Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційного веб-сайту на тематику огляду спортивних подій

Виконав: студент IV курсу, групи СН-43
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Федірко І.Б.
(підпис) (прізвище та ініціали)

Керівник _____ Марценко С.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«27» січня 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)
за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)
Студенту Федірко Ігорю Богдановичу
(прізвище, ім'я, по батькові)
1. Тема роботи Розробка інформаційного веб-сайту на тематику огляду спортивних подій

Керівник роботи Марценко Сергій Володимирович, к. т. н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом ректора від «*07*» *травня* 2025 року № *4/7-444*
2. Термін подання студентом завершеної роботи 27 червня 2025 р.
3. Вихідні дані до роботи Літературні та інтернет джерела, які застосовані для розробки інформаційного веб-сайту на тематику огляду спортивних подій

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та вибір технологій. 1.1. Аналіз існуючих веб-сайтів для огляду спортивних подій. 1.2. Веб-технології для створення сучасних інформаційних платформ. 1.3. Архітектурні підходи до розробки веб-сайтів. 1.4. Постановка завдання та обгрунтування вибору технологій. 1.5. Висновок до першого розділу. 2. Проєктування та архітектура веб-сайту. 2.1. Опис архітектури веб-сайту. 2.2. Проєктування фронтенд-частини веб-сайту. 2.3. Проєктування бекенд-частини веб-сайту. 2.4. Висновок до другого розділу. 3. Розробка та тестування веб-сайту. 3.1. Реалізація фронтенд-частини веб-сайту на основі React.js. 3.2. Реалізація бекенд-частини веб-сайту з використанням Node.js та Express.js. 3.3. Тестування веб-сайту. 3.4 Висновок до третього розділу. 4. Безпека життєдіяльності, основи охорони праці. Висновки. Перелік джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С., кандидат технічних наук, доцент кафедри МТ		

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	27.01.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	28.01.2025-06.02.2025	*Виконано*
3.	Виконання дослідження щодо розробки веб-сайту та	07.02.2025-30.03.2025	*Виконано*
	розроблення веб-сайту		
4.	Оформлення розділу «Аналіз предметної області та	03.06.2025-05.06.2025	*Виконано*
	вибір технологій»		
5.	Оформлення розділу «Проєктування та архітектура	06.06.2025-08.06.2025	*Виконано*
	веб-сайту»		
6.	Оформлення розділу «Розробка та тестування	09.06.2025-11.06.2025	*Виконано*
	веб-сайту»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	12.06.2025-14.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	12.06.2025-14.06.2025	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	16.06.2025-17.06.2025	*Виконано*
10.	Нормоконтроль	18.06.2025-19.06.2025	*Виконано*
11.	Перевірка на плагіат	20.06.2025	*Виконано*
12.	Попередній захист кваліфікаційної роботи	21.06.2025	*Виконано*
13.	Захист кваліфікаційної роботи	28.06.2025	*Виконано*

Студент _____ Федірко І.Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інформаційного веб-сайту на тематику огляду спортивних подій // Кваліфікаційна робота освітнього рівня «Бакалавр» // Федірко Ігор Богданович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-43 // Тернопіль, 2025 // С. 68, рис. – 14, додат. – 2, бібліогр. – 44.

Ключові слова: веб-сайт, спортивні події, react.js, node.js, express.js, фронтенд, бекенд, розробка

Кваліфікаційна робота присвячена дослідженню процесу створення веб-сайту для огляду спортивних подій.

У першому розділі кваліфікаційної роботи описано існуючі веб-сайти спортивної тематики, висвітлено особливості сучасних веб-технологій, розглянуто архітектурні підходи до розробки веб-застосунків, проаналізовано доцільність вибору технологій для реалізації проєкту.

У другому розділі кваліфікаційної роботи досліджено архітектуру сайту, подано структуру фронтенд- та бекенд-частин, обґрунтовано прийняті технічні рішення.

У третьому розділі кваліфікаційної роботи описано реалізацію клієнтської частини на основі React.js, серверної частини за допомогою Node.js і Express.js, проаналізовано результати тестування веб-сайту, проведено оцінку працездатності й зручності користування.

Об'єкт дослідження: процес створення інформаційного веб-ресурсу для огляду спортивних подій.

Предмет дослідження: методи, технології та засоби розробки веб-сайтів на основі JavaScript-стека.

ANNOTATION

Development of an Informational Website Reviewing Sports Event // Qualification work of the educational level "Bachelor" // Fedirko Ihor Bogdanovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-43 // Ternopil, 2025 // P. 68, fig. – 14, annexes. – 2, references – 44.

Keywords: website, sports events, react.js, node.js, express.js, frontend, backend, development

The qualification work is devoted to the study of the process of creating a website for reviewing sports events.

The first section of the qualification work describes existing sports-themed websites, highlights the features of modern web technologies, considers architectural approaches to developing web applications, and analyzes the feasibility of choosing technologies for implementing the project.

The second section of the qualification work examines the architecture of the site, presents the structure of the front-end and back-end parts, and justifies the technical decisions made.

The third section of the qualification work describes the implementation of the client part based on React.js, the server part using Node.js and Express.js, analyzes the results of website testing, and assesses its performance and usability.

Object of research: the process of creating an information web resource for reviewing sports events.

Subject of research: methods, technologies, and tools for developing websites based on the JavaScript stack.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API (Application Programming Interface) – інтерфейс прикладного програмування.

AWS (Amazon Web Services) – хмарна платформа від Amazon для надання різноманітних IT-сервісів.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

DOM (Document Object Model) – об'єктна модель документа.

HTTP (Hypertext Transfer Protocol) – протокол передачі гіпертексту.

JSON (JavaScript Object Notation) – формат обміну даними між сервером і клієнтом.

ORM (Object-Relational Mapping) – об'єктно-реляційне відображення.

REST (Representational State Transfer) – архітектурний стиль для створення веб-сервісів.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій суттєво впливає на всі сфери суспільного життя, зокрема й на галузь спортивної аналітики та огляду подій. Збільшення кількості спортивних змагань та зростання інтересу до них серед уболівальників створює попит на якісні та зручні веб-сайти для перегляду результатів матчів, аналітики, прогнозів та обговорення подій у реальному часі.

На сьогодні існує велика кількість веб-сайтів, присвячених огляду спортивних подій, проте більшість із них або мають застарілий інтерфейс, або не забезпечують достатньо швидке оновлення інформації. Користувачі прагнуть отримувати якісний та швидкий доступ до актуальних спортивних новин, аналітики, коментарів експертів і статистики матчів. Саме тому важливим є створення сучасного, інтерактивного веб-сайту, який надаватиме можливість оперативно отримувати оновлення щодо спортивних подій, переглядати відеоогляди та залишати коментарі [1].

Актуальність теми дипломної роботи зумовлена потребою у швидких, зручних та адаптивних веб-сайтах для огляду спортивних подій. Для досягнення цієї мети необхідно використати сучасні технології веб-розробки, які дозволять створити зручний та продуктивний веб-застосунок. Оптимальним вибором для реалізації фронтенд-частини є React.js, оскільки він забезпечує високу швидкість рендерингу компонентів, підтримку компонентного підходу та ефективну взаємодію з бекендом. Для реалізації серверної частини буде використано Node.js з фреймворком Express.js, що дозволить обробляти запити швидко та ефективно, а також забезпечити інтеграцію з базою даних.

Метою роботи є розробка інформаційного веб-сайту на тематику огляду спортивних подій, який надасть користувачам можливість переглядати результати матчів, читати аналітичні матеріали, обговорювати події та отримувати сповіщення про найважливіші оновлення [2].

Для досягнення мети необхідно виконати такі завдання:

1. Провести аналіз існуючих веб-сайтів для огляду спортивних подій та визначити їхні основні переваги та недоліки.

2. Дослідити сучасні веб-технології для створення інформаційних платформ та обґрунтувати вибір технологій для реалізації проєкту.

3. Розробити архітектуру веб-сайту, спроєктувати його фронтенд- та бекенд-частини.

4. Реалізувати фронтенд-частину веб-застосунку за допомогою React.js.

5. Реалізувати бекенд-частину на основі Node.js і Express.js, забезпечивши зберігання та обробку даних.

6. Виконати тестування веб-сайту, перевірити його продуктивність та зручність використання.

Об’єктом дослідження є процес розробки веб-сайтів для огляду спортивних подій.

Предметом дослідження виступають методи, підходи та технології веб-розробки для створення сучасних інформаційних платформ.

Результати дипломної роботи мають практичну цінність, оскільки розроблений веб-сайт може бути використаний для створення сучасних спортивних порталів та платформ. Запропоновані рішення можуть бути адаптовані для різних видів спорту, додані функції персоналізації контенту та інтеграція з зовнішніми API (наприклад, офіційними базами спортивних ліг).

Запропонована архітектура та технологічний стек дозволяють забезпечити високу продуктивність, масштабованість та гнучкість веб-застосунку, що є важливими критеріями для подібних проєктів.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБОРУ ТЕХНОЛОГІЙ

1.1 Аналіз існуючих веб-сайтів для огляду спортивних подій

Існуючі веб-сайти для огляду спортивних подій відіграють важливу роль у наданні користувачам актуальної інформації про спортивні змагання, команди, гравців, статистику та аналітику. Вони пропонують широкий спектр можливостей, серед яких публікація новин, онлайн-трансляції матчів, оновлення рахунків у реальному часі, детальна статистика, відеоогляди, коментарі експертів та форуми для обговорення подій [3].

FlashScore є одним із найпопулярніших сервісів для відстеження спортивних подій у режимі реального часу (див. рисунок 1.1). Він охоплює понад 30 видів спорту, включаючи футбол, баскетбол, теніс, хокей та багато інших дисциплін. Основна перевага платформи – швидкість оновлення результатів, що дозволяє користувачам стежити за ходом матчів у режимі реального часу. Крім рахунку, сервіс надає детальну статистику, включаючи кількість ударів, володіння м’ячем, кількість кутових ударів, склади команд і навіть коментарі про ключові моменти гри.

