доступу до часто використовуваних функцій;

– візуальні підказки, повідомлення про помилки, індикатори завантаження.

7. Технологічна сумісність:

– front-end реалізовано з використанням Angular (TypeScript);

– back-end реалізовано на основі Nest.js (TypeScript);

- реляційна база даних (Microsoft SQL);
- REST API для взаємодії клієнта і сервера;
- docker-контейнери для розгортання на сервері.

Сформульовані функціональні та нефункціональні вимоги створюють основу для подальшої архітектури системи та дозволяють забезпечити баланс між функціональністю, безпекою, зручністю використання й технічною ефективністю. Ці вимоги будуть покладені в основу формалізованої постановки задачі проектування (розділ 2.3) і визначатимуть критерії успішності реалізації системи «EventManager».

2.3 Постановка задачі проектування системи

Проєктування програмної системи «EventManager» передбачає створення інструменту для ефективного управління подіями з урахуванням широкого спектра функціональних та нефункціональних вимог, що були сформульовані на основі аналізу потреб користувачів. На цьому етапі необхідно чітко визначити архітектуру, ключові компоненти системи, моделі взаємодії між користувачами, внутрішню логіку та зовнішні інтеграції, які забезпечать досягнення поставлених цілей.

2.3.1 Мета проєктування

Основна мета проєктування – побудова масштабованої, надійної та зручної у використанні системи керування подіями, яка підтримує три основні категорії користувачів: неавторизовані відвідувачі, авторизовані користувачі та адміністратори. Кожна роль має доступ до окремого набору функцій, що повинні бути чітко обмежені й реалізовані з дотриманням принципів безпеки та логічної ізоляції.

2.3.2 Завдання, що вирішуються в процесі проєктування

У межах проєктування системи вирішуються такі ключові задачі:

- розробка моделі ролей і прав доступу: необхідно чітко розмежувати дії, доступні різним категоріям користувачів, реалізувати механізми аутентифікації

(login/signup) та авторизації (JWT/Session-based control);

- організація інтерфейсу користувача: побудова інтуїтивно зрозумілого, адаптивного і багато-лінгвального інтерфейсу за допомогою Angular, що дозволяє зручно переглядати, створювати та редагувати події, вести комунікацію через месенджер та здійснювати адміністративні дії;
- проєктування моделі подій: визначення структури події як об'єкта даних, її атрибутів (назва, дата, час, місце, формат, ціна, статус, автор, список учасників тощо) та зв'язків з іншими сутностями (користувачі, повідомлення, категорії);
- інтеграція месенджера в систему: реалізація базового функціоналу обміну повідомленнями в особистих і групових чатах, з підтримкою CRUD-операцій над повідомленнями та пошуку користувачів;
- адміністративний модуль: створення окремої панелі керування для адміністраторів, де можна переглядати та модифікувати всі події й акаунти, проводити аналітику (у вигляді графіків, таблиць, зведених звітів);
- збереження й обробка даних: проєктування реляційної бази даних, що ефективно зберігатиме інформацію про користувачів, події, повідомлення та дії в системі;
- забезпечення масштабованості та розширюваності: передбачити можливість подальшого розширення функціоналу без порушення цілісності системи.

2.3.3 Архітектурні підходи

Планується реалізація системи за принципом клієнт-серверної архітектури, де:

- front-end реалізується за допомогою Angular – сучасного фреймворку, який дозволяє створювати SPA (Single Page Application) з високою продуктивністю та адаптивною верстkою;
- back-end реалізується з використанням Nest.js – прогресивного серверного фреймворку на Node.js, що підтримує модульну структуру, впровадження залежностей, REST API;

– база даних – реляційна СУБД, що зберігає структуровану інформацію про події, користувачів, повідомлення, платежі;

– комунікація між клієнтом і сервером здійснюється через REST API.

2.3.4 Обмеження та припущення:

– система орієнтована на десктопні та мобільні браузери, окремий мобільний додаток не передбачається;

– початкова версія підтримує українську та англійську мови;

– у системі передбачено базовий рівень захисту даних користувача (використання HTTPS, JWT, хешування паролів);

– система не вимагає глибокої інтеграції з зовнішніми календарями чи соціальними мережами в першій версії.

2.3.5 Очікувані результати проєктування

У результаті реалізації задач проєктування має бути створено:

– детальну архітектурну модель системи (схеми, діаграми, моделі класів, ER-моделі);

– визначені інтерфейси взаємодії між модулями та ролями;

– формалізовані вимоги до UI/UX;

– структура бази даних, що враховує UML-діаграму та інше;

– концепція безпечного доступу і зберігання інформації;

– сценарії використання (Use Case) для кожної категорії користувачів.

2.4 Висновки до розділу

У ході проведеного дослідження та аналізу вимог користувачів було встановлено ключові потреби та очікування, які має задовольняти розроблювана система управління подіями «EventManager». Збір та систематизація вимог дозволили сформувати чітке уявлення про функціональні можливості платформи, включно з підтримкою різних ролей користувачів, реєстрацією, створенням і редагуванням подій, організацією комунікації через месенджер, а також адміністративним контролем.

Формулювання функціональних та нефункціональних вимог допомогло визначити основні технічні параметри системи, такі як масштабованість, безпека, зручність користування та інтеграція з платіжними сервісами. Це, у свою чергу, слугувало надійною основою для постановки задачі проектування, що охопила не лише розробку архітектури і моделей даних, а й визначення принципів взаємодії між компонентами системи та ролями користувачів.

Постановка задачі проектування висвітлила основні напрямки роботи, включаючи побудову клієнт-серверної архітектури з використанням сучасних технологій Angular та Nest.js, розробку інтуїтивно зрозумілого інтерфейсу, а також реалізацію захищених механізмів авторизації й аутентифікації. Особлива увага була приділена забезпеченню безпеки даних і комфортному досвіду користувачів різного рівня доступу.

Отже, результати проведеного аналізу та проектних рішень створюють ґрунтовну базу для подальшої реалізації системи, що відповідатиме сучасним вимогам ринку та зможе ефективно підтримувати управління подіями, забезпечуючи надійну та зручну взаємодію між всіма учасниками процесу.

РОЗДІЛ 3. ПРОЄКТУВАННЯ СИСТЕМИ

3.1 Розробка архітектури програмного забезпечення

Проєктування архітектури є одним з найважливіших етапів розробки інформаційної системи, адже саме архітектура визначає логічну організацію компонентів, взаємодію між ними, а також забезпечує масштабованість, надійність, безпеку та підтримуваність розробленого програмного забезпечення. В межах системи «EventManager» розробка архітектури базується на багаторівневому підході з використанням принципів модульності, інкапсуляції, відокремлення відповідальностей та REST-орієнтованої взаємодії між клієнтом і сервером.

3.1.1. Архітектурний стиль

Основною архітектурною моделлю обрано клієнт-серверну архітектуру з чітким розділенням Front-end та Back-end частин, що взаємодіють через REST API [20]. Такий підхід дозволяє незалежно розробляти, масштабувати та оновлювати обидві частини системи, а також забезпечити гнучкість у виборі технологій.

На стороні клієнта реалізується односторінковий застосунок (SPA) за допомогою фреймворку Angular, який забезпечує динамічне оновлення вмісту сторінки без необхідності повного перезавантаження. Серверна частина, побудована на основі Nest.js, виконує роль обробника запитів, контролю доступу, логіки взаємодії з базою даних, а також реалізує всі REST API для клієнта.

3.1.2. Логічна структура системи

Логічна структура передбачає наявність таких основних модулів:

– модуль автентифікації та авторизації – забезпечує реєстрацію, вхід, перевірку ролей користувачів (JWT-токени);

– модуль управління подіями – реалізує створення, редагування, перегляд, фільтрацію подій, обробку оплат;

– модуль користувача – містить логіку редагування профілю, відображення особистих даних, керування обліковим записом;

– месенджер – відповідає за створення, обслуговування чатів, зберігання

повідомлень та взаємодію;

– модуль адміністратора – дає змогу адміністратору переглядати статистику, керувати подіями та користувачами;

– база даних – централізоване збереження всіх об'єктів доменної моделі: користувачі, події, повідомлення, ролі.

Діаграму компонентів зображено на рисунку 3.1.

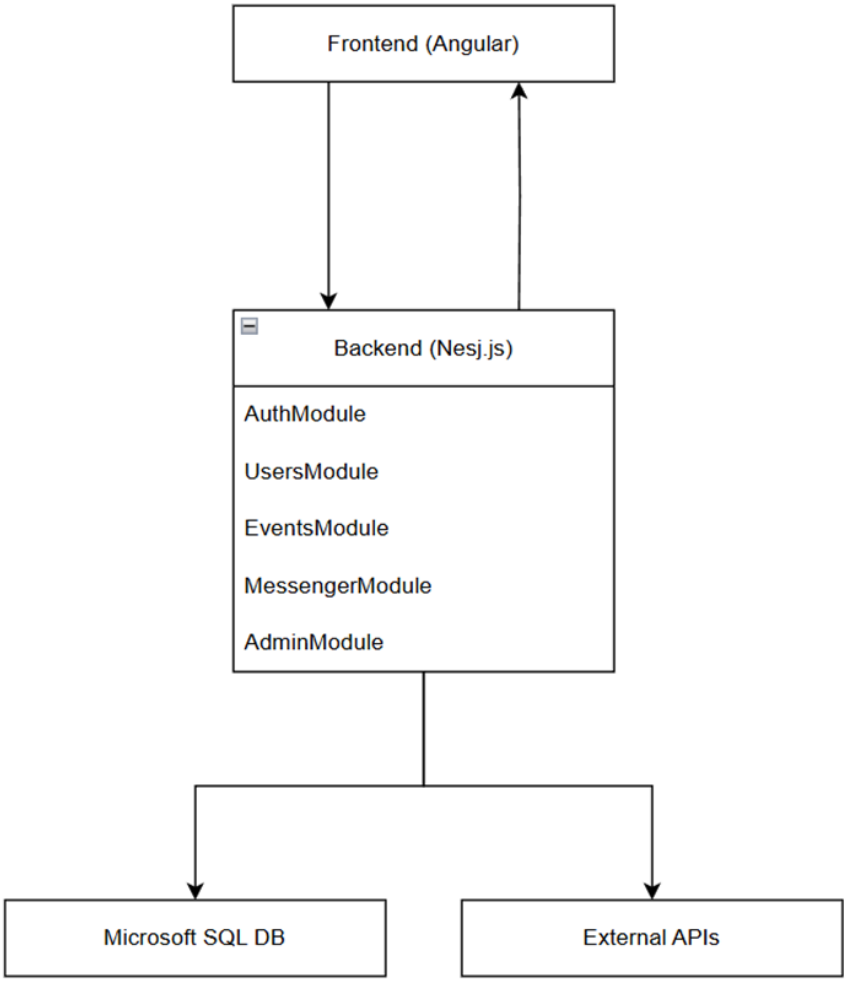

Рисунок 3.1 – Діаграма компонентів

3.1.3. Діаграма розгортання

Для наочного представлення фізичної структури системи «EventManager» доцільно використати діаграму розгортання (Deployment Diagram), яка відображає взаємодію між апаратними компонентами та програмними модулями.

В даній системі передбачено такі основні вузли розгортання:

– клієнтський пристрій (браузер користувача) – виконує Angular SPA, здійснює запити до серверу;

– веб-сервер (Back-end) – сервер, на якому працює додаток Nest.js, обробляє REST API та WebSocket-з’єднання;

– сервер бази даних – зберігає усі дані системи в реляційній СУБД (Microsoft SQL);

– сервер аутентифікації – використовується для авторизації через зовнішні сервіси;

– система зберігання медіа або файлів – для збереження додаткових даних, якщо це передбачено.

Діаграма розгортання демонструє мережеві з'єднання між цими вузлами, а також програмні артефакти, які розміщені на відповідних серверах (зображено на рисунку 3.2).