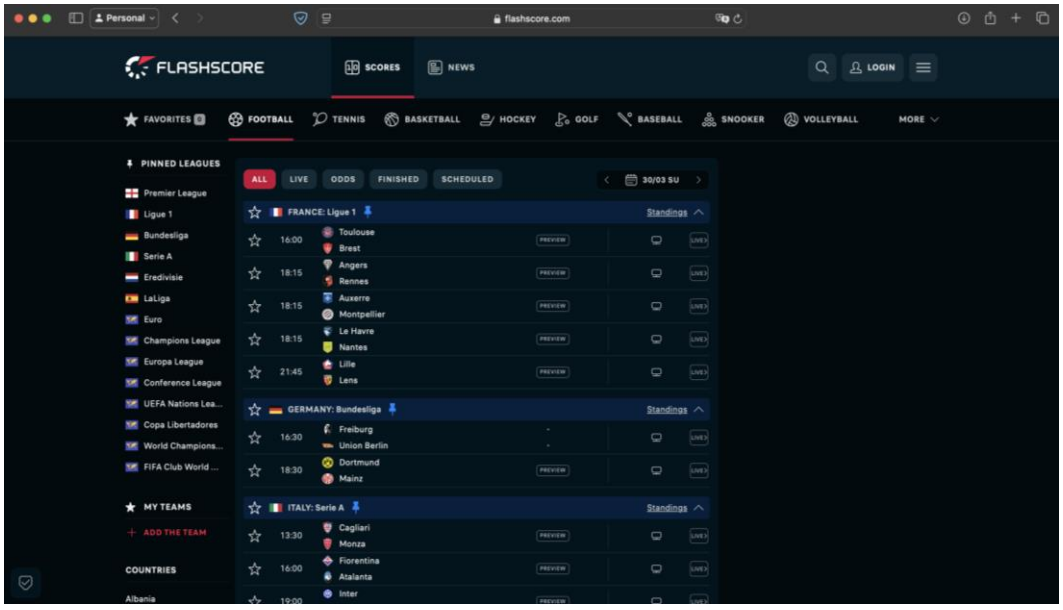

Рисунок 1.1 – Веб-сайт FlashScore

FlashScore також дозволяє персоналізувати досвід користувача, додаючи улюблені команди та турніри, щоб отримувати сповіщення про їхні матчі. Доступ до сервісу можливий як через веб-браузер, так і через мобільний додаток, що забезпечує зручність використання. Однак FlashScore зосереджується переважно на статистичних даних і не містить розширених аналітичних статей або відеоогляді матчів, що може бути недоліком для користувачів, які шукають більш глибокий аналіз спортивних подій [4].

ESPN є однією з найбільших платформ для висвітлення спортивних подій, яка пропонує не тільки результати матчів, але й детальні огляди, аналітику, ексклюзивні інтерв'ю та відеоконтент. Веб-сайт охоплює широкий спектр видів спорту, зокрема футбол, баскетбол, американський футбол, бейсбол, теніс, Формулу-1 та інші (див. рисунок 1.2).

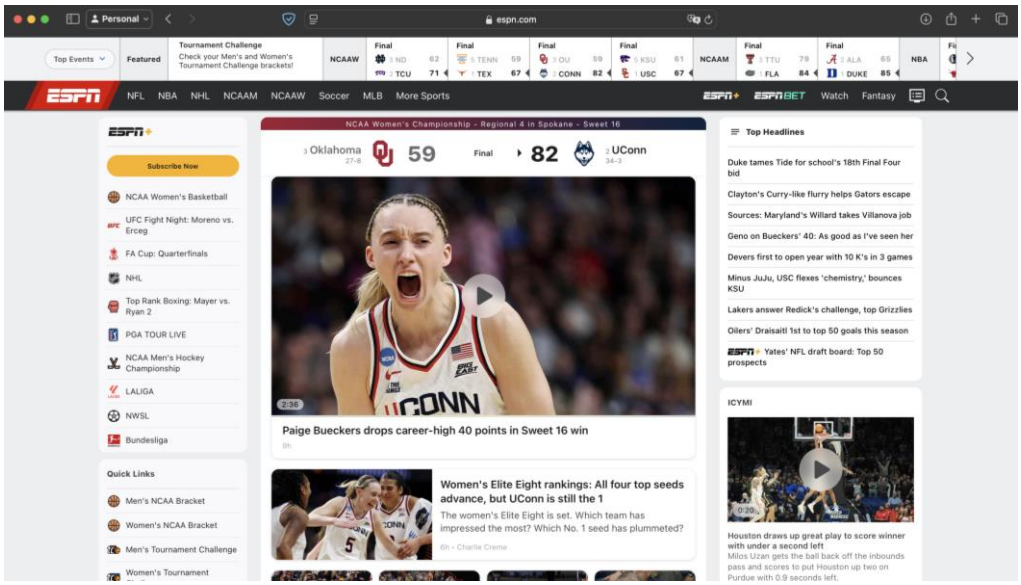

Рисунок 1.2 – Веб-сайт ESPN

Одна з головних переваг ESPN – це глибокий аналіз матчів, що включає розбір тактики, прогнозування результатів та коментарі експертів. Крім цього, сайт надає можливість переглядати відеоогляди та прямі трансляції матчів завдяки співпраці з офіційними спортивними лігами. Користувачі можуть налаштовувати персональні стрічки новин, обираючи улюблені команди або чемпіонати, щоб отримувати актуальну інформацію в першу чергу. Проте ESPN

має певні обмеження – частина контенту доступна лише за підпискою, а велика кількість реклами може впливати на зручність перегляду [5].

Goal.com – це популярний сайт, орієнтований переважно на футбольні події. Він висвітлює національні та міжнародні змагання, зокрема чемпіонати Європи, Лігу чемпіонів, національні першості та трансферні новини. Основна особливість Goal.com – швидке оновлення новин, зокрема чуток про трансфери та ексклюзивних коментарів від футбольних експертів (див. рисунок 1.3). Сайт також містить аналітичні статті, статистику матчів, рейтинги гравців та коментарі тренерів.

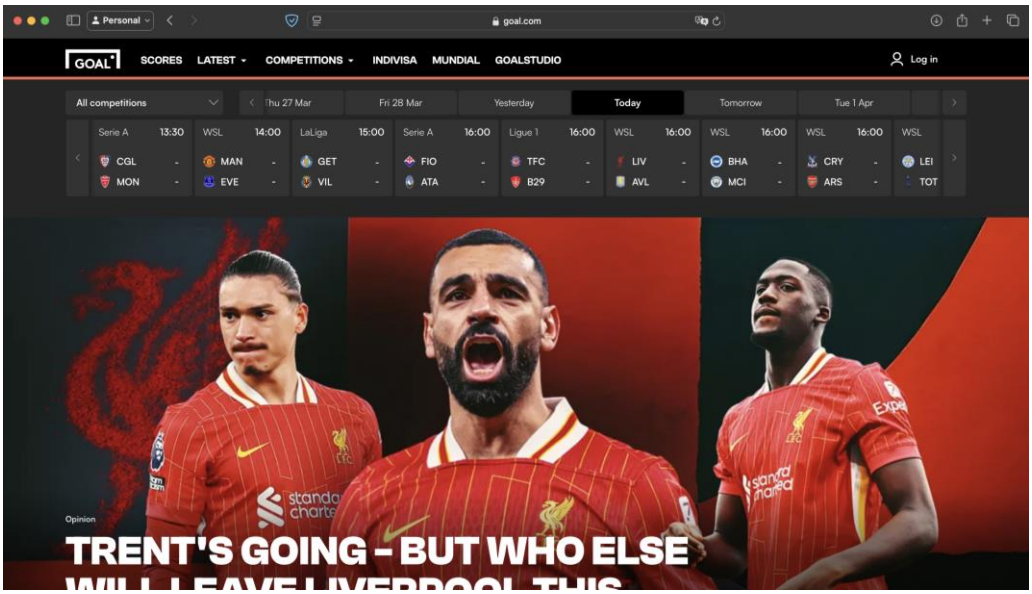

Рисунок 1.3 – Веб-сайт Goal.com

Велика увага приділяється візуальному оформленню: яскраві заголовки, інтерактивні блоки та відеоконтент роблять платформу привабливою для користувачів. Крім основної версії, Goal.com має локалізовані сайти для різних країн, що дозволяє адаптувати контент до конкретної аудиторії. Однак головний недолік полягає в тому, що сайт орієнтований майже виключно на футбол, тому любителі інших видів спорту можуть не знайти для себе достатньої кількості інформації [6].

Аналіз цих платформ дозволяє зробити висновок, що ефективний веб-сайт для огляду спортивних подій має включати швидке оновлення інформації,

персоналізований контент, аналітичні матеріали та мультимедійні елементи. Важливим є також зручний інтерфейс та адаптація для мобільних пристроїв. Враховуючи ці аспекти, при розробці власного веб-сайту слід зробити акцент на поєднанні аналітичних статей, інтерактивної статистики та відеоконтенту, що забезпечить користувачам повноцінний досвід взаємодії з платформою [7].

1.2 Веб-технології для створення сучасних інформаційних платформ

Сучасні інформаційні платформи потребують високої продуктивності, масштабованості та зручності в користуванні. Для їхньої розробки використовується широкий спектр веб-технологій, які охоплюють фронтенд, бекенд, бази даних, засоби кешування та хостинг-рішення.

Фронтенд-технології відіграють ключову роль у створенні інтерфейсу користувача та забезпеченні інтерактивності. Однією з найпопулярніших бібліотек для розробки фронтенду є React.js, який дозволяє створювати компонентний інтерфейс із можливістю повторного використання окремих елементів. Використання віртуального DOM значно підвищує продуктивність застосунку, а бібліотека React Router спрощує роботу з маршрутизацією. Альтернативами є Vue.js та Angular, які також підтримують розробку односторінкових застосунків. Для стилізації застосовуються CSS-препроцесори (Sass, LESS), CSS-фреймворки (Bootstrap, Tailwind CSS) або кастомні рішення на основі Styled Components чи Emotion.

Бекенд-технології відповідають за обробку даних, авторизацію користувачів, бізнес-логіку та взаємодію з базами даних. Одним із популярних виборів є Node.js – серверне середовище, яке дозволяє виконувати JavaScript-код на сервері. У поєднанні з Express.js воно надає легкий і швидкий фреймворк для створення RESTful API [8]. Альтернативами можуть бути Django (Python), Spring Boot (Java) або Laravel (PHP), залежно від вимог проєкту. Для зберігання й обробки даних бекенд часто використовує ORM-бібліотеки (Sequelize для Node.js, TypeORM або Mongoose для MongoDB), що спрощують взаємодію з базами даних.

Бази даних є невід'ємною частиною веб-сайтів, оскільки вони використовуються для зберігання інформації про користувачів, новини, статистику та інші дані. Серед реляційних баз даних найпопулярнішими є MySQL та PostgreSQL, які забезпечують надійність, масштабованість і підтримку складних запитів. Якщо платформа потребує гнучкішої схеми даних, використовуються NoSQL-бази, такі як MongoDB, що зберігає дані у форматі JSON. Для швидкого доступу до часто використовуваних даних застосовуються кешуючі системи на основі Redis або Memcached, що значно знижує навантаження на основну базу даних [9].

Хостинг та серверні рішення забезпечують розгортання веб-додатку в продакшн-середовищі. Використовуються як традиційні віртуальні сервери (VPS) на DigitalOcean, Linode або AWS EC2, так і безсерверна-архітектура, наприклад, Vercel або Netlify для фронтенду та AWS Lambda для бекенду. Також популярними є хмарні рішення на основі Docker та Kubernetes, що забезпечують контейнеризацію й автоматичне масштабування застосунку.

1.3 Архітектурні підходи до розробки веб-сайтів

Архітектурні підходи до розробки веб-сайтів визначають, як будуть організовані та взаємодіяти між собою різні компоненти системи, що є важливим для забезпечення масштабованості, надійності та продуктивності веб-додатків. Сучасні веб-сайти, особливо ті, що мають великий обсяг трафіку та складну функціональність, потребують ретельного підходу до архітектури для забезпечення ефективного обслуговування користувачів та високої швидкості роботи [10].

Один із основних архітектурних підходів – це клієнт-серверна архітектура. Вона передбачає, що клієнт запитує інформацію або послуги від сервера, який обробляє ці запити та надає відповідні дані. Сервер може бути центральним, або ж розподіленим, залежно від масштабів застосунку. Такі підходи використовуються в більшості сучасних веб-сайтів і підтримуються численними фреймворками для створення API та обробки запитів.

Одним із важливих підходів є архітектура на основі односторінкових застосунків (SPA), де весь інтерфейс завантажується лише один раз при першому доступі до сайту, а всі подальші запити та взаємодії обробляються за допомогою JavaScript. Це дозволяє значно зменшити навантаження на сервер, а користувач отримує більш швидку та плавну взаємодію з сайтом без необхідності перезавантаження сторінок. SPA часто використовується в поєднанні з фреймворками, такими як React.js, Angular, або Vue.js для фронтенду. Для реалізації цієї архітектури використовуються REST API або GraphQL для обміну даними між фронтендом і бекендом [11].

Інший популярний підхід – це мультишарова архітектура, яка передбачає розподіл додатка на кілька рівнів або шарів. У типовій мультишаровій архітектурі є окремі рівні для представлення (UI), бізнес-логіки та даних. Це дозволяє легко змінювати один із шарів без впливу на інші, що робить систему більш гнучкою й легко масштабованою. На прикладі веб-сайтів для огляду спортивних подій, фронтенд зазвичай працює з API для отримання інформації, бекенд обробляє бізнес-логіку та взаємодіє з базою даних, а сама база даних зберігає інформацію про події, команди, результати тощо.

Мікросервісна архітектура є ще одним важливим підходом, коли додаток розбивається на кілька самостійних сервісів, кожен з яких відповідає за конкретну частину функціональності. Такий підхід дозволяє масштабувати окремі сервіси в залежності від навантаження, а також спрощує розробку та тестування. Веб-сайти, побудовані за принципом мікросервісів, мають окремі сервіси для обробки запитів користувачів, роботи з базами даних, управління аутентифікацією, обробки платежів тощо. Мікросервіси спілкуються через стандартні протоколи, такі як HTTP або gRPC. Цей підхід є корисним для великих проєктів з високими вимогами до доступності та гнучкості [12].

Архітектура з розподіленими базами даних є важливою для платформ, що мають велику кількість користувачів або великий обсяг даних. Вона дозволяє зберігати дані в різних фізичних локаціях, що забезпечує високу доступність та стійкість до відмов. Це може включати використання баз даних, що реплікуються або автоматично масштабуються на основі навантаження. Такі підходи активно

використовуються в е-комерції та соціальних мережах, де потрібно підтримувати безперервний доступ до даних навіть під час великих пікових навантажень [13].

Усі ці підходи вимагають уважного планування та вибору відповідних технологій, щоб забезпечити високу продуктивність, надійність і масштабованість веб-сайту. Для веб-сайту, що надає огляди спортивних подій, критично важливо вибрати таку архітектуру, яка забезпечить швидку обробку запитів, гнучкість для масштабування та легкість у підтримці високої доступності сервісу.

1.4 Постановка завдання та обгрунтування вибору технологій

Постановка завдання є важливою частиною процесу розробки веб-сайту, оскільки вона визначає основні цілі, вимоги та функціональні можливості майбутньої платформи. Для веб-сайту, що надає огляди спортивних подій, завдання полягає в створенні інтерактивного, зручного і швидкого ресурсу для користувачів, який надаватиме актуальну інформацію про спортивні події, результати матчів, новини та статистику. Веб-сайт повинен забезпечити високу продуктивність, щоб забезпечити мінімальний час завантаження та швидке відображення сторінок. Користувачі часто взаємодіють із сайтом в реальному часі, особливо під час спортивних подій, тому важливо створити платформу, яка надасть швидкий доступ до даних, зберігаючи при цьому стабільну роботу в умовах високого навантаження [14].

Крім того, для забезпечення актуальності та точності даних система повинна мати механізм для регулярного оновлення інформації з різних джерел. Це дозволить користувачам отримувати найновіші результати матчів, статистику та новини, що є важливим аспектом для спортивних сайтів. Оскільки кількість користувачів може значно зростати під час великих спортивних подій, такими як фінали турнірів або матчі популярних ліг, система повинна бути масштабованою, щоб забезпечити ефективну обробку великої кількості запитів одночасно, не знижуючи продуктивність.

Окрім того, важливим аспектом є зручність і інтуїтивно зрозумілий інтерфейс. Користувачі повинні мати змогу швидко і легко знаходити потрібну інформацію, переглядати новини та результати матчів, а також користуватися додатковими функціями сайту, такими як фільтри для пошуку подій чи турнірів. Веб-сайт повинен бути адаптивним і добре працювати на різних пристроях, включаючи мобільні телефони, що дозволяє досягти максимальної доступності для користувачів. Таким чином, основне завдання полягає в створенні ефективного, функціонального і зручного інструменту для перегляду спортивних подій, який відповідає сучасним вимогам користувачів і технічним стандартам [15].

Вибір технологій для реалізації цього проєкту обґрунтований необхідністю створення швидкої, масштабованої та інтерактивної платформи. Використання React.js для розробки фронтенд-частини дозволяє створити динамічний інтерфейс, що забезпечує високу продуктивність та зручність у використанні. React.js є однією з найбільш популярних бібліотек для розробки інтерфейсів користувача завдяки своїй здатності швидко оновлювати частини веб-сторінки при зміні стану програми, що особливо важливо для сайтів, де інформація повинна оновлюватися в реальному часі. Це також дозволяє значно зменшити навантаження на сервери, оскільки фронтенд обробляє багато операцій без необхідності запитувати сервер щоразу, коли потрібно оновити дані.

Для бекенд-частини вибір упав на Node.js з Express.js через їх здатність ефективно обробляти велику кількість одночасних запитів завдяки асинхронному виконанню, що особливо важливо для проекту, який може мати велику кількість користувачів [16]. Node.js надає високу продуктивність і дозволяє створювати масштабовані системи з можливістю обробки великої кількості запитів за короткий проміжок часу. Враховуючи ці фактори, ці технології дозволяють створити надійну, швидку та масштабовану платформу для спортивного сайту.

1.5 Висновок до першого розділу

Перший розділ дипломної роботи присвячений аналізу предметної області та вибору технологій для створення веб-сайту для огляду спортивних подій. В результаті проведеного аналізу було визначено основні вимоги до такого ресурсу, серед яких важливими є швидкість роботи, масштабованість, актуальність та доступність даних, а також зручність користування. Вивчення існуючих веб-сайтів показало, що спортивні платформи повинні бути інтерактивними, забезпечувати оперативне оновлення інформації і мати простий та зрозумілий інтерфейс. Найбільшу увагу слід приділити ефективному управлінню даними, що потребує надійної архітектури та правильного вибору технологій.

З огляду на ці вимоги, вибір технологій для розробки веб-сайту обґрунтовано з точки зору високої продуктивності, гнучкості та можливості масштабування. Використання React.js для фронтенду дозволяє створити динамічний та високопродуктивний інтерфейс, здатний оперативно відображати оновлення без необхідності перезавантаження сторінки. Вибір Node.js з Express.js для бекенду обумовлений їх здатністю ефективно працювати з великою кількістю одночасних запитів, що є критичним для платформи з високим навантаженням, такою як спортивний сайт [17].

Таким чином, в результаті проведеного аналізу можна зробити висновок, що вибір цих технологій відповідає вимогам до створення ефективної, зручної та масштабованої веб-сайту для огляду спортивних подій. Наступні етапи розробки будуть спрямовані на реалізацію архітектури сайту та забезпечення його високої продуктивності та зручності для кінцевих користувачів.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ВЕБ-САЙТУ

2.1 Опис архітектури веб-сайту

Архітектура веб-сайту для огляду спортивних подій є ключовим аспектом розробки, що забезпечує надійність, масштабованість, ефективну взаємодію з користувачем та простоту обслуговування. У цьому проєкті реалізовано класичну клієнт-серверну архітектуру, в якій фронтенд і бекенд функціонують як окремі, але взаємодіючі частини, що дозволяє досягти високої гнучкості та розширюваності системи [18].

Фронтенд-частина реалізована з використанням сучасної JavaScript-бібліотеки React.js, яка дозволяє створювати зручний, динамічний та інтуїтивно зрозумілий інтерфейс у форматі SPA (Single Page Application). Компонентний підхід React забезпечує повторне використання коду та полегшує супровід. Додаток оновлюється швидко та плавно завдяки віртуальному DOM, що покращує користувацький досвід.

Бекенд-частина побудована на Node.js у зв’язці з Express.js, що надає гнучкі засоби для створення API, обробки HTTP-запитів, маршрутизації, автентифікації та роботи з базою даних. Сервер відповідає за зберігання, обробку і повернення даних фронтенду через REST API. Обмін даними відбувається у форматі JSON, що спрощує інтеграцію між частинами застосунку [19].

Для зберігання даних обрано MongoDB – документно-орієнтовану нереляційну базу даних, що відмінно підходить для динамічних та масштабованих веб-додатків. MongoDB зберігає інформацію у вигляді гнучких документів формату BSON, які подібні до JSON-об’єктів. Це дозволяє зручно зберігати дані про новини, події, матчі, команди, користувачів та інші об’єкти без необхідності жорсткого визначення схем, що особливо корисно в умовах частої зміни структури даних. MongoDB також підтримує індексацію, агрегацію та швидкий пошук, що підвищує продуктивність застосунку при роботі з великим обсягом інформації.

Архітектура також включає механізми автентифікації та авторизації користувачів на основі JWT (JSON Web Token), що дозволяє безпечно зберігати сесії, обмежувати доступ до певних ресурсів та реалізовувати рольову модель доступу (наприклад, звичайний користувач та адміністратор). Дані паролів користувачів шифруються за допомогою алгоритму bcrypt, що підвищує рівень безпеки [20].

Загальна архітектура побудована за принципами модульності, розділення обов'язків (Separation of Concerns) та RESTful-дизайну, що дозволяє легко підтримувати, оновлювати та розширювати функціональність веб-сайту. Обрана структура є сучасним та ефективним рішенням для реалізації веб-сайту огляду спортивних подій з актуальним контентом і широкими можливостями для подальшого розвитку.