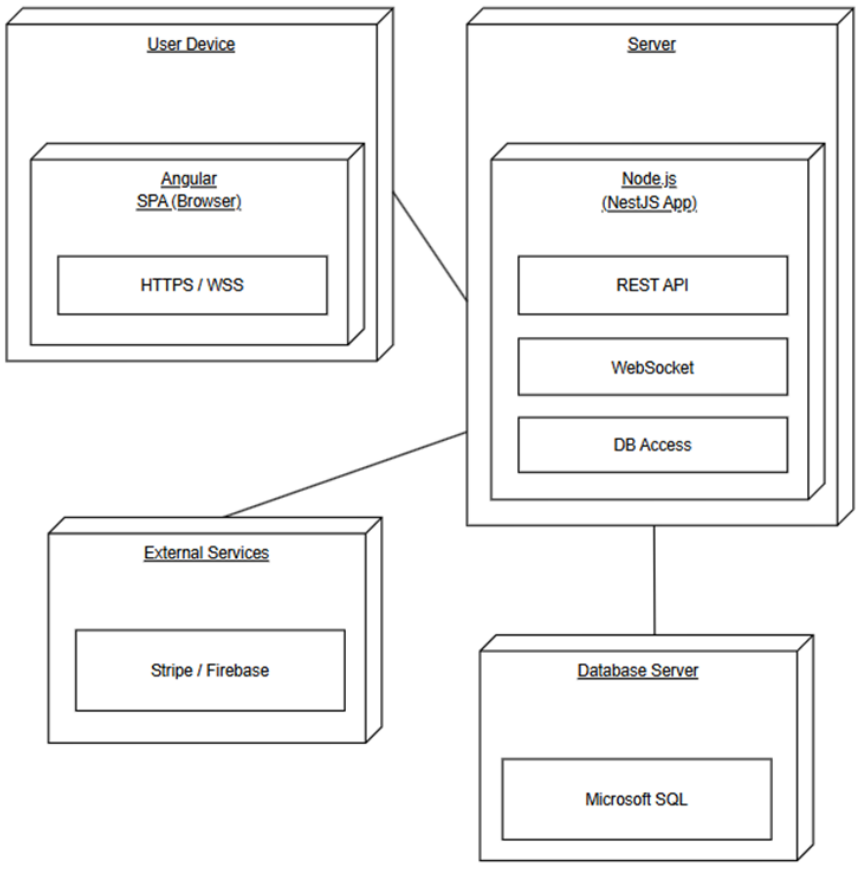

Рисунок 3.2 – Діаграма розгортання

3.2 Моделювання бізнес-процесів та системних компонентів

3.2.1. Базова структура класів

Для опису основних об’єктів системи розроблено набір класів, які відображають сутності, що використовуються у функціоналі управління подіями, користувачами та обміном повідомленнями:

- user – клас, який описує користувача системи. Має унікальний ідентифікатор, ім'я користувача, email, захищений пароль (хеш), роль (гостьовий, авторизований користувач або адміністратор), а також списки подій, створених користувачем, подій, на які він зареєстрований, і чатів, у яких він бере участь;
- event – клас, що описує подію з такими атрибутами: заголовок, опис, дата і місце проведення, ознака платності та, за необхідності, вартість. Подія має автора (користувача, який її створив) і список учасників;
- authCredentialsDto – об’єкт передачі даних, що використовується під час авторизації або реєстрації користувача (електронна пошта та пароль);
- chat – клас для організації обміну повідомленнями між користувачами. Може бути як індивідуальним, так і груповим, містить учасників і список повідомлень;
- message – одиниця повідомлення в чаті, що містить текст, відправника, посилання на чат, часову позначку, а також опціональне посилання на повідомлення, на яке здійснено відповідь.

UML-діаграму класів зображено на рисунку 3.3.

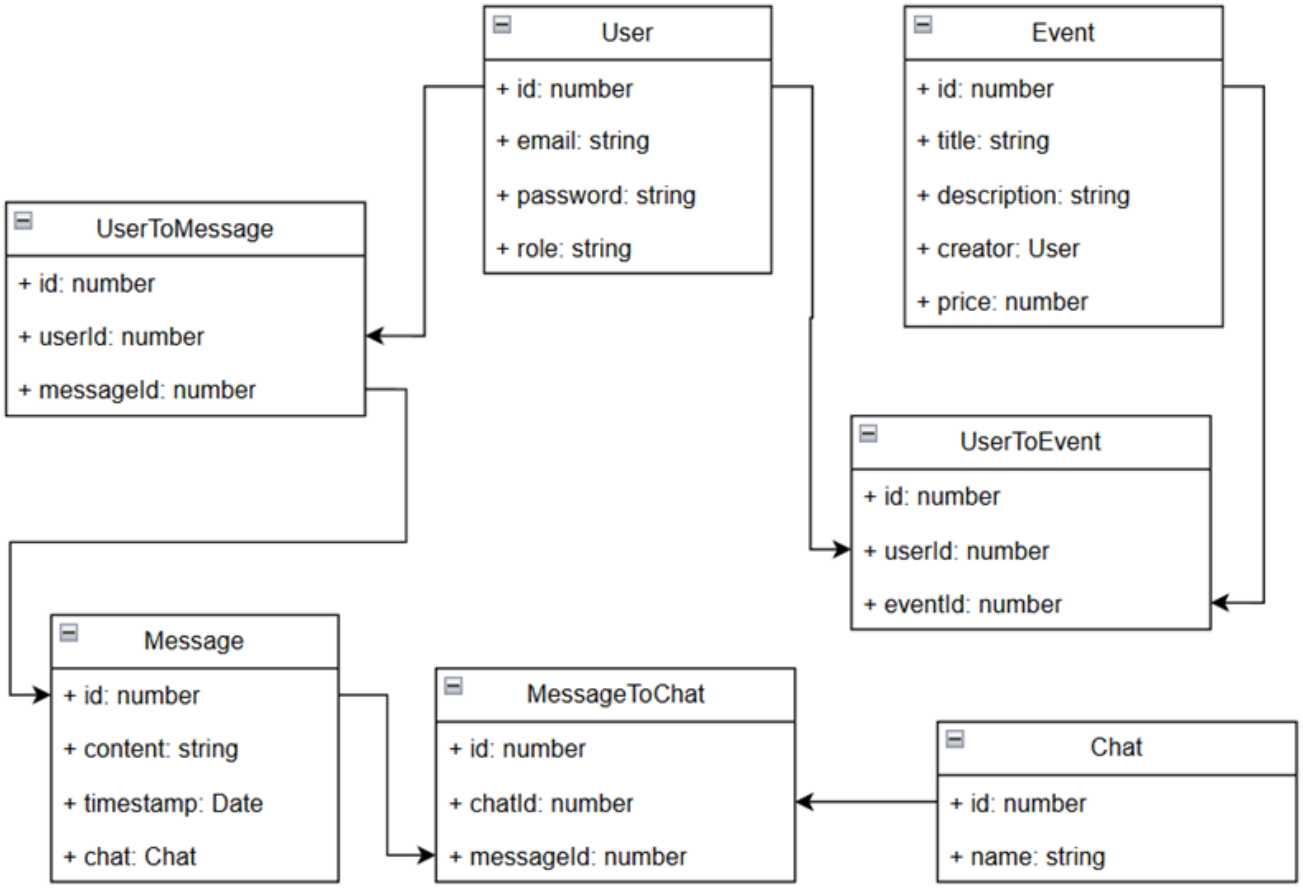

Рисунок 3.3 – Діаграма класів

3.2.2. Діаграма діяльності: надсилання повідомлення

Процес обміну повідомленнями є однією з найважливіших функцій у системі «EventManager», яка забезпечує оперативну комунікацію між користувачами. Для візуалізації логіки взаємодії між компонентами під час надсилання повідомлення доцільно застосувати діаграму діяльності (Activity Diagram).

Діаграма демонструє загальний сценарій надсилання повідомлення в чаті, включаючи обробку на фронтенді, серверну логіку та роботу з базою даних. Основна увага приділяється умовам валідації, перевірці наявності чату та авторства, а також подальшому збереженню та розсилці повідомлення.

Основні етапи процесу:

1. Користувач вводить текст повідомлення в чаті.
2. Натискає кнопку "Надіслати".
3. Frontend надсилає повідомлення через WebSocket на сервер.

4. Backend отримує повідомлення.

5. Виконується валідація:

– якщо валідація не пройдена (наприклад, порожній текст), повідомлення не надсилається, процес завершується;

– якщо валідація пройдена, виконується наступна перевірка.

6. Перевірка: чи є автор користувачем системи?

– якщо ні – повідомлення відхиляється або зберігається з відповідною поміткою;

– якщо так – перевіряється, чи існує чат.

7. Якщо чат існує – повідомлення зберігається в базу даних.

8. Якщо чат не існує – створюється новий запис чату (за необхідності), і повідомлення зберігається.

9. Після збереження повідомлення виконується розсилка повідомлення усім учасникам чату через WebSocket.

10. Процес завершено.

Діаграма діяльності дозволяє проаналізувати логіку обробки повідомлень у системі та забезпечує чітке розуміння алгоритму дій, включаючи ключові точки прийняття рішень. Такий підхід сприяє виявленню потенційних помилок і допомагає формалізувати вимоги до модулів обробки повідомлень.

Діаграму діяльності при відправці повідомлення зображено на рисунку 3.4.

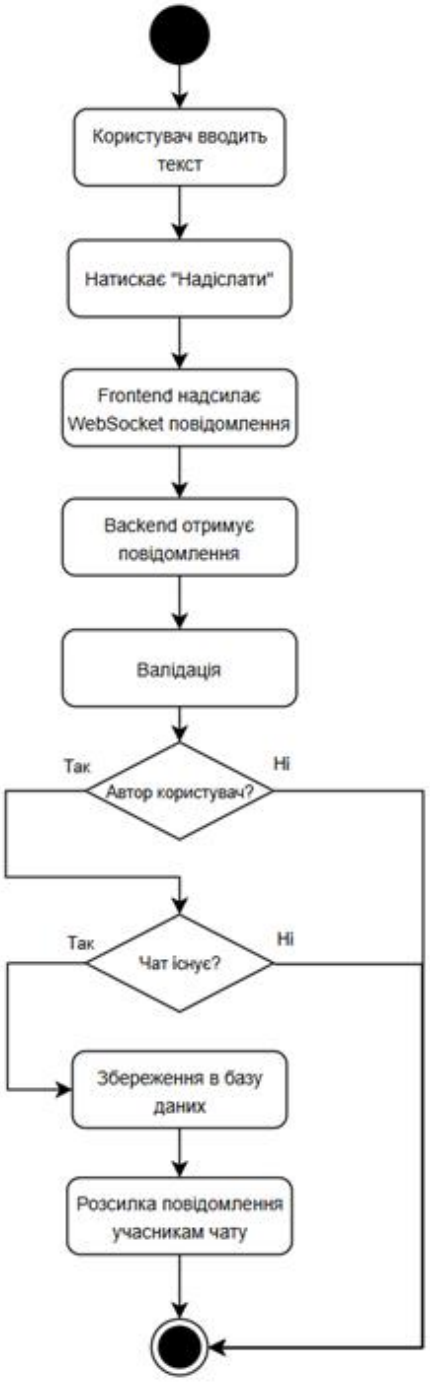

Рисунок 3.4 – Діаграма діяльності: надсилання повідомлення

3.1.4. Діаграма послідовностей (реєстрація користувача)

Однією з ключових функцій системи «EventManager» є реєстрація нового користувача. Для кращого розуміння взаємодії між компонентами під час цього процесу доцільно застосувати діаграму послідовностей (Sequence Diagram), яка демонструє порядок викликів між об’єктами системи.

У даній діаграмі задіяні такі основні учасники:

- User – кінцевий користувач, який заповнює форму реєстрації;
- Frontend – клієнтська частина застосунку на Angular, яка надсилає дані на сервер;
- AuthController – контролер на стороні Nest.js, що приймає запит;
- AuthService – сервіс авторизації, який обробляє логіку реєстрації;
- UserRepository – шар доступу до бази даних, відповідальний за збереження нового користувача.

Послідовність взаємодії:

1. User заповнює форму реєстрації на клієнтському інтерфейсі.
2. Frontend надсилає HTTP-запит POST /register з даними форми.
3. AuthController приймає запит і викликає метод validateAndCreateUser() у AuthService.
4. AuthService виконує перевірку вхідних даних, хешує пароль і створює новий об’єкт користувача.
5. AuthService викликає метод save(user) у UserRepository, щоб зберегти користувача в базі даних.
6. Після успішного збереження UserRepository повертає відповідь сервісу.
7. AuthService формує підтвердження успішної реєстрації.
8. AuthController надсилає відповідь 201 Created назад на Frontend.
9. Frontend виводить повідомлення "Успішна реєстрація" користувачу.

Діаграму послідовностей реєстрації користувача зображено на рисунку 3.5.