2.2 Проєктування фронтенд-частини веб-сайту

Фронтенд-частина веб-сайту розробляється з використанням бібліотеки React.js, яка дозволяє створювати сучасні, швидкі, масштабовані та зручні односторінкові застосунки. Завдяки використанню компонентного підходу, інтерфейс веб-сайту поділено на логічно ізольовані модулі, кожен з яких відповідає за окрему функціональність або візуальну частину (див. рисунок 2.1).

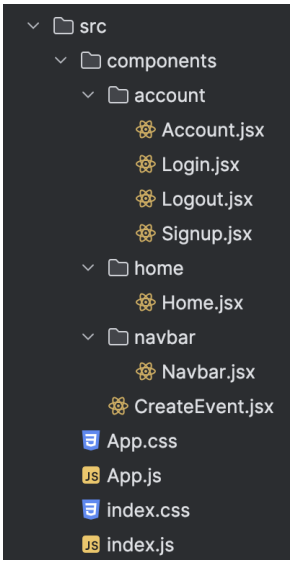

Рисунок 2.1 – Структура фронтенд частини

У структурі проєкту всі компоненти розміщено в директорії src/components, яка поділяється на підкаталоги відповідно до функціональних блоків: account, home, navbar та інші. Такий підхід дозволяє логічно впорядкувати код, полегшує навігацію та супровід, особливо при масштабуванні застосунку [21].

Директорія account містить компоненти, пов’язані з обліковим записом користувача: Login.jsx, Signup.jsx, Logout.jsx та Account.jsx. Компоненти Login і Signup реалізують форми авторизації та реєстрації, що дозволяють користувачам створювати акаунти та входити в систему. Дані форм передаються на сервер через REST API, після чого у разі успішної авторизації користувач отримує токен доступу (JWT), який зберігається у локальному сховищі. Компонент Logout.jsx очищає токен і перенаправляє користувача на головну сторінку, завершуючи сесію. Account.jsx використовується для відображення профілю користувача або інших персоналізованих даних.

Директорія home містить компонент Home.jsx, який відповідає за рендеринг головної сторінки сайту. Тут можуть виводитись новини, спортивні огляди, події та інший контент, доступний усім відвідувачам. Інформація може бути отримана з бекенду через API-запити й оновлюватись динамічно без перезавантаження сторінки [22].

Директорія navbar включає компонент Navbar.jsx, який реалізує навігаційне меню веб-сайту. Цей компонент є спільним для більшості сторінок і забезпечує швидкий доступ до основних розділів. Компонент CreateEvent.jsx дозволяє користувачам (адміністраторам) створювати нові події чи матчі, вказуючи відповідну інформацію через інтерактивну форму.

Файл App.js є основним кореневим компонентом, який поєднує всі вищезгадані компоненти та реалізує маршрутизацію за допомогою бібліотеки react-router-dom. У цьому файлі визначаються маршрути для переходу між сторінками, наприклад, /login, /signup, /home, /account, /create-event тощо. Такий підхід дозволяє реалізувати SPA-архітектуру, при якій нові сторінки завантажуються без повного перезавантаження браузера [23].

Файл index.js є точкою входу у застосунок, в якому відбувається рендеринг головного компоненту App у DOM-елемент. Також тут можуть підключатись глобальні стилі, контексти або провайдери.

Системні файли, такі як App.css та index.css, містять стилі для базових елементів інтерфейсу.

Загалом, проєктування фронтенд-частини відображає сучасний підхід до розробки веб-додатків, що базується на розділенні логіки за функціональними зонами, повторному використанні компонентів, зручному маршрутизуванні та інтеграції з сервером через API. Обрана структура сприяє легкому масштабуванню, тестуванню, рефакторингу та майбутньому розширенню функціональності сайту [24].

2.3 Проєктування бекенд-частини веб-сайту

Бекенд-частина веб-сайту реалізована за допомогою середовища виконання Node.js у поєднанні з фреймворком Express.js та базою даних MongoDB, для роботи з якою використовується ODM-бібліотека Mongoose. Такий стек технологій дозволяє створювати гнучке та масштабоване API, яке легко поєднується з фронтендом через HTTP-запити.

Всі серверні файли зосереджено у папці src, яка містить підкаталоги config, model, routes та файл index.js – точку входу в застосунок (див. рисунок 2.2).

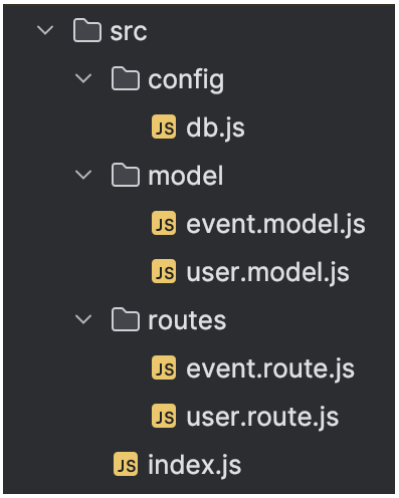

Рисунок 2.2 – Структура бекенд-частини

У папці config знаходиться файл db.js, який відповідає за підключення до бази даних MongoDB. У цьому файлі зазвичай виконується ініціалізація з’єднання з використанням URI (отриманого з .env або конфігурації), а також обробка подій успішного підключення або помилок [25].

Директорія model містить файли з моделями event.model.js та user.model.js. Вони описують структуру документів, що зберігаються в базі даних MongoDB, за допомогою схем Mongoose.

Модель події включає наступні поля:

- desc – опис події;
- date – дата проведення;
- venue – місце проведення;
- playersLimit – ліміт учасників;
- type – тип події (наприклад, футбольний матч, баскетбол тощо);
- org_id та org_name – ідентифікатор та ім’я організатора;
- players – масив учасників події.

Додатково, завдяки параметрам timestamps: true та versionKey: false, до кожного запису автоматично додаються поля створення та оновлення, а поле версії (__v) виключається.

Модель користувача містить:

- email – адреса електронної пошти, яка перевіряється на валідність за допомогою бібліотеки validator;
- password – пароль (має зберігатись у хешованому вигляді);
- role – роль користувача (звичайний користувач або організатор).

Усі маршрути зосереджені в папці routes. Вони розділені за функціоналом:

- event.route.js – містить маршрути для створення, перегляду, редагування та видалення подій;
- user.route.js – маршрути для реєстрації, входу в систему, отримання інформації про користувача тощо.

Ці файли експортують Express-роутери, які згодом підключаються до основного застосунку у файлі index.js.

У файлі index.js створюється та налаштовується сервер на базі Express. У цьому файлі:

- Підключаються всі необхідні модулі.
- Виконується з’єднання з базою даних через db.js.
- Підключаються маршрути user.route.js та event.route.js.
- Запускається сервер на вказаному порту.

Цей файл є ядром сервера, який відповідає за обробку всіх запитів, перенаправлення до відповідних маршрутів, а також налаштування проміжних функції для роботи з JSON, CORS тощо.

2.4 Висновок до другого розділу

У другому розділі було розглянуто процес проєктування веб-сайту для огляду спортивних подій, включаючи як фронтенд-, так і бекенд-частини системи. Розробка здійснювалася із застосуванням сучасного технологічного стеку, що забезпечує надійність, масштабованість і зручність у подальшому супроводі та розширенні системи [26].

На етапі проєктування фронтенд-частини була сформована чітка структура компонентів, згрупованих за функціональними зонами: автентифікація, головна сторінка, навігація тощо. Компоненти React.js реалізовано таким чином, щоб забезпечити модульність, повторне використання коду та зручність у розширенні функціональності. Користувацький інтерфейс було організовано відповідно до сучасних принципів UX/UI-дизайну, що гарантує зручну взаємодію з сайтом для різних категорій користувачів.

Бекенд-частина веб-застосунку реалізована на основі Node.js із використанням фреймворку Express.js та бази даних MongoDB. У ході проєктування було створено дві основні моделі: модель користувача та модель події, що повністю відображають логіку доменної області. Усі маршрути були чітко розділені за функціоналом та організовані в окремі файли, що підвищує підтримуваність і зручність в обслуговуванні системи. Забезпечено валідацію

вхідних даних, зокрема перевірку електронної пошти, та структурування об’єктів у базі даних за допомогою схем Mongoose [27].

Загалом, структура проєкту є логічною, зручною та відповідає принципам розділення відповідальностей (Separation of Concerns). Усі компоненти веб-сайту – як клієнтська, так і серверна частини – взаємодіють між собою через API, що дозволяє легко масштабувати систему в майбутньому, зокрема шляхом додавання нових типів подій, функціоналу для адміністрування або аналітики.

Таким чином, розроблена архітектура сайту повністю відповідає поставленим функціональним вимогам і є надійною основою для реалізації подальших етапів – впровадження, тестування та супроводу веб-сайту.

РОЗДІЛ 3 РОЗРОБКА ТА ТЕСТУВАННЯ ВЕБ-САЙТУ

3.1 Реалізація фронтенд-частини веб-сайту на основі React.js

Ключовим елементом клієнтської частини веб-застосунку є компонент App, який є кореневим компонентом усього інтерфейсу. Його основною функцією є організація глобального стану, ініціалізація даних, перевірка автентифікації користувачів та обгортання інших компонентів контекстами. Нижче розглянемо його поетапно [28].

На початку компонента імпортуються необхідні залежності: хуки useState та useEffect, що дозволяють працювати зі станом і побічними ефектами, а також компоненти Navbar і Home, які формують інтерфейс додатку. Також використовується бібліотека axios для здійснення HTTP-запитів до бекенд-сервера. Крім того, створюються два контексти – CreateEventContext і GetEventContext, які забезпечують доступ до відповідних змінних стану в інших компонентах, не передаючи їх через props (див. лістинг 3.1).

Лістинг 3.1 – Імпорт залежностей

```
import {createContext, useEffect, useState} from 'react';
import './App.css';
import Navbar from './components/navbar/Navbar';
import Home from './components/home/Home';
import axios from 'axios'
export const CreateEventContext = createContext();
export const GetEventContext = createContext();
```

У компоненті App визначено кілька станів: isOrganizerLoggedIn і isUserLoggedIn, які вказують, чи авторизований користувач або організатор відповідно, а також eventList, що зберігає список подій, отриманий із серверної частини. При монтуванні компонента виконується перевірка локального сховища localStorage на наявність об’єкта користувача під ключем sportEvent. Якщо такий об’єкт знайдено і його роль – організатор, відповідний стан змінюється, що дозволяє одразу відобразити відповідний інтерфейс без повторного входу (див. лістинг 3.2).

Лістинг 3.2 – Створення станів

```
const [isOrganizerLoggedIn, setIsOrganizerLoggedIn] =
useState(false);
const [isUserLoggedIn, setIsUserLoggedIn] = useState(false);
const [eventList, setEventList] = useState([]);
useEffect(() => {
     if (localStorage.getItem("sportEvent")) {
          const profile = JSON.parse(localStorage.getItem(
               "sportEvent"));
          ...
     }
}, []);
```

Окремо реалізована функція getEventList, яка надсилає запит на адресу http://localhost:8080/event, отримує список подій з бекенду й оновлює змінну стану eventList (див. лістинг 3.3). Ця функція надається до глобального контексту GetEventContext, що дає змогу викликати її з будь-якої частини інтерфейсу [29].

Лістинг 3.3 – Функція getEventList

```
const getEventList = async () => {
     await axios.get("http://localhost:8080/event")
          .then((res) => {
               setEventList(res.data)
          })
          .catch((e) => {
               console.log(e)
          })
}
```

У розмітці компонента JSX використано обгортання всіх дочірніх елементів у два контексти. Компонент Navbar, розміщений всередині CreateEventContext, має доступ до інформації про авторизацію користувача, що дозволяє динамічно змінювати його вміст залежно від ролі. Компонент Home, який знаходиться нижче, отримує доступ до списку подій через GetEventContext і відповідає за їх візуалізацію на головній сторінці.

Компонент Home відповідає за відображення списку подій, пошук подій за ключовими словами та можливість приєднатися до подій через модальне вікно. Спочатку імпортуються необхідні залежності, зокрема хуки useContext, useState,

useEffect, які дозволяють працювати з контекстом, станом та побічними ефектами (див. лістинг 3.4).

Лістинг 3.4 – Імморт залежностей

```
import React, { useContext, useEffect, useState } from 'react'
import { Box, Button, Input, Modal, ModalBody, ModalCloseButton,
ModalContent, ModalFooter, ModalHeader, ModalOverlay, SimpleGrid,
Text, useDisclosure, useToast } from '@chakra-ui/react'
import { GetEventContext } from '../../App'
import axios from 'axios'
```

Цей код імпортує необхідні компоненти з бібліотек: React, useContext, useEffect, useState для роботи з контекстом, життєвим циклом компонентів та станом. Компоненти з бібліотеки Chakra UI для створення стилізованих елементів інтерфейсу. axios для здійснення HTTP запитів. GetEventContext з іншого файлу для доступу до глобального стану [30].

Далі визначається ініціалізація стану.

```
const initState = {
     players: []
}
```

Це базовий стан для обраної події, де зберігаються гравці. Компонент використовує кілька стани, таких як rowSeleted, щоб зберігати дані про обрану подію, та keyword, щоб зберігати ключове слово для пошуку. Також використовується локальне сховище (localStorage), щоб отримати дані про користувача, зокрема, якщо користувач вже увійшов, це дозволяє одразу показати відповідний інтерфейс.

```
const profile = JSON.parse(localStorage.getItem("sportEvent")) ||
"";
```

Це отримує дані про користувача з localStorage. Якщо об’єкт користувача не знайдений, то значення profile залишатиметься порожнім рядком.

Далі, у компоненті є кілька функцій для обробки подій. Наприклад, функція handleJoinBtn відповідає за додавання користувача до події. Вона перевіряє, чи є користувач авторизованим, і чи має він роль організатора, оскільки організатори не можуть приєднуватися до подій (див. лістинг 3.5).

Лістинг 3.5 – Функція handleJoinBtn

```
const handleJoinBtn = async (id) => {
if (!profile) {
     ...
if (profile.role === "organizer") {
     ...
await axios({
          url: `http://localhost:8080/event/${id}`,
          method: "PUT",
```

Ця функція здійснює запит до серверу для додавання користувача до події через метод PUT. Якщо все проходить успішно, відображається повідомлення про успіх, оновлюється список подій і закривається модальне вікно.

Функція handleSearch відповідає за пошук подій за ключовим словом. Вона надсилає запит на сервер і оновлює список подій на основі результатів (див. лістинг 3.6).

Лістинг 3.6 – Функція handleSearch

```
const handleSearch = async (e) => {
e.preventDefault();
     await axios.get(`http://localhost:8080/event/search?
          keyword=${keyword}`)
     .then((res) => {
          setEventList(res.data)
     })
     ...
     setKeyWord("")
```

Користувач може натискати кнопку “Search”, щоб знайти події, що відповідають введеному ключовому слову.

Компонент також відображає список подій у вигляді карток, використовуючи компонент SimpleGrid з Chakra UI (див. лістинг 3.7).

Лістинг 3.7 – Створення списку подій

```
<SimpleGrid columns={[1, 1, 1, 3]} gap={10}>
     {eventList?.map((el) => (
          <Box key={el._id} boxShadow='outline' p='6' rounded='md'
          bg='white' fontFamily={'cursive'}>
          ...
          <Box textAlign={'center'} boxShadow='xl' p='6'
               rounded='md' bg='white'
               color={'gray'}>{el.desc}</Box>
          <Text textAlign={'left'} mt={3}
          fontSize={'2xl'}>{el.type}</Text>
          <Text textAlign={'left'} mt={3}>Organizer Name:
          {el.org_name}</Text>
          ...
```

Кожна подія відображається у вигляді картки, що включає в себе інформацію про місце проведення, дату, опис, тип події та організатора. Кнопка “More Detail” відкриває модальне вікно з додатковою інформацією [31].

Модальне вікно дозволяє переглядати деталі події та приєднуватися до неї, якщо користувач ще не є частиною команди. Кнопка “Join” викликає функцію handleJoinBtn (див. лістинг 3.8).

Лістинг 3.8 – Створення модального вікна

```
<Modal isOpen={isOpen} onClose={onClose}>
  <ModalOverlay />
  <ModalContent>
    <ModalCloseButton />
    <ModalBody fontFamily={'cursive'}>
      ...
    </ModalBody>
    <ModalFooter>
      ...
    </ModalFooter>
  </ModalContent>
</Modal>
```

Це модальне вікно дозволяє користувачам переглядати інформацію про ліміт гравців для події та список учасників, а також приєднуватися до події.

Компонент завершується розміткою, яка обгортає всі елементи в контейнер з певними стилями (див. лістинг 3.9).

Лістинг 3.9 – Створення контейнеру

```
<Box w={'90%'} m={'auto'} mt={'100px'}>
     <Text fontFamily={'cursive'} fontSize={'3xl'} mb={10}
     color={'red'}>Events</Text>
     ...
</Box>
```

Компонент Account реалізує модальне вікно (drawer) для управління акаунтом користувача, в якому доступні вкладки для реєстрації, входу та виходу з акаунту. Спочатку імпортуються необхідні компоненти та хуки, зокрема useDisclosure для керування відкриттям і закриттям модального вікна, а також компоненти Chakra UI для створення інтерфейсу.

Компонент використовує useDisclosure для керування відкриттям та закриттям модального вікна. Для кнопки, яка викликає модальне вікно, створюється ref.

```
const { isOpen, onOpen, onClose } = useDisclosure()
const btnRef = React.useRef()
const { isUserLoggedIn } = useContext(CreateEventContext);
```

Кнопка для відкриття модального вікна має іконку профілю і викликає функцію onOpen для відкриття модального вікна (див. лістинг 3.10).

Лістинг 3.10 – Створення кнопки для відкриття модального вікна

```
<Button ref={btnRef} colorScheme='white' variant='outline'
onClick={onOpen}>
     <CgProfile size={28}/>
</Button>
```

Коли користувач натискає на цю кнопку, відкривається компонент Drawer, що містить три вкладки: “Signup”, “Login” та “Logout”. Ці вкладки відображаються в залежності від того, чи користувач авторизований. Якщо користувач не авторизований, то відображаються вкладки для реєстрації та входу, а якщо авторизований – вкладка для виходу з акаунту. Вкладки реалізуються за допомогою компонента Tabs з Chakra UI, де кожна вкладка містить окремий компонент (див. лістинг 3.11).

Лістинг 3.11 – Створення спливаючого вікна для входу та реєстрації

```
<Drawer isOpen={isOpen} placement='right' onClose={onClose}
finalFocusRef={btnRef} size={"md"}>
  <DrawerOverlay />
  <DrawerContent height="74%  !important">
    <DrawerCloseButton />
    ...
    <DrawerBody>
      <Tabs size='md' variant='enclosed'>
     ...
      </Tabs>
    </DrawerBody>
    <DrawerFooter>
      ...
```

Компонент Drawer містить в собі заголовок, основну частину з вкладками і кнопки для закриття або скасування дій. Вкладки умовно відображаються в залежності від стану авторизації користувача:

- Якщо користувач не авторизований, відображаються вкладки для реєстрації та входу.
- Якщо користувач авторизований, то замість вкладок для реєстрації та входу відображається вкладка для виходу з акаунту.
- Вкладки “Signup”, “Login” та “Logout” містять відповідні компоненти:
- Signup – компонент для реєстрації.
- Login – компонент для входу в систему, з пропсом closeDrawer, який закриває модальне вікно після успішного входу.
- Logout – компонент для виходу з акаунту.

Наприкінці додається кнопка для скасування дії, яка просто закриває модальне вікно.

```
<Button variant='outline' mr={3} onClick={onClose}>
     Cancel
</Button>
```

Компонент Login реалізує форму для входу користувача в систему, використовуючи бібліотеку Chakra UI для стилізації та компонентів інтерфейсу. Спочатку імпортуються необхідні компоненти, такі як Button, Input для

створення кнопки та полів вводу, а також useToast для відображення повідомлень користувачеві про успішний або неуспішний вхід.

Ініціалізується стан форми за допомогою хука useState, де зберігаються значення полів email і password (див. лістинг 3.12). Також використовуються контекст CreateEventContext для доступу до стану авторизації, зокрема для визначення, чи є користувач організатором, а також функцій для оновлення стану [32].

Лістинг 3.12 – Об’явлення стану

```
const initState = {
     email: "",
     password: "",
}
const Login = ({ closeDrawer }) => {
     const [form, setForm] = useState(initState);
     const toast = useToast()
     const { isOrganizerLoggedIn, setIsOrganizerLoggedIn,
     setIsUserLoggedIn } = useContext(CreateEventContext);
```

Функція handleChange обробляє зміну значень у полях вводу, оновлюючи відповідні значення в стані.

Функція createAccount відповідає за обробку відправки форми. Вона надсилає запит на сервер для перевірки введених даних користувача за допомогою axios. Якщо запит успішний, дані користувача зберігаються в localStorage, а також змінюється стан авторизації в контексті. Якщо роль користувача – організатор, то відповідний стан також оновлюється. Після успішного входу модальне вікно закривається, і користувачу показується повідомлення про успішний вхід (див. лістинг 3.13).