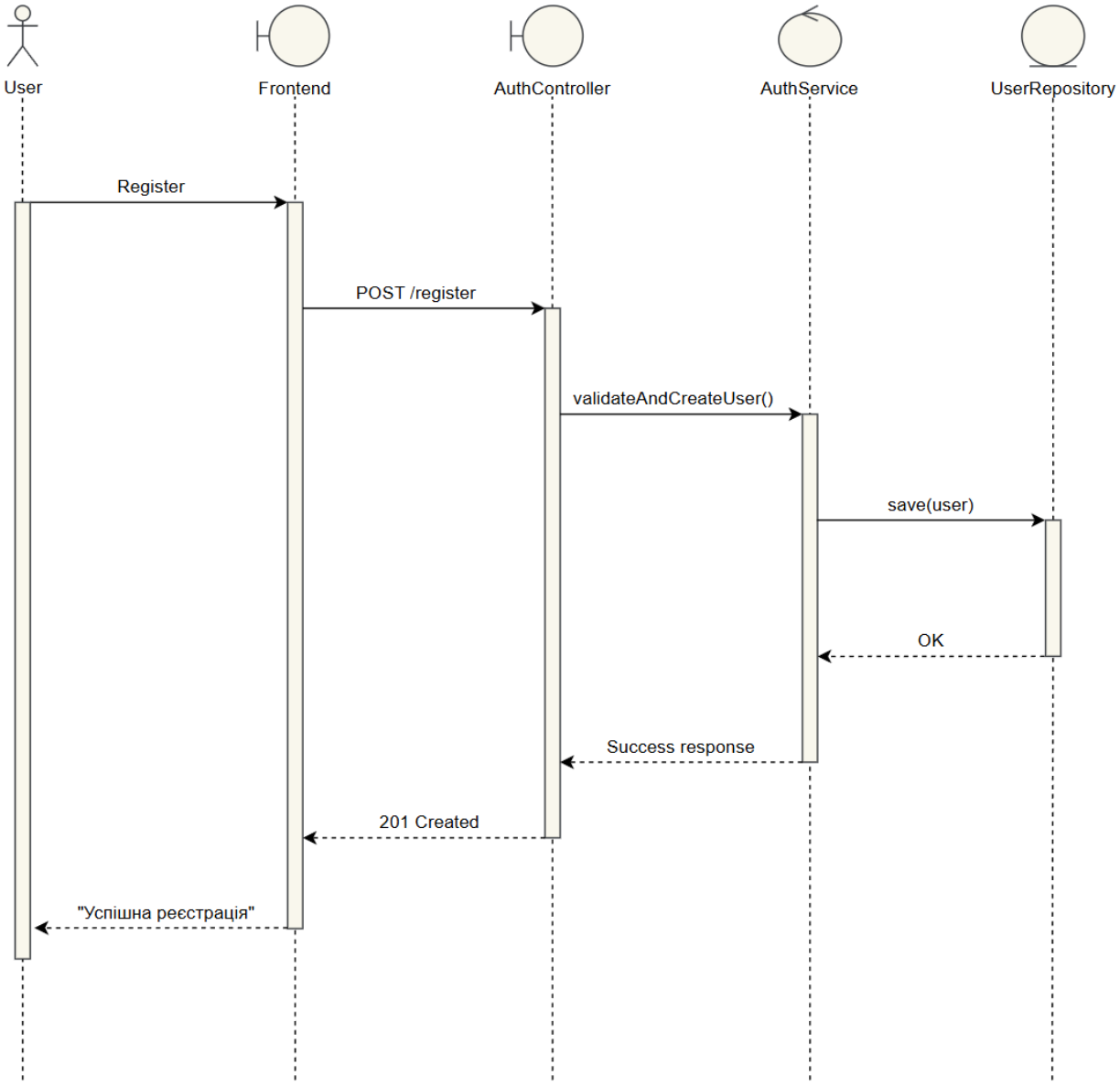

Рисунок 3.5 – Діаграма послідовностей: реєстрація користувача

3.3 Вибір та обгрунтування технологій та інструментів розробки

3.3.1. Загальний підхід до вибору технологій

Процес вибору технологій для реалізації програмного застосунку «EventManager» ґрунтується на поєднанні низки технічних, функціональних та організаційних критеріїв. Враховувалися не лише можливості реалізації необхідного функціоналу, але й масштабованість, продуктивність, безпека, простота підтримки, а також доступність ресурсів і рівень популярності технологій у розробницькій спільноті.

До основних критеріїв, за якими здійснювався вибір технологій, належать:

- функціональна відповідність: вибрані технології повинні забезпечувати реалізацію усіх запланованих функцій системи, включно з підтримкою аутентифікації користувачів, організацією подій, інтерактивною комунікацією (месенджером), обробкою оплат тощо;
- модульність і масштабованість: архітектура має дозволяти легке розширення системи в майбутньому – як у плані функціональності, так і у кількості користувачів;
- продуктивність і швидкодія: застосунок повинен працювати стабільно за умов високого навантаження, наприклад, під час одночасного надсилання повідомлень або масової реєстрації на події;
- безпека: враховуючи обробку персональних даних користувачів, необхідно обрати технології, що підтримують сучасні механізми захисту, зокрема шифрування, авторизацію за токенами, обмеження доступу за ролями;
- популярність і підтримка спільноти: перевага надавалась інструментам із широким використанням у розробницькій практиці, наявною документацією та активною спільнотою, що спрощує вирішення потенційних проблем під час реалізації;
- єдине середовище розробки: було прийнято рішення використовувати TypeScript [11] як універсальну мову для клієнтської (Angular) та серверної частини (Nest.js), що спрощує обмін логікою, зменшує ймовірність помилок і покращує підтримку проєкту;
- гнучкість у виборі бібліотек та інтеграцій: проєкт передбачає інтеграцію з зовнішніми API (наприклад, для оплати або розсилки повідомлень), тому технології повинні бути адаптивними до таких потреб.

На основі вказаних критеріїв було сформовано технологічний стек, який максимально відповідає задачам та обсягу проєкту.

3.3.2. Front-end: Angular

Для реалізації клієнтської частини програмного забезпечення було обрано фреймворк Angular, який є одним з найпотужніших та найпопулярніших

інструментів для побудови динамічних вебзастосунків. Angular розробляється та підтримується компанією Google, що забезпечує регулярне оновлення, розширену документацію та стабільність у довготривалих проєктах.

Angular є фреймворком з відкритим вихідним кодом, написаним мовою TypeScript, що дозволяє використовувати сильну типізацію, об'єктно-орієнтоване програмування та сучасні парадигми розробки. У межах проєкту «EventManager» Angular було обрано з таких причин:

– модульна структура: Angular підтримує поділ коду на окремі модулі, що дозволяє чітко організувати логіку та розмежувати відповідальність між різними частинами застосунку (автентифікація, робота з подіями, месенджер тощо);

– двостороннє зв’язування даних (two-way data binding): забезпечує автоматичну синхронізацію моделі та представлення, що особливо корисно при динамічній взаємодії користувача з формами, списками подій, повідомленнями тощо;

– компонентна архітектура: увесь інтерфейс застосунку побудований із багаторазових компонентів, що спрощує розробку, повторне використання та тестування елементів UI;

– маршрутизація (Routing): Angular Router дозволяє легко реалізувати переходи між сторінками (наприклад, головна сторінка, сторінка події, сторінка профілю, чат тощо), забезпечуючи логіку захисту маршрутів відповідно до ролі користувача;

– інтеграція з сервісами HTTP: Angular надає потужний інструмент HttpClient для взаємодії з серверною частиною (Nest.js API) – для отримання даних, надсилання форм, автентифікації користувачів, оновлення профілю, тощо;

– RxJS [17] (Reactive Extensions for JavaScript): Angular тісно інтегрований з бібліотекою RxJS, яка дозволяє ефективно обробляти асинхронні потоки даних – наприклад, підписку на нові повідомлення, оновлення подій, підключення до WebSocket-сервісів у реальному часі.

Також у межах реалізації інтерфейсу використовувалися такі додаткові бібліотеки:

– Angular Material – бібліотека UI-компонентів, яка забезпечує сучасний вигляд елементів інтерфейсу відповідно до принципів Material Design;

– ngx-translate – для підтримки багатомовного інтерфейсу;

– FormBuilder та Reactive Forms – для створення форм з валідацією (реєстрація, створення події тощо);

– SCSS – для стилізації з підтримкою змінних, міксинів та вкладеності стилів.

Розробка на Angular дозволить досягти високого рівня інтерактивності застосунку, забезпечити швидкий відгук інтерфейсу, чітке розмежування ролей користувачів та масштабовану структуру, зручно підтримувану у процесі розширення функціоналу

3.3.3. Back-end: Nest.js

Для реалізації серверної частини програмного забезпечення було обрано фреймворк Nest.js, який є сучасним інструментом для створення масштабованих та ефективних серверних застосунків на основі Node.js та мови TypeScript.

Nest.js є фреймворком, що базується на принципах об'єктно-орієнтованого програмування (ООП), функціонального програмування (ФП) та метапрограмування, і водночас підтримує архітектуру з використанням декораторів, подібно до таких фреймворків, як Angular. Це робить Nest.js зручним для розробників, які вже знайомі з Angular, та забезпечує уніфікований підхід до структурування коду у front-end і back-end частинах

У межах проєкту «EventManager» Nest.js використовується для реалізації таких функціональних можливостей:

– обробка HTTP-запитів та REST API – Nest.js надає зручний інструментарій для створення контролерів, що обробляють запити від клієнтської частини (автентифікація, події, чати, повідомлення тощо);

– сервіси (Services) – логіка бізнес-процесів, наприклад, створення події,

реєстрація користувача, надсилання повідомлення або перевірка прав доступу, реалізована в окремих сервісах, що полегшує повторне використання та тестування;

– модульна архітектура – Nest.js дозволяє розділяти код на модулі (наприклад, AuthModule, EventsModule, UsersModule, ChatModule), що забезпечує чітку структуризацію, масштабованість і зручність підтримки;

– робота з базами даних – для взаємодії з реляційною базою даних використовується бібліотека TypeORM, яка інтегрується з Nest.js. Це забезпечує зручне створення сутностей, зв’язків між таблицями, автоматичну міграцію даних та роботу з репозиторіями;

– безпека та авторизація – Nest.js підтримує реалізацію middleware, guard-об'єктів та інтерсепторів для перевірки JWT-токенів, обмеження доступу до ресурсів відповідно до ролі користувача (гостьовий, звичайний користувач, адміністратор).

Завдяки широким можливостям Nest.js, серверна частина застосунку буде реалізована з дотриманням принципів чистої архітектури, забезпеченням безпеки, масштабованості та підтримки в реальному часі. Обрана технологія дозволяє ефективно обробляти запити, керувати взаємодією між компонентами системи та підтримувати майбутні оновлення без значних змін у кодовій базі.

3.3.4. База даних: Microsoft SQL

У рамках реалізації серверної частини системи «EventManager» для зберігання, обробки та управління структурованими даними було обрано Microsoft SQL Server (MS SQL) – потужну реляційну систему керування базами даних (СКБД), що поєднує високу продуктивність, масштабованість та розширені можливості безпеки.

Вибір MS SQL обґрунтовується такими перевагами:

– реляційна модель даних, що ідеально підходить для структурованих об'єктів, які мають чітко визначені зв’язки (наприклад, зв'язки між користувачами, подіями, повідомленнями та чатами);

– підтримка транзакцій забезпечує цілісність даних, особливо при

операціях, що охоплюють декілька таблиць (наприклад, реєстрація користувача на подію із одночасною оплатою);

– інтеграція з TypeORM (ORM-інструмент у Nest.js) – дозволяє легко працювати з таблицями як з об'єктами, виконувати складні запити через TypeScript, а також автоматизувати міграції схем БД;

– масштабованість і продуктивність, що дозволяє підтримувати зростання кількості користувачів, подій і повідомлень без втрати швидкодії системи.

3.3.5 Інші інструменти та сервіси

У процесі розробки програмного застосунку «EventManager» було використано низку допоміжних інструментів та сервісів, що забезпечують ефективну організацію командної роботи, контроль версій, налаштування середовища розробки, а також можливість масштабування та розгортання системи в майбутньому.

Для збереження історії змін у кодовій базі та забезпечення командної роботи використано систему контролю версій Git [15], а також хостинг репозиторіїв – GitHub [19]. Це дозволило:

– вести незалежну розробку окремих функціональностей у гілках (branches);

– контролювати внесення змін через pull request-и та рев’ю коду;

– забезпечити зручний інтерфейс для керування змінами за допомогою GitHub Desktop, особливо для візуального контролю комітів, конфліктів та історії змін без використання CLI.

Відповідно створено репозиторій та клоновано його завдяки GitHub Dekstop (зображено на рисунку 3.6).

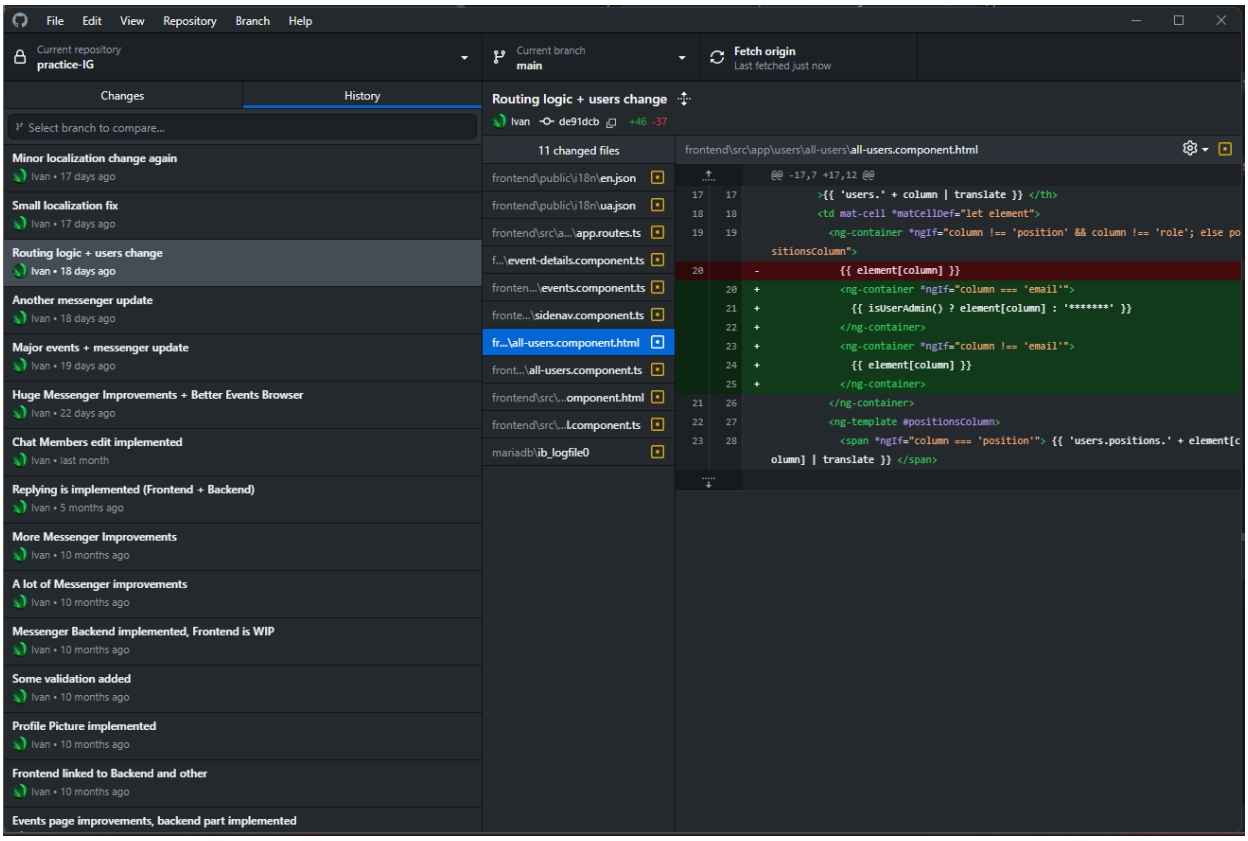

Рисунок 3.6 – Вигляд репозиторію в GitHub Dekstop

IntelliJ IDEA [13] є однією з найпопулярніших інтегрованих середовищ розробки (IDE) для програмування, розроблених компанією JetBrains. Ця IDE спеціально створена для підтримки мов програмування, таких як Java, Kotlin, Groovy, Scala, Python, PHP, JavaScript, і багатьох інших.