Лістинг 3.13 – Функція createAccount

```
async function createAccount(e) {
     e.preventDefault();
     await axios({
          url: "http://localhost:8080/user/login",
          method: "POST",
     ...
    .then((res) => {
     localStorage.setItem("sportEvent",
```

```
            JSON.stringify(res.data));
    ...
```

Форма складається з двох полів вводу – для електронної пошти та паролю, обидва з яких обробляються за допомогою handleChange (див. лістинг 3.14).

Лістинг 3.14 – Два поля для вводу

```
<Input
     type={"email"}
     onChange={handleChange}
     ...
/>
<Input
     type={"password"}
     onChange={handleChange}
     ...
/>
```

Кнопка “Login” відправляє форму, викликаючи функцію createAccount.

```
<Button type='submit' mt={10} colorScheme='teal' size='md'>
     Login
</Button>
```

По завершенні обробки, форма очищається.

```
setForm({
     email: "",
     password: ""
});
```

Компонент Logout відповідає за виведення користувача з системи, очищаючи стан авторизації та видаляючи інформацію про користувача з localStorage. Він використовує Chakra UI для відображення інтерфейсу та інтерактивних елементів, таких як кнопка та повідомлення для користувача.

Перш за все, в компоненті імпортуються необхідні елементи: Box, Button, Text, а також хук useToast, який забезпечує виведення повідомлень користувачеві. Крім того, використовуються іконки з бібліотеки react-icons (конкретно іконка FaRegSmileWink) для покращення візуального вигляду. Також

імпортується контекст CreateEventContext, який дозволяє оновлювати глобальні стани авторизації [33].

Компонент Logout використовує хук useToast для показу повідомлень про результат операції (успішний вихід або необхідність спочатку увійти до системи). Важливим моментом є перевірка наявності об’єкта користувача в localStorage, що дає змогу визначити, чи потрібно спочатку увійти для того, щоб вийти з системи.

```
const toast = useToast()
const { setIsOrganizerLoggedIn, setIsUserLoggedIn } =
useContext(CreateEventContext);
```

Функція handleClick виконується при натисканні на кнопку “Logout”. Спочатку вона перевіряє, чи є в localStorage дані користувача. Якщо користувач не авторизований, з’являється попередження (див. лістинг 3.15).

Лістинг 3.15 – Перевірка чи є в localStorage дані користувача

```
if (!localStorage.getItem("sportEvent")) {
     toast({
          title: 'Login First.',
          ...
     })
```

Якщо користувач авторизований, після натискання на кнопку відображається повідомлення про успішний вихід з системи, а також відбувається скидання станів isOrganizerLoggedIn і isUserLoggedIn через контекст. Також видаляються дані користувача з localStorage (див. лістинг 3.16).

Лістинг 3.16 – Логіка якщо користувач авторизовний

```
else {
     toast({
          title: 'Logout Successfully.',
          description: "",
          ...
     setIsOrganizerLoggedIn(false);
     setIsUserLoggedIn(false);
}
localStorage.removeItem("sportEvent");
```

У розмітці компоненту використовуються елементи Chakra UI для створення кнопки та виведення тексту. На початку додається текст “Come back soon”, що супроводжується іконкою FaRegSmileWink, яка додає компоненту більш дружній та приємний вигляд. Це елемент виводиться у блоці, що має флекс-стилізацію для вирівнювання елементів (див. лістинг 3.17).

Лістинг 3.17 – Форма для виходу з акаунту

```
<Box display={'flex'} gap={2} mt={5}>
     <Text mt={-3} mb={8} fontFamily={'cursive'}
     fontSize={'3xl'}>Come back soon</Text>
     <FaRegSmileWink size={28}/>
</Box>
```

Кнопка “Logout” реалізована як компонент Button з кольором teal і розміром md. Вона викликає функцію handleClick при натисканні, що здійснює вихід з системи.

```
<Button onClick={handleClick} colorScheme='teal'
size='md'>Logout</Button>
```

Компонент Signup реалізує форму для реєстрації нового користувача. Вона дозволяє заповнити поля для імені, електронної пошти та пароля, а також вибрати роль користувача через чекбокс, де можна вказати, чи є користувач організатором подій. Використовуються компоненти Chakra UI, такі як Button, Checkbox, Input, а також хук useToast для показу повідомлень користувачу після спроби реєстрації.

Спочатку імпортуються необхідні компоненти з Chakra UI, бібліотека axios для здійснення HTTP запитів і хук useState для роботи з локальним станом.

Ініціалізація стану форми з початковими значеннями для полів: name, email, password і role (див. лістинг 3.18).

Лістинг 3.18 – Створення початкового стану

```
const initState = {
    name: "",
    email: "",
```

```
    password: "",
    role: "user"
}
```

Компонент використовує хук useState для відстеження значень, введених користувачем у форму.

```
const [form, setForm] = useState(initState);
const toast = useToast();
```

Функція handleChange обробляє зміни в полях вводу. Вона відповідає за оновлення відповідного поля форми в стані компонента. Якщо змінюється значення перемикача (який визначає, чи є користувач організатором), то змінюється роль на “organizer” (див. лістинг 3.19).

Лістинг 3.19 – Функція handleChange

```
const handleChange = (e) => {
     const { name, value, type } = e.target;
     const valueToUpdate = type === "checkbox" ? "organizer" :
     value;
     setForm({
          ...form,
          [name]: valueToUpdate
```

Функція createAccount викликається при відправленні форми. Вона здійснює асинхронний запит на сервер для створення нового користувача. Якщо реєстрація пройшла успішно, виводиться повідомлення через useToast, а поля форми очищаються. Якщо виникає помилка, наприклад, якщо електронна пошта вже використовується, з’являється відповідне попередження (див. лістинг 3.20).

Лістинг 3.20 – Функція createAccount

```
async function createAccount(e) {
     e.preventDefault();
     await axios({
          url: "http://localhost:8080/user/signup",
          method: "POST",
     ...
     setForm({
          name: "",
```

```
        ...
        role: "user"
    });
```

Відображення форми включає три поля вводу для імені, електронної пошти та пароля. Для пароля та електронної пошти використовується тип password і email відповідно [34]. Всі поля є обов’язковими для заповнення. Також є чекбокс для вибору ролі користувача – організатора подій. При включенні перемикача роль змінюється на “organizer” (див. лістинг 3.21).

Лістинг 3.21 – Форма яка включає поля вводу для імені, електронної пошти та пароля

```
<Input
    type={"text"}
    onChange={handleChange}
    ...
/>
...
<Button type='submit' mt={20} colorScheme='teal' size='md'>
    Create Account
</Button>
```

Кнопка “Create Account” відправляє форму, викликаючи функцію createAccount для виконання HTTP запиту. Після успішної реєстрації або виникнення помилки, за допомогою useToast виводиться відповідне повідомлення.

Компонент Navbar реалізує навігаційну панель для веб-додатку, яка відображає різні елементи в залежності від того, чи авторизований організатор подій. Він використовує компоненти Chakra UI для стилізації, такі як Box, і контекст із CreateEventContext для визначення стану авторизації організатора.

У компоненті використовується хук useContext для доступу до контексту CreateEventContext, що зберігає інформацію про стан авторизації організатора подій.

```
const { isOrganizerLoggedIn } = useContext(CreateEventContext);
```

Компонент повертає JSX, що складається з розмітки для навігаційної панелі, яка фіксована в верхній частині екрану (position: fixed). Панель має стиль фону #010026, білий текст і специфічні розміри для різних розмірів екрану (див. лістинг 3.22).

Лістинг 3.22 – Розмітка навігаційної панелі

```
<Box
     w={'100%'}
     display={'flex'}
     ...
     top={0}
     zIndex={5}
>
     <Box color={'red'} fontSize={['2xl', '3xl']}>Sport
Events</Box>
```

У верхній частині панелі є заголовок “Sport Events”, який має червоний колір і розмір шрифта, який змінюється в залежності від розміру екрану (див. лістинг 3.23).

У правій частині панелі розміщено два елементи:

1. CreateEvent – цей компонент буде відображатися тільки у випадку, якщо організатор подій авторизований (isOrganizerLoggedIn має значення true).

2. Account – цей компонент завжди відображається.

Лістинг 3.23 – Відображення правої частини в панелі

```
<Box
     display={'flex'}
     gap={10}
>
     <Box>{isOrganizerLoggedIn ? <CreateEvent /> : ""}</Box>
     <Box><Account /></Box>
</Box>
```

Компонент Box з параметром display={'flex'} створює контейнер для розміщення елементів у рядку, а параметр gap={10} визначає відстань між ними. Якщо організатор не авторизований, компонент CreateEvent не буде відображатися, замість нього залишиться порожній елемент.

Компонент CreateEvent призначений для створення нових подій у додатку. Він використовує різні компоненти з бібліотеки Chakra UI, такі як кнопки, форми, модальні вікна та інші елементи інтерфейсу, щоб надати користувачу зручний інтерфейс для введення даних про подію. Оскільки створення події є важливою функцією, для зручності це відбувається в модальному вікні, яке відкривається при натисканні кнопки “Create Event”.

Компонент зберігає початкові значення для події, зокрема опис, дату, місце проведення, ліміт гравців, тип події та ідентифікацію організатора, що зберігаються в стані eventDetail. Ці значення використовуються для заповнення полів форми, що дозволяє користувачеві вводити необхідну інформацію. Якщо дані користувача (наприклад, профіль організатора) зберігаються в localStorage, то ці дані автоматично інтегруються в компонент через змінні org_id та org_name (див. лістинг 3.24).

Лістинг 3.24 – Перевірка чи є данні користувача а localStorage

```
const profile = JSON.parse(localStorage.getItem("sportEvent")) ||
        "";
    const initState = {
        desc: "",
        ...
        org_id: profile.id,
        org_name: profile.user
    }
```

Щоб здійснити зміни в полях форми, використовуються обробники подій. Функція handleChange відповідає за оновлення стану eventDetail при введенні даних у поля форми. Вона приймає подію зміни, отримує ім’я та значення змінюваного поля і відповідно оновлює стан (див. лістинг 3.25).

Лістинг 3.25 – Функція handleChange

```
const handleChange = (e) => {
    const { name, value } = e.target;
    setEventDetail({
        ...eventDetail,
        [name]: value
    });
}
```

При відправці форми викликається функція handleSubmit, яка відповідає за обробку запиту. Вона запобігає стандартній поведінці форми, виконує HTTP-запит через axios, відправляючи дані на сервер для збереження нової події. Після успішної обробки запиту відображається повідомлення через toast з підтвердженням успішного створення події. Якщо ж виникає помилка, з’являється попереджувальне повідомлення. Після цього стан форми скидається до початкових значень.

Що стосується зовнішнього вигляду, то основна частина інтерфейсу – це форма, що містить поля для введення опису події, дати, місця проведення, ліміту гравців і типу події. Всі поля мають відповідні підписи, і вони інтегровані зі станом компонента, що забезпечує двосторонній зв’язок між введеними даними та станом. Кнопка для відправки форми розташована внизу, а сама форма обгорнута в модальне вікно, яке зручно відкривається та закривається через функції onOpen та onClose з хука useDisclosure (див. лістинг 3.26).

Лістинг 3.26 – Спливаюче вікно для створення події

```
return (
    <>
             <Button colorScheme='white' variant='outline'
     onClick={onOpen}>Create Event</Button>
     <Modal blockScrollOnMount={false} isOpen={isOpen}
     onClose={onClose}>
            <ModalOverlay />
            <ModalContent>
    ...
```

Завдяки цьому підходу користувач може швидко і зручно створювати події, отримуючи зворотній зв’язок через повідомлення про успіх або помилки. Компонент інтегрується в загальний контекст додатку, оскільки після успішного додавання події оновлюється список подій, використовуючи функцію getEventList з контексту GetEventContext.

3.2 Реалізація бекенд-частини веб-сайту з використанням Node.js та Express.js

Почемо з основного файлу index.js. Для початку підключаються необхідні бібліотеки: express для створення веб-сервера, Mongoose для роботи з MongoDB, cors для забезпечення дозволу на запити до ресурсів з різних IP-адрес (CORS), а також body-parser, що дозволяє коректно обробляти тіла запитів у форматі JSON та URL-encoded (див. лістинг 3.27).

Лістинг 3.27 – Підключення бібліотек

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const bodyparser = require("body-parser");
```

Після підключення бібліотек, створюється інстанс сервера через express(), а також застосовуються middleware, так звані проміжні функції які викликаються між перед які викликаються між передачею запиту клієнта та надсиланням відповіді сервером. express.json() обробляє запити з JSON тілами, bodyparser.urlencoded({ extended: false }) дозволяє обробляти URL-encoded запити, а bodyparser.json() допомагає обробляти інші типи запитів з тілами у форматі JSON (див. лістинг 3.28). Додатково, на сервері використовується cors() для того, щоб дозволити запити між різними доменами, що є важливим для взаємодії з фронтенд частиною, яка може бути на іншому домені чи порту [35].

Лістинг 3.28 – Використання бібліотек

```
const app = express();
app.use(express.json());
app.use(cors());
app.use(bodyparser.urlencoded({extended: false}));
app.use(bodyparser.json());
```

Далі підключаються маршрути для обробки запитів до ресурсів користувачів та подій. Вони імпортуються з відповідних файлів за допомогою

require і додаються до додатку через app.use(). Це дозволяє розділити логіку обробки запитів і зробити серверну частину більш структурованою. Всі запити, що починаються з маршруту “user”, обробляються за допомогою маршруту userRouter, а запити до маршруту “event” – через eventRouter.

```
app.use("/user", userRouter);
app.use("/event", eventRouter);
```

Наступним кроком є налаштування з’єднання з MongoDB через Mongoose. Це з’єднання ініціюється за допомогою функції connect, яка імпортується з конфігураційного файлу. Важливо зазначити, що налаштовується mongoose.set("strictQuery", false), що дозволяє працювати з MongoDB без жорсткої перевірки запитів.

```
mongoose.set("strictQuery", false);
```

На головному маршруті сервер відповідає простим текстом, що інформує про те, що сервер працює – “Sport Events”. Це корисно для перевірки, чи сервер запущений і чи відповідає на запити.

```
app.get("/", (req,res)=> {
     res.send("Sport Events");
})
```

Завершується налаштування прослуховування порту через app.listen(), де сервер починає слухати на визначеному порту (в даному випадку 8080), і після успішного підключення до бази даних виводиться повідомлення про те, що сервер працює і доступний за відповідним URL (див. лістинг 3.29).

Лістинг 3.29 – Запуск застосунку

```
app.listen(PORT, async()=>{
     connect();
     console.log(`listening at port http://localhost:${PORT}`);
})
```

У файлі db.js реалізовано логіку підключення до бази даних MongoDB за допомогою бібліотеки Mongoose. На початку файлу відбувається імпорт модуля Mongoose, який є обгорткою для зручної роботи з базою даних MongoDB у Node.js (див. лістинг 3.30). Далі оголошується асинхронна функція connect, яка виконує підключення до віддаленої бази даних, використовуючи метод mongoose.connect().

Лістинг 3.30 – Підключення до бази даних

```
const mongoose = require("mongoose");
const connect = async ()=>{
     return await mongoose.connect("...");
}
module.exports = connect;
```

У параметрах цього методу вказано повний URI для з’єднання з кластером MongoDB Atlas. Цей рядок містить ім’я користувача, пароль, назву кластера та бази даних (sport-events), а також додаткові параметри підключення, як-от retryWrites=true для автоматичного повторного надсилання записів у разі помилок та w=majority для підтвердження запису більшістю реплік. Також вказано ім’я кластера (appName=Cluster0), що дозволяє коректно ідентифікувати з’єднання. Результат виконання функції повертається за допомогою return await, що гарантує повернення завершеного підключення або помилки, якщо з’єднання не вдалося встановити. Наприкінці файл експортує функцію connect, щоб її можна було використати в інших частинах застосунку, зокрема у головному файлі сервера (index.js) [36].

Далі перейдемо до файлу event.model.js, який відповідає за створення схеми та моделі події (event) у базі даних MongoDB з використанням бібліотеки Mongoose.

Спочатку імпортуємо з бібліотеки Mongoose два ключові об’єкти: Schema – для створення схеми документа, та model – для створення моделі, яка дозволяє взаємодіяти з колекцією у базі даних.

Далі створюємо нову схему EventSchema, яка описує структуру документа події. Поле desc містить опис події, date – дату проведення, venue – місце події,

playersLimit – максимальну кількість гравців. Поле type вказує тип події, наприклад, “футбол” або “баскетбол”. org_id – це ідентифікатор організатора, а org_name – його ім’я. players – масив, що може містити список гравців, які зареєструвалися на подію (див. лістинг 3.31).

Лістинг 3.31 – Створення схеми EventSchema

```
const EventSchema = new Schema({
     desc: {type: String, required: true},
     ...
     org_name: {type: String, required: true},
     players: {
          type: Array
     }
}, {
     timestamps: true,
     versionKey: false,
});
```

Опції { timestamps: true, versionKey: false } додають автоматичні поля createdAt і updatedAt, але вимикають поле __v, яке зазвичай додає Mongoose для відстеження версій документа.

На основі створеної схеми EventSchema за допомогою функції model створюється модель EventModel. Перший аргумент "event" визначає ім’я колекції в базі даних (воно буде автоматично змінено на множину – events), а другий – передає саму схему.

```
const EventModel = model("event", EventSchema);
```

Далі розберемо user.model.js. Він відповідає за створення схеми та моделі користувача (user) у базі даних MongoDB з використанням бібліотеки Mongoose.

Спочатку імпортуємо Schema та model з бібліотеки Mongoose, які необхідні для створення схеми та моделі користувача. Також імпортуємо бібліотеку validator, яка використовується для валідації значень, зокрема електронної пошти.

Створюємо UserSchema, яка визначає структуру об’єкта користувача в базі даних. Поле name є обов’язковим і зберігає ім’я користувача. Поле email також

обов’язкове, має бути унікальним і проходити перевірку за допомогою методу validator.isEmail, що гарантує правильний формат електронної адреси. Якщо валідація не проходить, буде виведено відповідне повідомлення про помилку.

Поле password містить обов’язковий пароль користувача у вигляді текстового рядка.

Поле role визначає тип користувача, який може бути або звичайним користувачем ("user"), або організатором подій ("organizer"). Якщо роль не вказана, за замовчуванням встановлюється "user". Це реалізовано за допомогою обмеження enum, яке дозволяє лише конкретні значення (див. лістинг 3.32).

Лістинг 3.32 – Створення схеми користувача

```
const UserSchema = new Schema({
     name: {type: String, required: true},
     email: {
          ...
          validate:{
               validator: validator.isEmail,
               message: '{VALUE} is not a valid email',
               isAsync: false
     ...
     timestamps: true,
     versionKey: false,
})
```

Опції схеми { timestamps: true, versionKey: false } додають автоматичні поля createdAt і updatedAt, а також вимикають поле __v, яке зазвичай використовується для внутрішньої версії документа.

І наприкінці створюємо модель UserModel на основі створеної схеми UserSchema. Назва "user" буде використана для формування колекції у базі даних (MongoDB автоматично зробить її множиною – users).

```
const UserModel = model("user", UserSchema);
```

Наступний на черзі event.route.js. Він відповідає за обробку запитів, пов’язаних із подіями (events), за допомогою Express.js. На початку файлу здійснюється імпорт бібліотеки express, а також моделі EventModel, яка

використовується для взаємодії з колекцією подій у базі даних MongoDB. Далі створюється екземпляр маршрутизатора Express – app = express.Router(), що дозволяє групувати маршрути, пов’язані з подіями, в одному файлі [37].

```
const express = require("express");
const EventModel = require("../model/event.model");
const app = express.Router();
```

Першим визначається маршрут GET "/", який повертає список усіх подій. За допомогою методу EventModel.find() отримуються всі документи з колекції подій, а .sort({createdAt: -1}) сортує їх у порядку спадання за датою створення – тобто спочатку новіші. Результат надсилається клієнту у вигляді JSON-відповіді.

Наступний маршрут – POST "/" – дозволяє створити нову подію. З тіла запиту (req.body) витягуються всі необхідні поля: опис, дата, місце проведення, ліміт гравців, тип події, ідентифікатор та ім’я організатора. Додатково створюється порожній масив players, який згодом буде містити користувачів, що долучилися до події. У блоці try...catch створюється новий екземпляр моделі події, зберігається у базі даних методом save(), після чого клієнту повертається відповідь з інформацією про щойно створену подію. У випадку помилки повертається статус 500 і повідомлення про помилку (див. лістинг 3.33).

Лістинг 3.33 – POST запит на маршрут "/"

```
app.post("/", async(req,res)=> {
     let {desc, date, venue, playersLimit, type, org_id, org_name}
     = req.body;
     let players = [];
     try{
          let event = new EventModel({desc, date, venue,
          playersLimit, type, org_id, org_name, players});
          await event.save();
          ...
```

Маршрут PUT "/:id" реалізує додавання гравця до вже існуючої події. Із параметрів маршруту зчитується id події, а з тіла запиту – об’єкт гравця. Далі подія шукається в базі даних за допомогою findById(). Якщо подію знайдено, до масиву players додається новий гравець, і після збереження змін оновлений

документ повертається у відповіді. Якщо подію не знайдено, повертається статус 404. У разі помилки при запиті повертається статус 500 (див. лістинг 3.34).

Лістинг 3.34 – PUT запит на маршрут "/:id"