IntelliJ IDEA надає розширені можливості для розробки програмного забезпечення, що допомагають розробникам писати код швидше і ефективніше. Деякі з основних функцій IntelliJ IDEA включають:

1. Автодоповнення: IDE надає контекстно-залежне автодоповнення коду, що спрощує написання коду і зменшує кількість помилок.

2. Рефакторинг: IntelliJ IDEA має широкий набір інструментів для рефакторингу коду, які допомагають поліпшити структуру, читабельність та ефективність коду.

3. Вбудовані інструменти аналізу: IDE надає можливості для аналізу коду, виявлення помилок, попереджень і непотрібного коду, що допомагає вдосконалювати якість програмного забезпечення.

4. Відлагодження: IntelliJ IDEA має потужний відлагоджувач, який дозволяє крокувати через код, переглядати значення змінних, виконувати вирази та виявляти помилки.

5. Керування версіями: IDE має інтегровану підтримку систем контролю версій, таких як Git, що дозволяє зручно працювати з репозиторіями і виконувати операції зі змінами.

6. Інструменти для роботи з базами даних: IntelliJ IDEA має підтримку різних баз даних і надає інструменти для розробки, керування та відлагодження SQL-запитів.

7. Підтримка інших технологій: Крім мов програмування, IntelliJ IDEA надає підтримку для фреймворків та технологій, таких як Spring, Android, HTML/CSS, JavaScript, Angular, і багато інших.

IntelliJ IDEA має інтуїтивний і зручний інтерфейс (зображено на рисунку 3.7), що дозволяє розробникам працювати ефективно. IDE також має широкий вибір плагінів, які можна встановити для розширення функціональності.

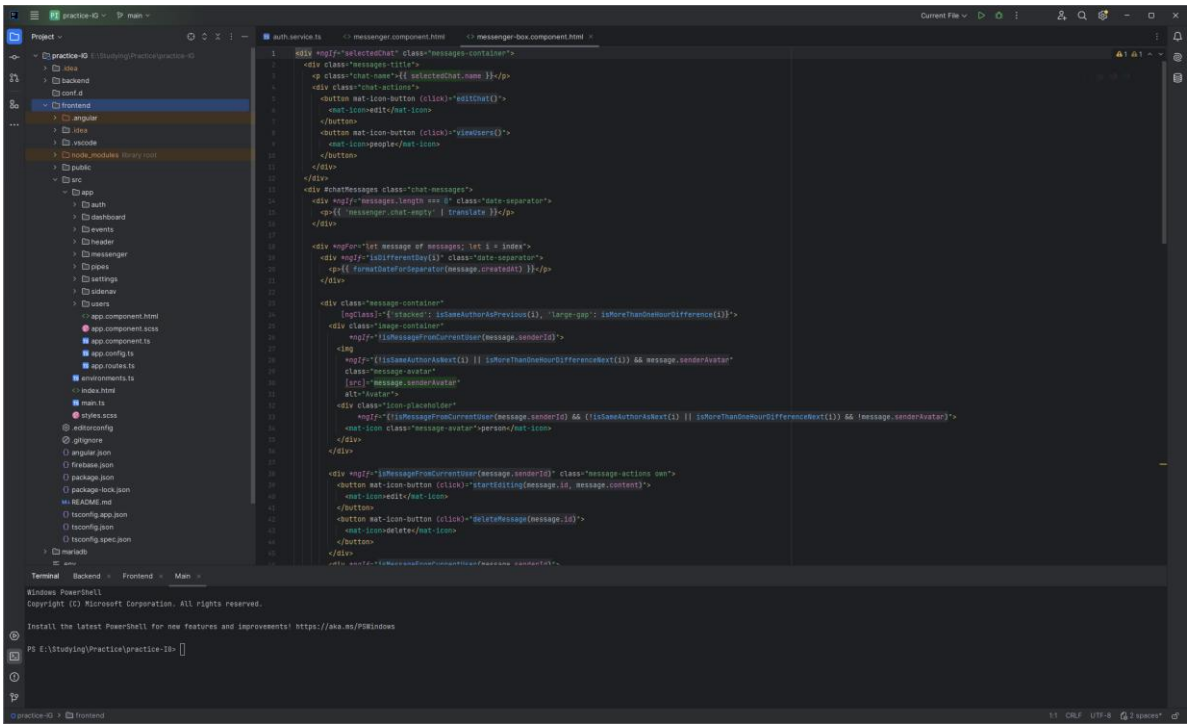

Рисунок 3.7 – Зовнішній вигляд середовища розробки IntelliJ IDEA

Узагальнюючи, IntelliJ IDEA є потужним інструментом для розробки програмного забезпечення, який допомагає розробникам писати якісний код швидше і ефективніше.

3.4 Висновки по розділу

У цьому розділі було виконано проєктування інформаційної системи «EventManager», що є критичним етапом для забезпечення її коректної архітектури, структурованості та подальшої реалізації.

На етапі розробки архітектури програмного забезпечення (підпункт 3.1) було обґрунтовано доцільність застосування клієнт-серверної архітектури з поділом на front-end (Angular) та back-end (Nest.js), що забезпечує масштабованість, розділення відповідальностей та зручність обслуговування.

Проведено моделювання основних бізнес-процесів та системних компонентів. Створено діаграму класів, яка відображає основні об'єкти системи та їх взаємозв'язки, а також діаграми діяльності та послідовностей для опису ключових сценаріїв використання (наприклад, надсилання повідомлення та реєстрація користувача).

Здійснено вибір і обґрунтування технологічного стеку. Було підтверджено доцільність використання Angular для реалізації клієнтської частини, Nest.js – для серверної логіки, Microsoft SQL Server – для зберігання структурованих даних, а також таких допоміжних інструментів, як GitHub та IntelliJ IDEA. Усі вибрані технології забезпечують надійність, зручність підтримки та можливість розширення функціональності системи в майбутньому.

Таким чином, результати цього розділу формують повноцінну основу для реалізації функціонального прототипу системи в наступному етапі – безпосередній розробці та тестуванні програмного застосунку.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ПРОЕКТУ

4.1 Опис розробки програмного коду

4.1.1. Фронтенд частина (Angular)

Архітектура фронтенду організована модульно, з чітким розділенням компонентів за функціональним призначенням (зображено на рисунку 4.1):

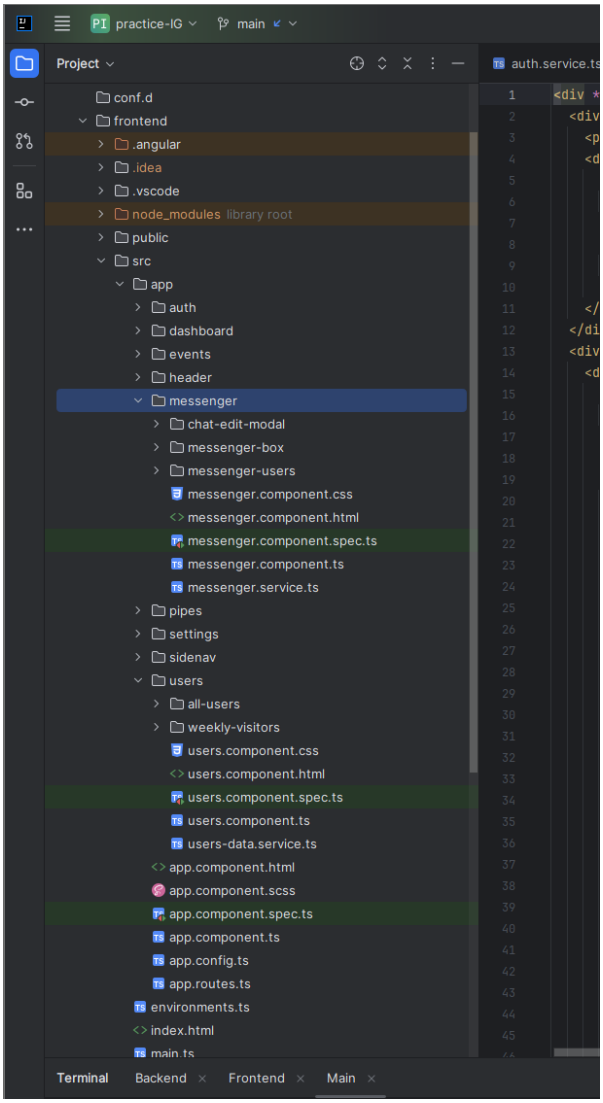

Рисунок 4.1 – Структура Front-end частини застосунку

В модулі auth відбувається авторизація користувача:

– в компоненті login (зображено на рисунку 4.2) реалізовано форму входу

та взаємодію з auth.service.ts;

```
signIn(formValue: any) {
this.http.post(`${environment.apiUrl}/auth/login`, { email: formValue.email,
password: formValue.password })
.subscribe({
next: (res: any) => {
this.snackbarMessage('Sign in successful!');
localStorage.setItem('access_token', res.access_token);
this.isAuth.next(true);
this.router.navigate(['/dashboard']).then(r => r);
},
error: (error) => {
this.snackbarMessage('Sign in failed');
}
});
}
```

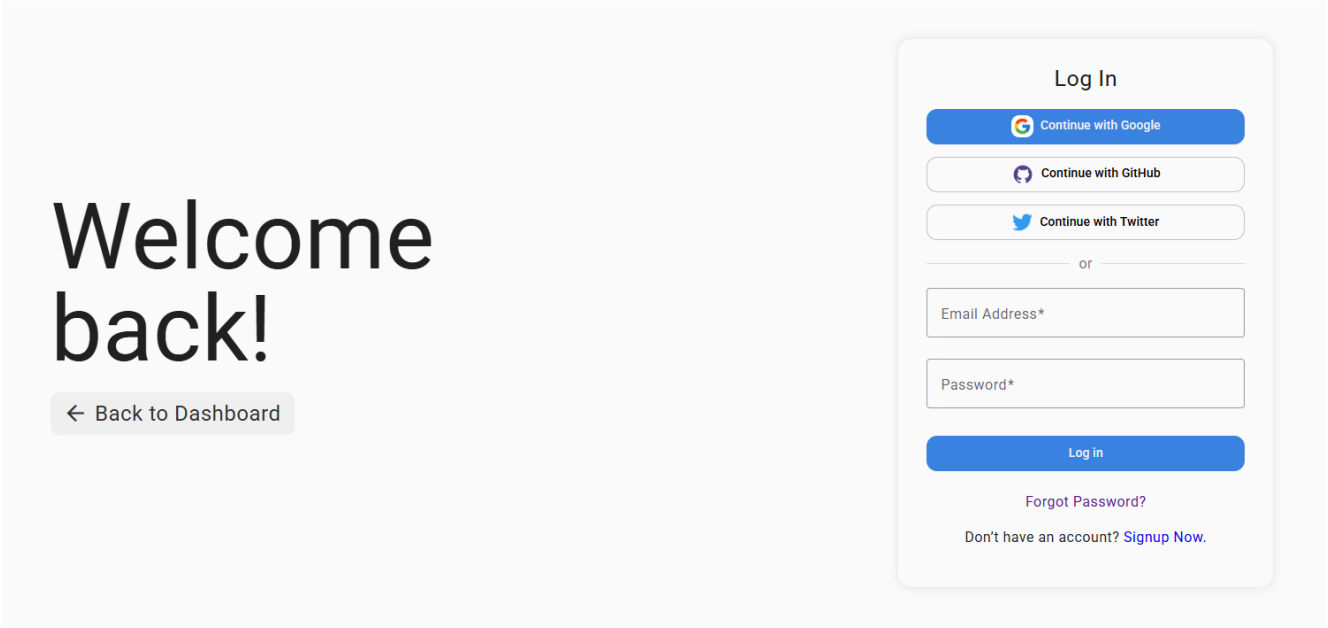

Рисунок 4.2 – Вигляд вікна реєстрації або логіну

– в файлі auth.guard.ts реалізовано захист маршруту для авторизованих користувачів;

```
export const authGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const router = inject(Router);

  if (!authService.isAuth.value) {
    router.navigate(['/login']);
    return false;
  }

  return true;
};
```

– is-signed-in.guard.ts використовується для обмеження доступу до

сторінок авторизації вже авторизованим користувачам;

```
export const isSignedInGuard:
  CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const router = inject(Router);

  if (authService.isAuth.value) {
    router.navigate(['/dashboard']);
    return false;
  }

  return true;
};
```

Модуль events є ключовим у функціоналі системи, він відповідає за створення, перегляд (зображено на рисунку 4.3), редагування та публікацію подій;

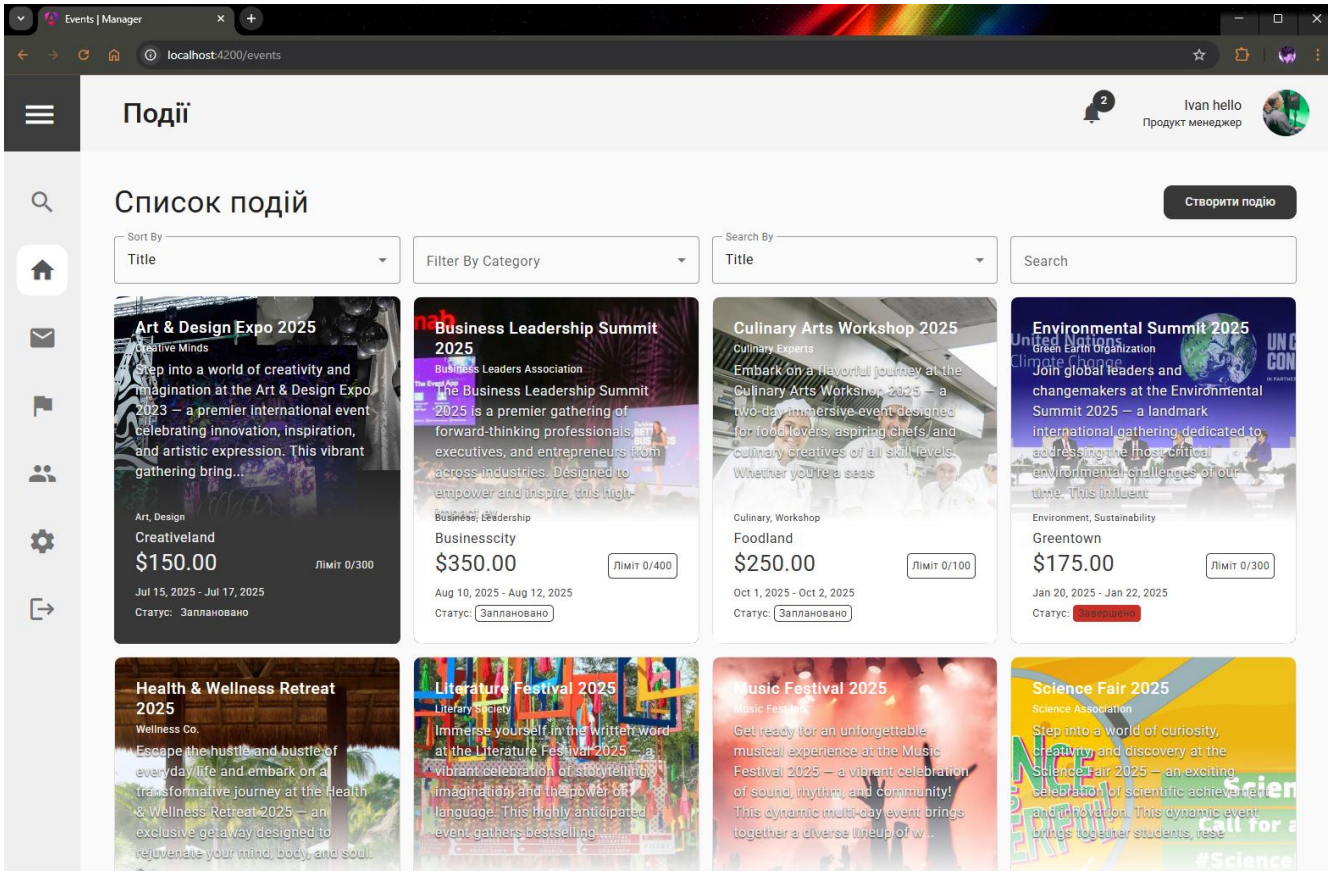

Рисунок 4.3 – Перегляд подій

– компонент create-event містить форму для створення нової події (зображено на рисунку 4.4), яка передає дані до сервісу event.service.ts;

```
createEvent() {
  if (this.eventForm.valid) {
    const newEvent = this.eventForm.value;
    this.eventService.addEvent(newEvent);
```

```
      this.eventForm.reset();
    }
  }
```

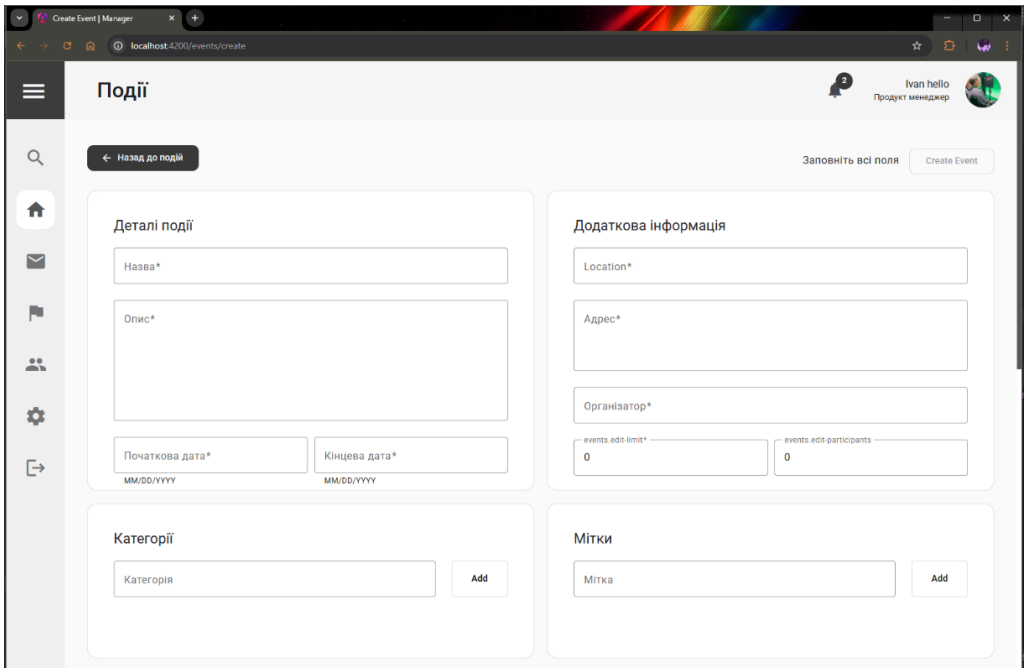

Рисунок 4.4 – Вигляд форми створення подій

– у компоненті event-details реалізовано перегляд детальної інформації про подію (зображено на рисунку 4.5);

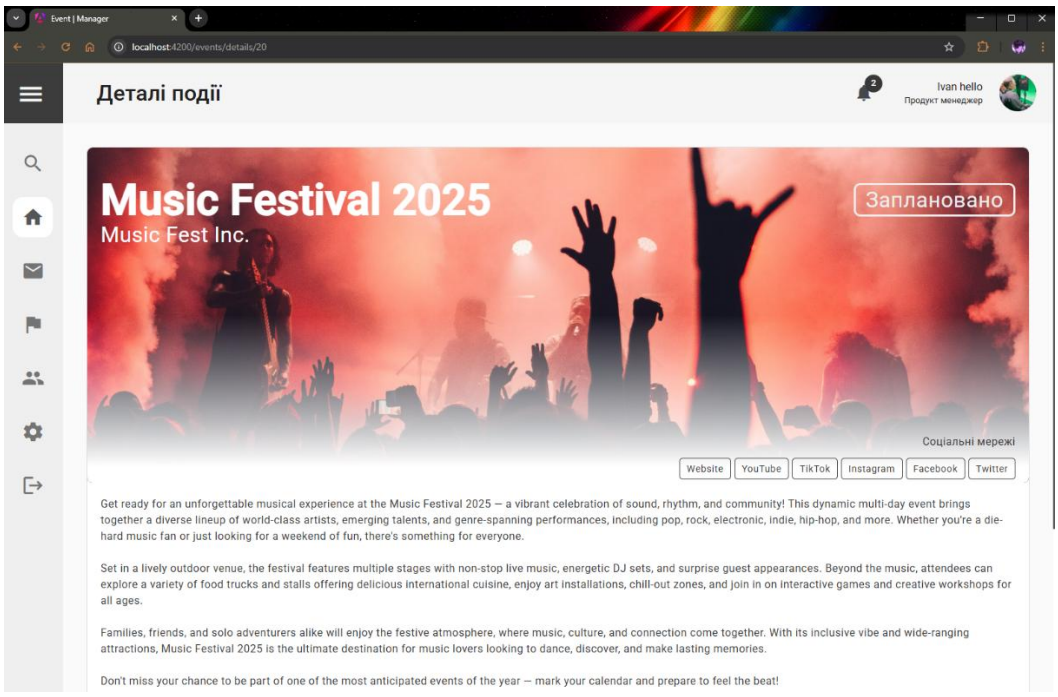

Рисунок 4.5 – Вигляд сторінки події

– компонент dashboard-charts будує графіки відвідуваності та активності подій (зображено на рисунку 4.6);

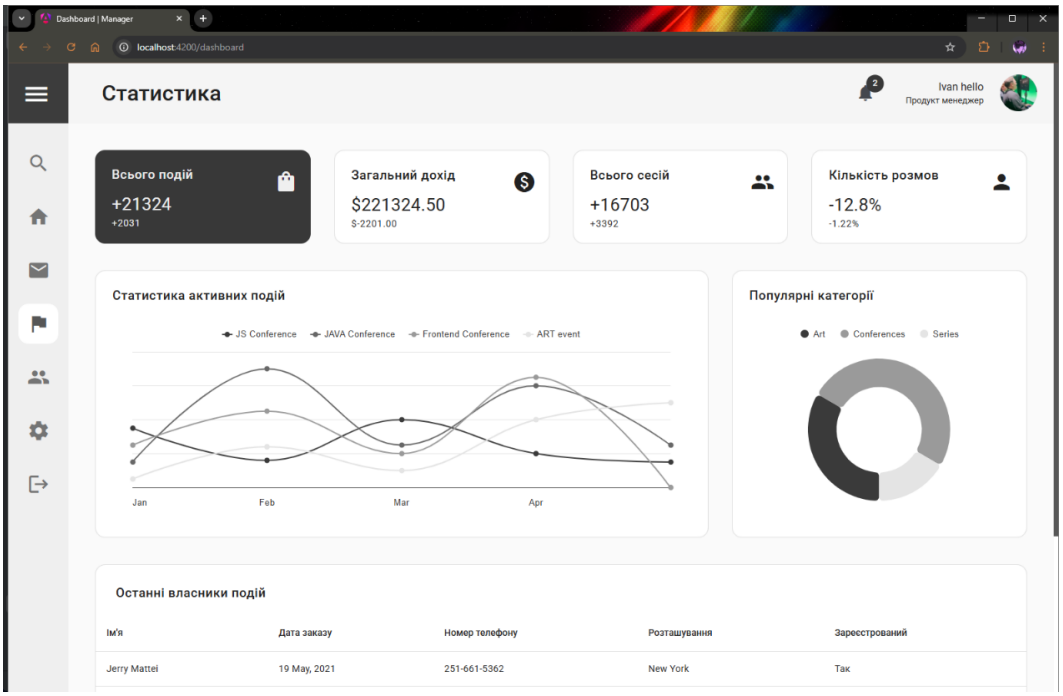

Рисунок 4.6 – Вигляд компоненту dashboard-charts

– компонент messenger-box відповідає за інтерфейс чату (зображено на рисунку 4.7);

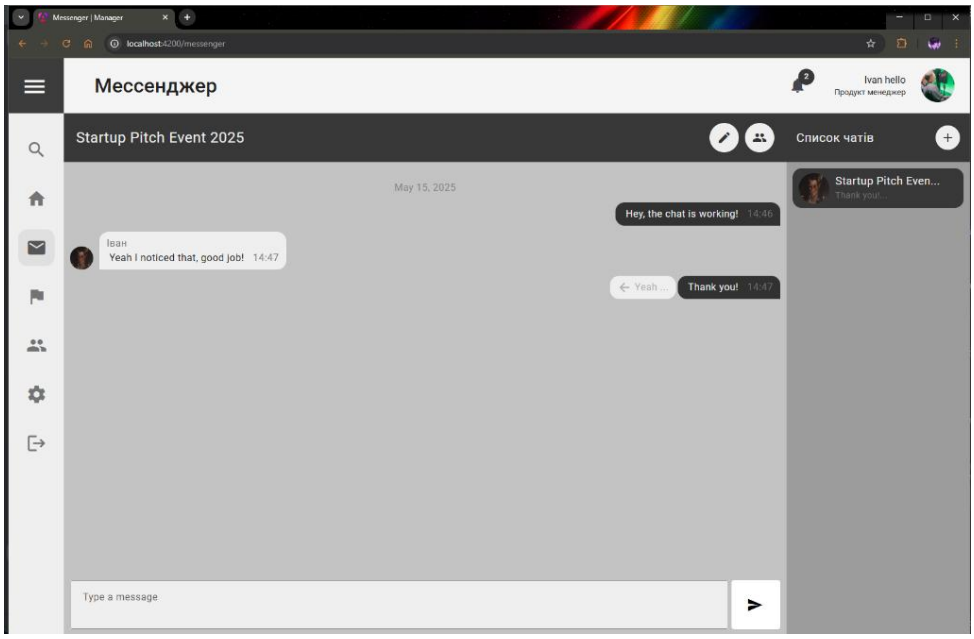

Рисунок 4.7 – Вигляд компоненту messenger-box

– сервіс messenger.service.ts керує запитами до серверу, зокрема створенням чатів, надсиланням повідомлень, редагуванням, видаленням та отриманням історії. Нижче наведено код отримання чатів з back-end.

```
getChats(): Observable<Chat[]> {
  return this.http.get<Chat[]>(`${environment.apiUrl}/inbox/user-
chats?userId=${this.getUserIdFromToken()}`);
}
```

Реалізовано підтримку групових чатів із можливістю додавання/видалення учасників (зображено на рисунку 4.8) та відповіді на конкретні повідомлення.

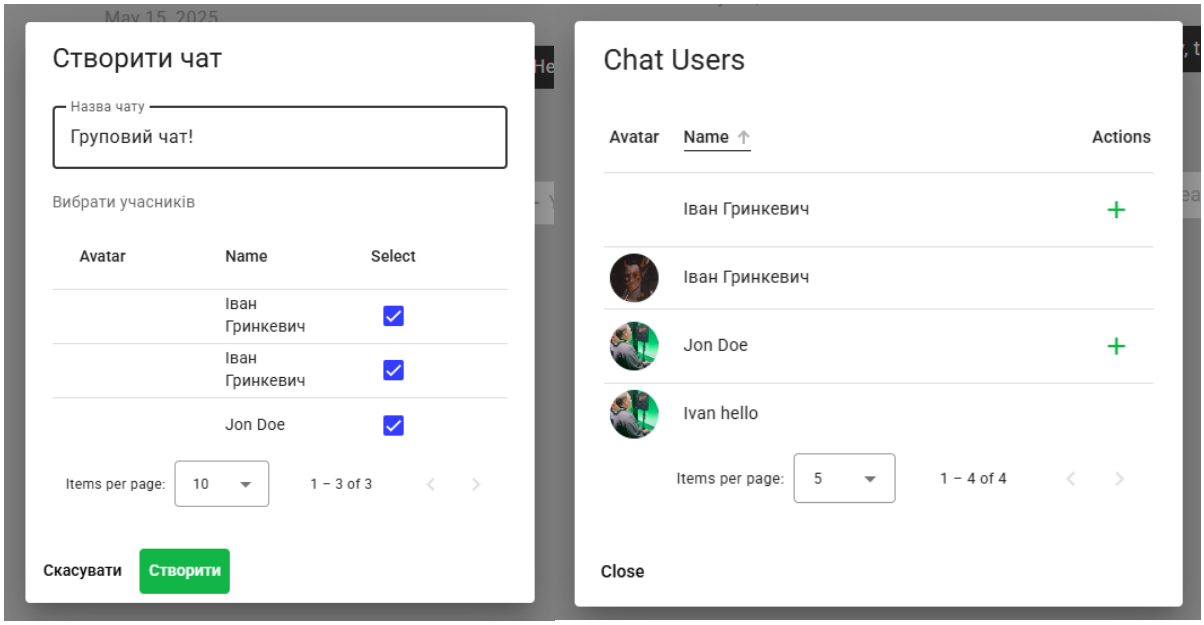

Рисунок 4.8 – Створення групового чату та редагування учасників

Модуль users є надзвичайно важливим, адже віннотрібен для управління користувачами (зображено на рисунку 4.9):

– компонент all-users дозволяє адміністраторам переглядати перелік користувачів;

– компонент weekly-visitors відображає активність користувачів за тиждень.

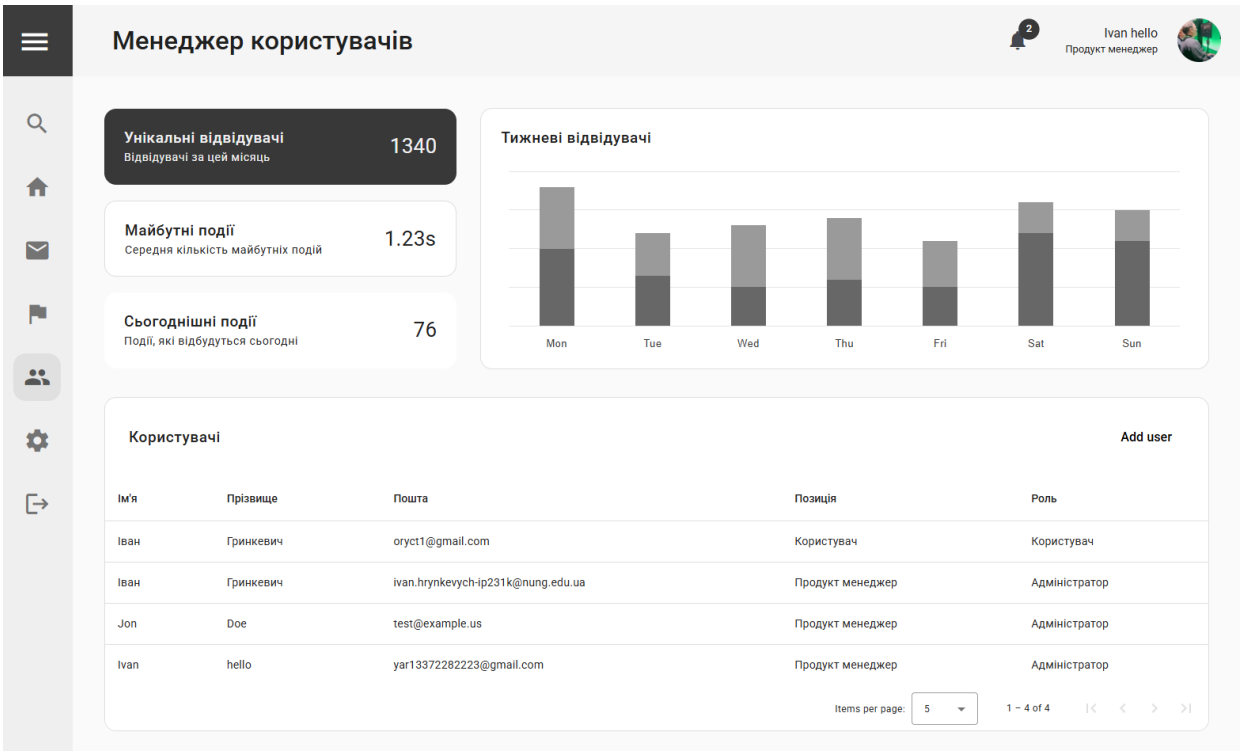

Рисунок 4.9 – Менеджер користувачів

Модулі header, sidenav, pipes, settings (зображено на рисунках 4.10 – 4.11) – необхідні для загального інтерфейсу користувача, фільтрації даних, зміни мови, теми та конфігурації застосунку.

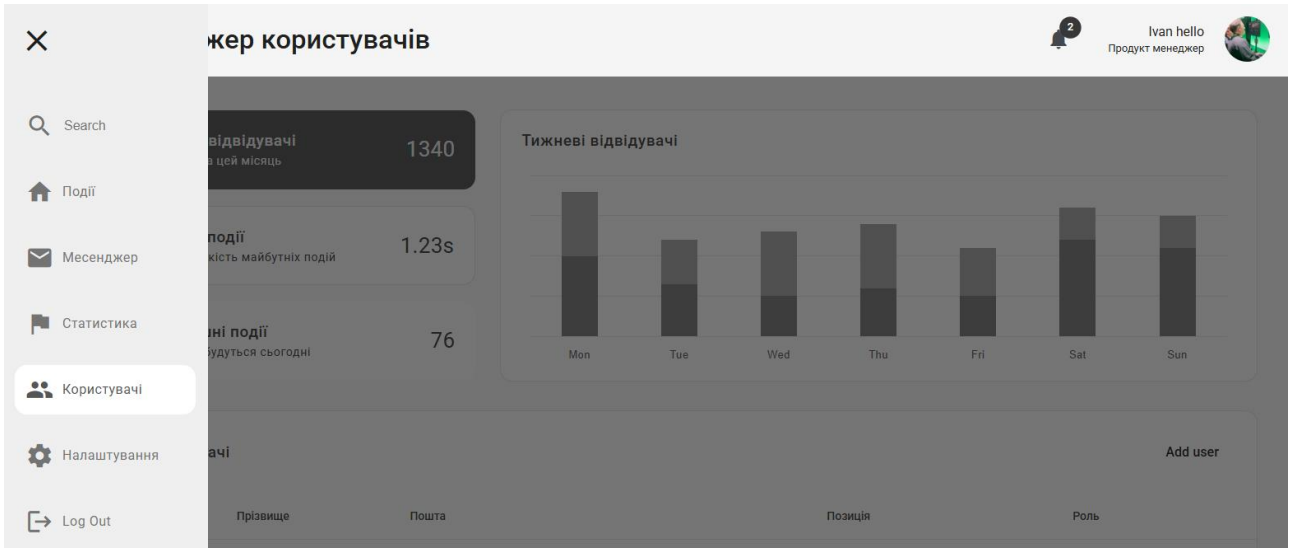

Рисунок 4.10 – Вигляд бокової панелі та заголовку

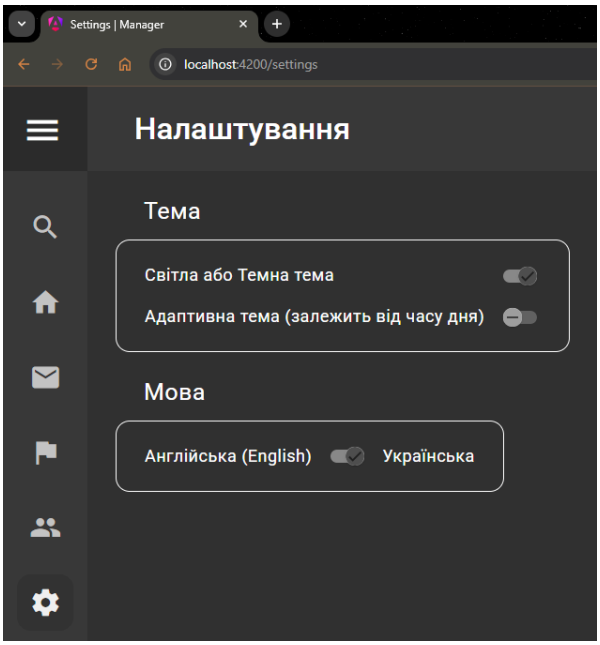

Рисунок 4.11 – Вікно налаштувань

4.1.2. Бекенд частина (Nest.js)

Серверна частина реалізована за допомогою Nest.js, що дозволяє ефективно організувати структуру проєкту (зображено на рисунку 4.12), використовуючи модулі, контролери та сервіси.

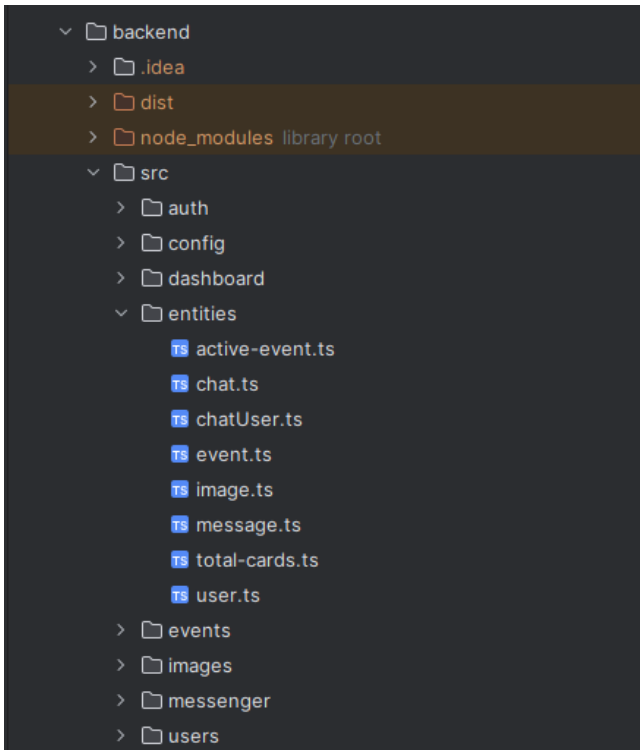

Рисунок 4.12– Структура Back-end частини застосунку

Модуль events відповідає за обробку логіки, пов’язаної з подіями:

– контролер events.controller.ts обробляє запити на створення, зміну, видалення та отримання подій;

Метод POST для створення події:

```
@UseGuards(RoleGuard)
@Post('create')
createEvent(@Body() createEventDto: CreateEventDto) {
  return this.eventsService.createEvent(createEventDto);
}
```

– DTO createEvent.dto.ts визначає формат вхідних даних.

```
export class CreateEventDto {
  @IsNotEmpty()
  @IsString()
  title: string;

  @IsNotEmpty()
  @IsString()
  description: string;

  @IsNotEmpty()
  startDate: Date;

  @IsNotEmpty()
  endDate: Date;
  ...
```

– сервіс events.service.ts реалізує основну бізнес-логіку подій (зображено на рисунку 4.13);

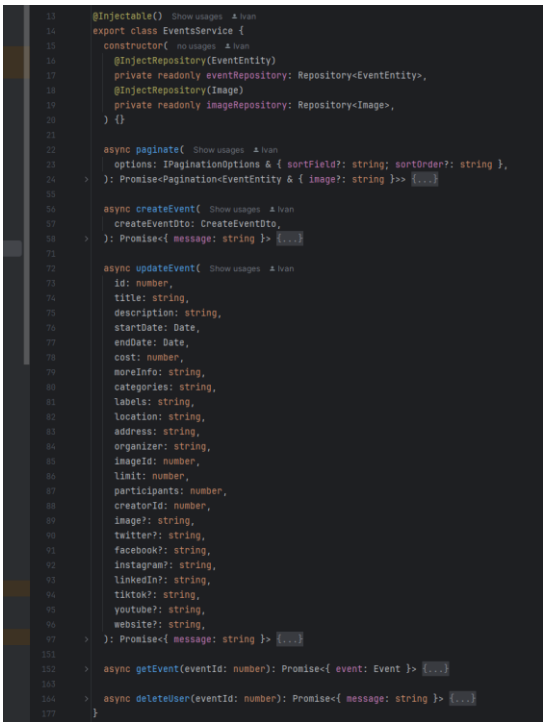

Рисунок 4.13 – Методи сервісу events.service.ts

Модуль messenger необхідний для обміну повідомленнями:

– контролер messenger.controller.ts обробляє маршрути для створення чатів та повідомлень. Нижче наведено код методу відправки повідомлень в messenger.controller.ts:

```
@Post('create-message')
sendMessage(
  @Body()
  body: {
    userId: number;
    chatId: number;
    content: string;
    replyToId: number;
  },
) {
  console.log('replyToId: ', body.replyToId);
  return this.messengerService.sendMessage(
    body.userId,
    body.chatId,
    body.content,
    body.replyToId,
  );
}
```

– сервіс messenger.service.ts відповідає за доступ до БД, зокрема використовуючи репозиторії для роботи з ентиті chat, message, chatUser. Нижче наведено код методу відправки повідомлень в messenger.service.ts:

```
async sendMessage(
    userId: number,
    chatId: number,
    content: string,
    replyToId: number,
  ): Promise<Message> {
    const chatUser = await this.chatUserRepository.findOne({
      where: { userId: userId, chatId: chatId },
    });

    if (!chatUser) {
      throw new Error('User is not a member of the chat');
    }

    // Mark the chat as new for all users except the sender
    const chatUsers = await this.chatUserRepository.find({
      where: { chatId: chatId },
    });

    for (const chatUser of chatUsers) {
      if (chatUser.userId !== userId) {
        chatUser.new = true;
        await this.chatUserRepository.save(chatUser);
      }
    }

    return await this.messageRepository.save({
      name: 'message',
      chatId,
      content,
```

```
      senderId: userId,
      createdAt: new Date(),
      replyToId,
      updatedAt: new Date(),
    });
  }
```

Модуль users відповідає за управління користувачами:

– контролер users.controller.ts обробляє запити на створення, оновлення профілів та отримання інформації про користувачів;

– DTO createUser.dto.ts визначає структуру реєстраційних даних;

– сервіс users.service.ts забезпечує бізнес-логіку користувацької частини.

Нижче наведено код методу створення користувача в users.controller.ts та users.service.ts:

```
@Post('create')
createUser(@Body() createUserDto: CreateUserDto) {
  return this.userService.createUser(createUserDto);
}

async createUser(
    createUserDto: CreateUserDto,
  ): Promise<{ access_token: string }> {
    const userExists: UserEntity = await this.userRepository.findOne({
      where: { email: createUserDto.email },
    });
    if (userExists) {
      throw new UnauthorizedException('User already exists');
    }

    if (createUserDto.profileImage !== '') {
      const newImage = this.imageRepository.create({
        image: createUserDto.profileImage,
      });
      const savedImage = await this.imageRepository.save(newImage);
      createUserDto.profileImageId = savedImage.id;
    }

    const newUser = this.userRepository.create(createUserDto);
    const payloadUser = await this.userRepository.save(newUser);
    const payload = {
      sub: payloadUser.id,
      email: payloadUser.email,
      firstname: payloadUser.firstname,
      lastname: payloadUser.lastname,
      position: payloadUser.position,
      role: payloadUser.role,
      profileImageId: payloadUser.profileImageId,
    };
    return {
      access_token: this.jwtService.sign(payload),
    };
  }
```

– active-events та total-cards містять функціонал для збору й обробки статистичних даних, які відображаються у фронтенді через графіки;

Модуль images необхідний для завантаження зображень подій та користувачів:

– реалізовано можливість додавання фото до подій та користувачів через images.controller.ts та images.service.ts. Нижче наведено код методу отримання повідомлення в images.controller.ts та images.service.ts:

```
@Get('get')
  getImage(@Query() getImageDto: { id: number }) {
    return this.imagesService.getImage(getImageDto.id);
  }


async getImage(imageId: number): Promise<{ image: string }> {
    const imageExists = await this.imageRepository.findOne({
      where: { id: imageId },
    });

    if (!imageExists) {
      throw new Error(`Image not found: ${imageId}`);
    } else {
      return { image: imageExists.image };
    }
  }
```

4.2 Використані алгоритми та структури даних

У розробці програмного забезпечення системи «EventManager» було застосовано низку структур даних та алгоритмів, які забезпечують ефективну обробку, зберігання та взаємодію між користувачами, подіями та повідомленнями. Основна мета використання відповідних підходів – підвищити швидкодію системи, забезпечити коректну логіку бізнес-процесів та надати зручний і стабільний інтерфейс для користувачів.

4.2.1. Об’єктно-реляційне відображення (ORM)

Для взаємодії з базою даних використовується TypeORM, який дозволяє описувати структуру таблиць через Entity-класи. Кожна сутність (entity) у проєкті відповідає певному об’єкту предметної області:

– user: описує користувача системи;

– event: описує подію з усіма мета-даними;

– message, Chat, ChatUser: описують об'єкти месенджера;

– image: описує зображення, прикріплені до подій;

– totalCards, ActiveEvent: для побудови різноманітних статистик.

Ці класи фактично є реалізацією структур даних, які мають поля типу string, number, Date, boolean, а також зв’язки OneToMany, ManyToOne, ManyToMany, що реалізують асоціації між таблицями.

4.2.2. DTO (Data Transfer Objects)

DTO об'єкти (createUser.dto.ts, createEvent.dto.ts тощо) застосовуються для передачі строго типізованих даних між клієнтом і сервером. Це дозволяє:

– захищати сервер від надлишкових або некоректних даних;

– перевіряти вхідні значення через валідаційні декоратори (@IsString, @IsEmail, @IsNotEmpty тощо).

Код об’єкта createUser.dto.ts:

```
import { IsEmail, IsNotEmpty, IsString, MinLength } from 'class-validator';

export class CreateUserDto {
  @IsString()
  @IsNotEmpty()
  @IsEmail()
  email: string;

  @MinLength(8)
  password: string;

  @IsString()
  @IsNotEmpty()
  firstname: string;

  @IsString()
  lastname: string;

  @IsString()
  position: string;

  @IsString()
  role: string;

  @IsString()
  profileImageId?: number;
}
```

4.2.3. Пагінація

Для обмеження обсягу даних, що завантажуються з сервера, реалізовано пагінацію (посторінковий вивід) для подій та повідомлень. Алгоритм базується на параметрах offset і limit, які передаються у запитах клієнта.

4.2.4. Обробка повідомлень у чаті

Кожне повідомлення проходить такі етапи обробки:

– збереження в базі даних.

– актуалізація повідомлень для всіх користувачів

Код методу редагування повідомлень в messenger.controller.ts:

```
@Post('update-message')
updateMessage(@Body() body: { messageId: number; content: string }) {
  return this.messengerService.updateMessage(body.messageId, body.content);
}
```

Код методу редагування повідомлень в messenger.service.ts:

```
async updateMessage(messageId: number, content: string): Promise<Message> {
    const message = await this.messageRepository.findOne({
      where: { id: messageId }, });
    message.content = content;
    message.updatedAt = new Date();
    return this.messageRepository.save(message);
  }
```

4.2.5. Реактивні форми

Angular Reactive Forms використовуються для створення динамічних та перевірених форм, наприклад, при реєстрації, створенні подій, надсиланні повідомлень. Валідація вводиться як на рівні компонентів, так і через власні валідатори (наприклад, перевірка пароля чи email).

Код ініціалізації форми груп loginForm для валідації даних при авторизації:

```
      loginForm = new FormGroup({
          email: new FormControl('', [Validators.required,
Validators.email]),
          password: new FormControl('', [Validators.required,
Validators.minLength(8)]),
        });
```

4.2.6. Сортування, фільтрація та пошук подій

На стороні клієнта реалізовано швидкий пошук та фільтрацію подій по заголовку або категорії (зображено на рисунку 4.14).

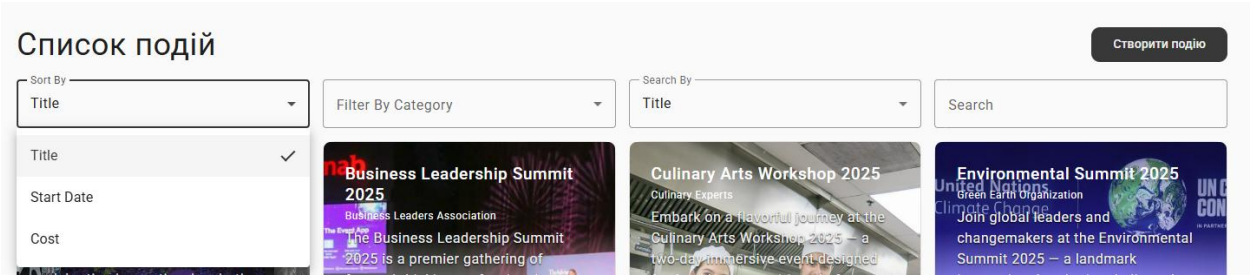

Рисунок 4.14 – Сортування, фільтрація та пошук подій

4.3 Модульне тестування та налагодження

Якість програмного забезпечення напряму залежить від надійності та передбачуваності його поведінки у різних ситуаціях. Саме тому при реалізації проєкту «EventManager» було приділено значну увагу модульному тестуванню (unit testing) та процесу налагодження (debugging) як на стороні бекенду, так і фронтенду.

Основна увага зосереджувалась на тестуванні сервісних класів, де реалізована логіка обробки даних. Наприклад:

- eventsService – тестування методів створення, оновлення, видалення подій;
- usersService – перевірка логіки створення та пошуку користувачів;
- messengerService – тестування обробки повідомлень і створення чатів;

Код тесту на метод створення події (events.service.spec.ts):

```
describe('EventsService', () => {
  let service: EventsService;
  let repo: Repository<Event>;

  beforeEach(async () => {
    const module = await Test.createTestingModule({
      providers: [
        EventsService,
        {
          provide: getRepositoryToken(Event),
          useClass: Repository,
        },
      ], }).compile();

    service = module.get<EventsService>(EventsService);
    repo = module.get<Repository<Event>>(getRepositoryToken(Event));
  });

  it('should create a new event', async () => {
    const mockEvent = { title: 'Test Event', date: new Date(), ... };
    jest.spyOn(repo, 'save').mockResolvedValue(mockEvent as any);
    expect(await service.create(mockEvent)).toEqual(mockEvent);
  });
});
```

Для перевірки коректної обробки HTTP-запитів проводились тести контролерів (events.controller.spec.ts, users.controller.spec.ts) з моканими залежностями сервісів.

У фронтенд-частині тестування проводилось за допомогою Karma та

Jasmine, які постачаються з Angular CLI.

Було протестовано базову функціональність компонентів:

- events.component – перевірка завантаження списку подій та відображення у шаблоні;
- messenger.component – тестування обробки повідомлень;
- dashboard.component – перевірка підвантаження аналітичних даних.

Код тесту на компонент events.component.spec.ts:

```
        it('should fetch events on init', () => {
          const mockEvents = [{ id: 1, title: 'Event' }];
          eventService.getAllEvents =
jasmine.createSpy().and.returnValue(of(mockEvents));
          fixture.detectChanges();
          expect(component.events.length).toBe(1); });
```

4.4 Висновки по розділу

У даному розділі було детально розглянуто етапи реалізації інформаційної системи «EventManager», що охоплює опис структури програмного коду, використані алгоритми, структури даних, а також підходи до модульного тестування та налагодження.

На етапі реалізації було реалізовано чіткий поділ системи на фронтенд (Angular) та бекенд (Nest.js), що забезпечило високу модульність та масштабованість. Для клієнтської частини реалізовано інтерфейси для взаємодії з подіями, повідомленнями, користувачами та налаштуваннями, а також механізми авторизації, маршрутизації та локалізації.

Особливу увагу приділено реалізації ефективних алгоритмів фільтрації, пагінації, обробки чатів у реальному часі та побудови аналітичної інформації. Для зберігання даних використано реляційну модель з урахуванням зв’язків між сутностями (подія, користувач, повідомлення, чат).

Модульне тестування дало змогу перевірити ключову бізнес-логіку системи в ізоляції. Налагодження, логування та тестування REST-запитів через Postman забезпечили виявлення помилок на ранніх етапах.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ

5.1 Стратегії та методи тестування

У процесі створення програмного забезпечення важливим етапом є тестування, яке дозволяє виявити дефекти, перевірити відповідність функціоналу вимогам, а також забезпечити стабільність, продуктивність та безпеку системи. Тестування – це систематичний процес виявлення помилок, який передбачає перевірку як функціональних, так і нефункціональних аспектів роботи системи.

Для тестування застосунку «EventManager» було розроблено тест-плани, які охоплюють як функціональні сценарії, так і нефункціональні аспекти. Тестові плани наведені в таблиці 5.1.

Таблиця 5.1

Функціональні та нефункціональні тест-плани

№	Функціональність	Тестові дії	Очікуваний результат
1	Авторизація	Ввести правильні email і пароль	Користувач успішно входить
2	Авторизація	Ввести неправильний пароль	Показано повідомлення про помилку
3	Перегляд подій	Натиснути на меню “Події”	Завантажується список подій
4	Перегляд конкретної події	Натискання на подію	Відкривається сторінка з деталями події
5	Надсилання повідомлення в месенджері	Ввести текст і натиснути "Надіслати"	Повідомлення з’являється в чаті
6	Перемикання теми	Вибрати темну тему в налаштуваннях	Застосовується темна тема до всього інтерфейсу
7	Зміна мови	Обрати іншу мову в налаштуваннях	Увесь інтерфейс перекладається
8	Перегляд статистики	Увійти в адмін-панель	Виводиться статистика користувачів і подій
9	Рольова перевірка	Спробувати доступ до адмін-панелі як звичайний користувач	Доступ заборонено

Продовження таблиці 5.1

№	Категорія	Тестові дії	Очікуваний результат
1	Продуктивність	Відкрити сторінку подій при 100+ подіях	Сторінка завантажується < 2 секунд
2	Юзабіліті	Навігація між сторінками	Зрозумілий і логічний інтерфейс
3	Сумісність	Відкрити застосунок у Chrome, Firefox, Safari	Інтерфейс однаковий у всіх браузерах
4	Безпека	Ввести SQL-ін’єкцію в полі логіну	Система блокує запит, немає витоку
5	Доступність	Використання клавіатурної навігації	Усі елементи доступні з клавіатури

5.2 Результати тестування системи

У процесі тестування системи було виявлено низку дефектів, які впливали на зручність користування, функціональність та безпеку додатку. Всі знайдені помилки були задокументовані, проаналізовані та отримали пропозиції щодо їх усунення, що відображено в таблиці 5.2.

Таблиця 5.2

Результати тестування

№	Сторінка	Опис дефекту	Критичність	Пропозиція щодо виправлення
1	Авторизація	Немає валідації email	Середня	Додати перевірку формату email
2	Сторінка події	Довгі назви виходять за межі блоку	Низька	Додати обмеження тексту або перенос
3	Месенджер	Повідомлення інколи дублюються	Висока	Перевірити логіку дублювання та id
4	Налаштування	Перемикання теми не зберігається при перезавантаженні	Середня	Використовувати локальне збереження (localStorage)
5	Адмін-панель	Звичайний користувач має доступ до адмін-панелі	Висока	Додати перевірку ролі користувача

Відсутність валідації email в авторизації могла призводити до введення некоректних або шкідливих даних, що не лише погіршувало користувацький досвід, але й створювало потенційні вразливості для системи (наприклад, ін’єкції). Впровадження подвійної валідації (клієнт та сервер) суттєво підвищило якість обробки даних.

Візуальні проблеми з довжиною заголовків подій впливали на зовнішній вигляд інтерфейсу, знижували його естетичність і могли ускладнювати сприйняття інформації. Впровадження обмежень та адаптивного форматування забезпечило акуратний і приємний вигляд UI.

Дублювання повідомлень у месенджері негативно позначалося на комунікації між користувачами, створювало плутанину і навантаження на мережу. Усунення проблеми потребувало вдосконалення механізму генерації унікальних ідентифікаторів та обробки подій на клієнті й сервері.

Відсутність збереження налаштувань теми спричиняла незручності, особливо для користувачів, які віддають перевагу темному інтерфейсу у вечірній час. Збереження стану теми у браузері гарантувало комфортне користування навіть після оновлення сторінки.

Недостатній контроль доступу до адмін-панелі створював загрозу безпеці даних, оскільки неавторизовані користувачі могли отримати привілеї адміністратора. Впровадження багаторівневого контролю та серверної перевірки ролей підвищило загальну безпеку системи.

Додаткові результати тестування:

- UI/UX-тестування показало необхідність покращення логіки перемикання між розділами: було реалізовано фіксовану бічну навігацію для спрощення доступу до основних функцій;
- перевірка кросбраузерної сумісності засвідчила коректну роботу застосунку у сучасних браузерах (Chrome, Firefox, Edge). Для Safari було внесено незначні стилістичні коригування;

– аналіз продуктивності показав, що при використанні пагінації та lazy loading сторінка подій завантажується менш ніж за 1.5 секунди навіть при 25+ подіях, що відповідає вимогам до швидкодії;

– безпекове тестування підтвердило наявність захисту від SQL-ін’єкцій та міжсайтових скриптів, а також продемонструвало необхідність посилення перевірки ролей – після чого було реалізовано серверну перевірку прав доступу.

Після проведення виправлень критичних та середніх дефектів, повторне тестування підтвердило відповідність системи вимогам, стабільність та безпеку, а також готовність до впровадження в виробниче середовище. Результати тестування стали основою для розробки плану подальшого супроводу та розвитку програмного продукту.

5.3 Аналіз ефективності та вдосконалення

Після завершення основних етапів розробки та тестування системи було проведено комплексний аналіз ефективності її впровадження, а також визначено напрямки для подальшого вдосконалення. Цей аналіз включає як технічні показники, так і якість користувацького досвіду, що дозволяє комплексно оцінити успішність реалізації проєкту.

5.3.1 Оцінка продуктивності системи

Під час тестування продуктивності було виявлено, що завдяки застосуванню сучасних підходів, таких як пагінація, lazy loading та кешування, сторінки з великим обсягом інформації (наприклад, список подій) завантажуються швидко – у середньому менше ніж за 1.5 секунди. Цей показник відповідає сучасним стандартам веб-застосунків і забезпечує комфортну роботу користувачів навіть при збільшенні кількості подій і користувачів.

5.3.2 Безпекова ефективність

Впроваджені механізми авторизації, ролі доступу та захист від основних типів атак (SQL-ін’єкції, XSS) забезпечили високий рівень безпеки системи. Проведені пентести показали відсутність критичних вразливостей. Водночас,

аналіз показав необхідність регулярного оновлення залежностей та впровадження додаткових заходів для моніторингу підозрілої активності в реальному часі.

5.3.3 Напрямки для подальшого вдосконалення

На основі отриманих результатів визначено основні напрямки подальшої роботи над системою:

- розширення функціоналу месенджера – додавання можливості відправлення медіа-файлів, голосових повідомлень та покращення пошуку повідомлень;
- інтеграція з платіжними системами – для автоматизації оплати платних подій та спрощення процесу покупки квитків;
- вдосконалення системи сповіщень – впровадження push-сповіщень та SMS-повідомлень для підвищення оперативності комунікації;
- автоматизація звітності та аналітики – розширення можливостей адміністративної панелі для формування детальних звітів і прогнозі;
- підвищення доступності – забезпечення підтримки стандартів доступності для користувачів з інвалідністю.

Для забезпечення стабільної роботи системи після впровадження було розроблено план регулярного супроводу, який включає:

- моніторинг продуктивності та безпеки;
- регулярне оновлення програмних компонентів і бібліотек;
- виправлення виявлених дефектів та багів;
- впровадження нових функціональних можливостей відповідно до потреб користувачів.
- Організацію служби підтримки користувачів.

Загальна оцінка ефективності впровадження є позитивною. Система відповідає поставленим вимогам, демонструє високу продуктивність, безпеку і зручність для кінцевих користувачів. Визначені напрямки вдосконалення стануть основою для подальшого розвитку проєкту, що забезпечить його конкурентоспроможність і відповідність сучасним вимогам.

5.4 Висновки по розділу

У цьому розділі було розглянуто процес тестування програмного застосунку «EventManager», що є важливою складовою забезпечення його якості та надійності. Завдяки комплексному підходу до тестування, який включав функціональні та нефункціональні аспекти, вдалося виявити низку помилок і недоліків, що впливали на стабільність роботи, зручність використання та безпеку системи.

Результати тестування дали змогу чітко окреслити пріоритетні напрямки для удосконалення, серед яких збереження користувацьких налаштувань, оптимізація інтерфейсу, забезпечення доступності та посилення заходів безпеки. Вжиті заходи суттєво покращили загальну якість продукту, що підтверджується як позитивною реакцією користувачів, так і зменшенням кількості виявлених дефектів під час подальших перевірок.

Реалізоване тестування і впроваджені вдосконалення стали необхідною базою для успішного подальшого впровадження системи «EventManager» у практику, гарантують стабільну роботу та високий рівень задоволення користувачів.

Крім того, було визначено план підтримки та оновлень в майбутньому, що є надзвичайно важливим для проєкту, який має довгострокові плани для підтримки.

ВИСНОВКИ

У даній бакалаврській роботі було представлено процес розробки веб-застосунку «EventManager» для управління подіями з розширеними можливостями. Для розробки було обрано сучасні та ефективні технології – Angular для фронтенду та Nest.js для бекенду, що забезпечують масштабованість, зручність та безпеку системи.

Робота виконувалась у рамках навчання в університеті, з урахуванням знань, отриманих за 2 роки, а також з використанням результатів попередніх курсових робіт. Було проведено детальний аналіз предметної області, визначено типи користувачів системи (неавторизований користувач, авторизований користувач, адміністратор), а також функціональні та нефункціональні вимоги, що враховують реальні потреби в управлінні подіями.

Особливу увагу приділено моделюванню архітектури системи, розробці бізнес-процесів і компонентів, а також вибору технологій і інструментів, які дозволили реалізувати багатофункціональний месенджер, гнучку систему авторизації та можливості оплати платних подій.

Для забезпечення стабільності та надійності системи проведено детальне тестування, яке охоплювало перевірку відповідності функціональним і нефункціональним вимогам, а також аналіз коду відповідно до стандартів «чистого коду». Розроблено план рефакторингу, що дозволяє підтримувати високу якість програмного продукту у майбутньому.

Отже, веб-застосунок «EventManager» є цілісним програмним рішенням, яке можна застосовувати для ефективного управління подіями та комунікації користувачів у реальному часі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. В. Я. Піх, М. М. Піх, В.В. Бандура. Методичні рекомендації та вимоги щодо виконання та оформлення бакалаврських робіт. Івано-Франківськ, 67 с. (Інформація та документація).
2. ДСТУ 8302:2015. Бібліографічне посилання. Загальні положення та правила складання. Вид. офіц. Київ, 2016. 20 с. (Інформація та документація).
3. ДСТУ ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. Вид. офіц. Київ, 2016. 31 с. (Інформація та документація).
4. Офіційна документація Angular [Електронний ресурс]. – URL: https://angular.io/
5. Офіційна документація NestJS [Електронний ресурс]. – URL: https://docs.nestjs.com/
6. Офіційна документація Angular Material [Електронний ресурс]. – URL: https://material.angular.io/
7. Офіційна документація Microsoft SQL Server [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/sql/sql-server/
8. Платформа для організації командної роботи Trello [Електронний ресурс]. – URL: https://trello.com/
9. Освітня платформа Udemy [Електронний ресурс]. – URL: https://www.udemy.com/
10. Платформа з великою кількістю різноманітних курсів [Електронний ресурс]. – URL: https://www.w3schools.com/
11. Офіційна документація TypeScript [Електронний ресурс]. – URL: https://www.typescriptlang.org/
12. Офіційна документація Node.js [Електронний ресурс]. – URL: https://nodejs.org/
13. Офіційна документація IntelliJ IDEA [Електронний ресурс]. – URL: https://www.jetbrains.com/idea/
14. Платформа розробки мобільних та веб-застосунків Firebase

[Електронний ресурс]. – URL: https://firebase.google.com/

15. Інструмент для контролю версій Git [Електронний ресурс]. – URL: https://git-scm.com/

16. Онлайн-курс “The Complete Guide to Angular” [Електронний ресурс]. – URL: https://www.udemy.com/course/the-complete-guide-to-angular-2/

17. Офіційна документація RxJS [Електронний ресурс]. – URL: https://rxjs.dev/

18. Офіційна документація Postman для тестування API [Електронний ресурс]. – URL: https://www.postman.com/

19. Офіційна документація GitHub [Електронний ресурс]. – URL: https://docs.github.com/

20. Онлайн-курс “RESTful API Design” [Електронний ресурс]. – URL: https://www.udemy.com/course/restful-api-design-with-nodejs/

21. Офіційна документація JSON Web Token (JWT) [Електронний ресурс]. – URL: https://jwt.io/introduction

22. Офіційна документація Docker для розгортання застосунків [Електронний ресурс]. – URL: https://docs.docker.com/

23. Документація про REST-архітектуру [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

24. Офіційна документація ESLint – інструменту для аналізу коду [Електронний ресурс]. – URL: https://eslint.org/docs/latest/

25. Офіційна документація Prettier для форматування коду [Електронний ресурс]. – URL: https://prettier.io/

26. Офіційна документація JWT Authentication у NestJS [Електронний ресурс]. – URL: https://docs.nestjs.com/security/authentication

27. Офіційна документація Angular CLI [Електронний ресурс]. – URL: https://angular.io/cli

28. Офіційна документація бібліотеки для візуалізації даних Highcharts в Angular [Електронний ресурс]. – URL: https://www.highcharts.com/integrations/angular/

29. Офіційна документація Lighthouse для покращення якості веб-застосунків [Електронний ресурс]. – URL: https://developer.chrome.com/docs/lighthouse/overview

30. Офіційна документація Single-page application [Електронний ресурс]. – URL: https://learn.microsoft.com/en-us/entra/identity-platform/index-spa

31. Офіційна документація Angular Reactive Forms [Електронний ресурс]. – URL: https://angular.dev/guide/forms/reactive-forms

32. Офіційна документація Angular Internationalization [Електронний ресурс]. – URL: https://angular.dev/guide/i18n

33. Офіційна документація Angular Components [Електронний ресурс]. – URL: https://angular.dev/guide/components

34. Офіційна документація Angular Routing [Електронний ресурс]. – URL: https://v17.angular.io/guide/routing-overview

Форма бібліографічної довідки

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “Розробка ресурсу модерації та планування розважальних івентів”

Обсяг пояснювальної записки: 73 аркуша

Дата закінчення роботи 10 червня 2025р.

Підпис _____