```
app.put('/:id', async (req, res) => {
     const eventId = req.params.id;
     const player = req.body.player;
     try {
          const event = await EventModel.findById(eventId);
          ...
          event.players.push(player);
          ...
```

Останній маршрут – GET "/search" – реалізує функцію пошуку подій за ключовим словом. Ключове слово береться з рядка запиту (req.query.keyword) і використовується для пошуку по кількох полях документа: desc, venue, type та org_name. Для кожного поля використовується регулярний вираз ($regex) з опцією i, яка вказує на нечутливість до регістру. Результати пошуку повертаються клієнту у форматі JSON. Якщо виникає помилка під час пошуку, надсилається відповідь з повідомленням про помилку та статусом 500 (див. лістинг 3.35).

Лістинг 3.35 – GET запит на маршрут "/search"

```
app.get('/search', async (req, res) => {
     const keyword = req.query.keyword;
     try {
          const results = await EventModel.find({
               $or: [
               { desc: { $regex: keyword, $options: 'i' } },
          ...
});
```

У самому кінці файл експортує створений маршрутизатор app, щоб він міг бути підключений у головному файлі серверу (index.js) і використовуватися для обробки запитів, пов’язаних із подіями.

Останнім файлом розберемо user.route.js. Він реалізує маршрути для управління користувачами (реєстрація, вхід, оновлення токенів) за допомогою бібліотек Express, JWT (JSON Web Token) та Argon2 для хешування паролів.

Першим кроком відбувається імпорт необхідних бібліотек: express для створення маршрутизаторів, jsonwebtoken для генерації та верифікації токенів, argon2 для хешування паролів, а також моделі користувача UserModel, яка дозволяє працювати з документами користувачів у базі даних MongoDB [38].

Маршрут GET "/" обробляє запит на отримання всіх користувачів з бази даних. За допомогою методу UserModel.find() витягуються всі користувачі, і результат відправляється назад клієнту у вигляді JSON (див. лістинг 3.36).

Лістинг 3.36 – GET запит на маршрут "/"

```
app.get("/", async (req,res)=> {
    const users = await UserModel.find();
    return res.send(users)
})
```

Маршрут POST "/signup" відповідає за реєстрацію нових користувачів. З тіла запиту отримуються ім’я, email, пароль і роль користувача. Пароль хешується за допомогою argon2.hash(), щоб забезпечити безпеку. Після цього перевіряється, чи вже існує користувач з таким email в базі даних. Якщо користувач знайдений, повертається статус 409 із повідомленням про конфлікт. Якщо користувач не знайдений, створюється новий об’єкт користувача, який зберігається в базі даних за допомогою методу save(). Після цього повертається статус 201 та дані нового користувача. У випадку помилки повертається статус 500 і повідомлення про помилку (див. лістинг 3.37).

Лістинг 3.37 – POST запит на маршрут "/signup"

```
app.post("/signup", async(req,res)=> {
     let {name, email, password, role} = req.body;
     const hash = await argon2.hash(password);
     let user = await UserModel.findOne({email});
     try{
          ...
          let newUser = new UserModel({name, email, password:
          hash, role});
          await newUser.save();
```

Маршрут POST "/login" обробляє вхід користувача. Користувач вводить email і пароль, які порівнюються з інформацією в базі даних. Якщо користувача з таким email знайдено і пароль вірний (перевірка відбувається за допомогою argon2.verify()), генеруються два токени:

1. Access token – короткочасний токен, що дає доступ до ресурсів на сервері. Термін дії – 7 днів.

2. Refresh token – довготривалий токен, який використовується для оновлення access token. Термін дії – 28 днів.

Ці токени створюються за допомогою методу jwt.sign() і передаються клієнту у відповіді разом з повідомленням про успішний вхід [39]. Якщо пароль або email невірні, повертається статус 401 і повідомлення “invalid credentials” (див. лістинг 3.38).

Лістинг 3.38 – POST запит на маршрут "/login"

```
app.post("/login", async(req,res)=> {
     const {email, password} = req.body;
     const user = await UserModel.findOne({email});
     if(await argon2.verify(user.password, password)){
          const token = jwt.sign(
               {id: user._id, name: user.name, email: user.email,
               role: user.role},
               secrectToken,
               {expiresIn: "7 days"}
```

Маршрут POST "/refresh" відповідає за оновлення access token за допомогою refresh token. Токен передається через заголовок Authorization. Якщо refresh token відсутній, повертається статус 401 і повідомлення “login again”. Якщо токен є, він перевіряється за допомогою jwt.verify(). Якщо верифікація успішна, генерується новий access token, який надсилається клієнту. Якщо refresh token недійсний або верифікація не вдалася, повертається статус 401 і повідомлення “unauthorized” (див. лістинг 3.39).

Лістинг 3.39 – POST запит на маршрут "/refresh"

```
app.post("/refresh", async(req,res)=> {
    const refreshT = req.headers["authorization"];
```

```
    ...
    try{
        const verification = jwt.verify(refreshT, refreshToken);
        if(verification){
            const newToken = jwt.sign(
                {id: verification.id, name: verification.name,
email: verification.email, role: verification.role},
```

У кінці файл експортує маршрутизатор app, щоб інші частини програми могли використовувати ці маршрути для обробки запитів, пов'язаних з користувачами [40].

3.3 Тестування веб-сайту

При запуску веб-сайту можна побачити головну сторінку (див. Додаток А) з пошуком по подіях, і зверху присутня кнопка для відкриття вікна для реєстрації чи входу в акаунт (див. рисунок 3.1).

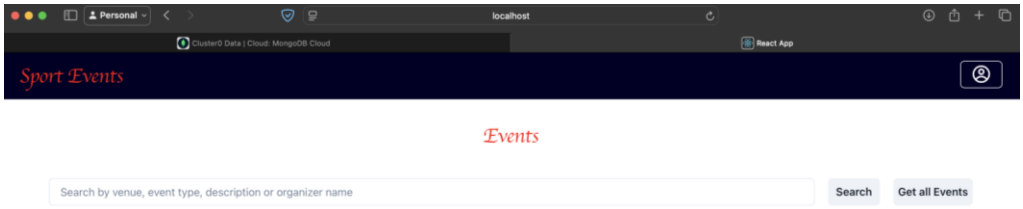

Рисунок 3.1 – Головна сторінка

У результаті виконання дій відкриваємо спливаюче вікно для реєстрації (див. додаток Б) і створимо акаунт адміністратора (див. рисунок 3.2).

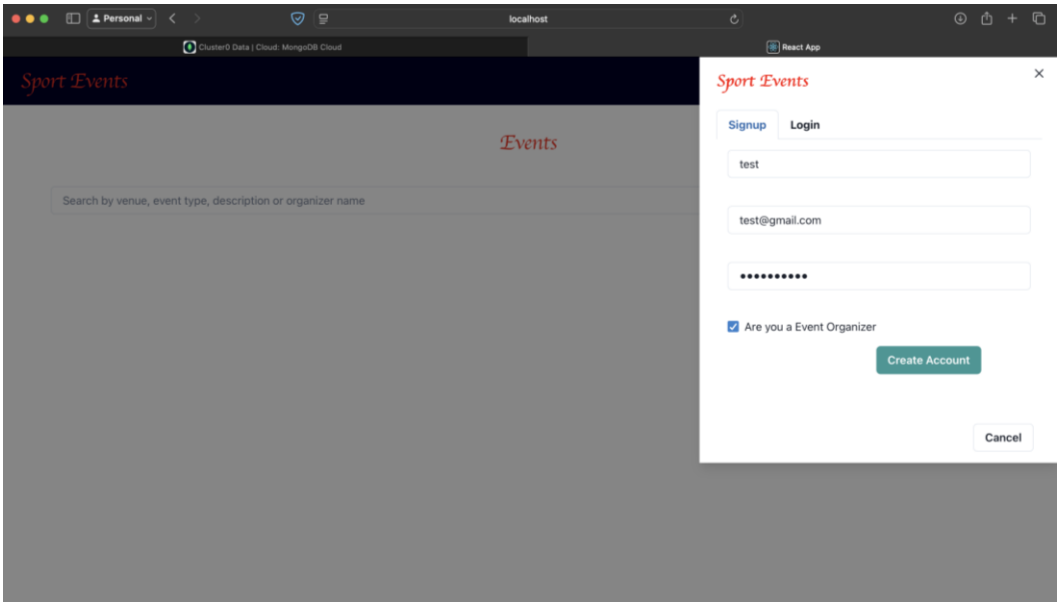

Рисунок 3.2 – Створення акаунту

Далі можна перейти на іншу вкладку з входом і зайти в створений акаунт (див. рисунок 3.3).

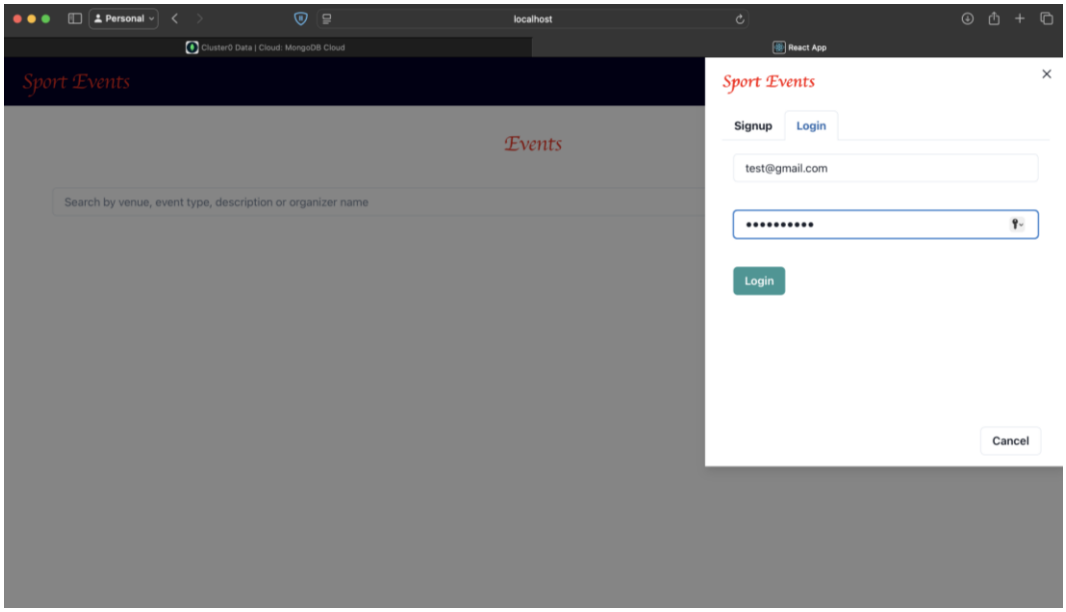

Рисунок 3.3 – Вхід в акаунт

Після цього створюємо спортивну подію (див. рисунок 3.4).

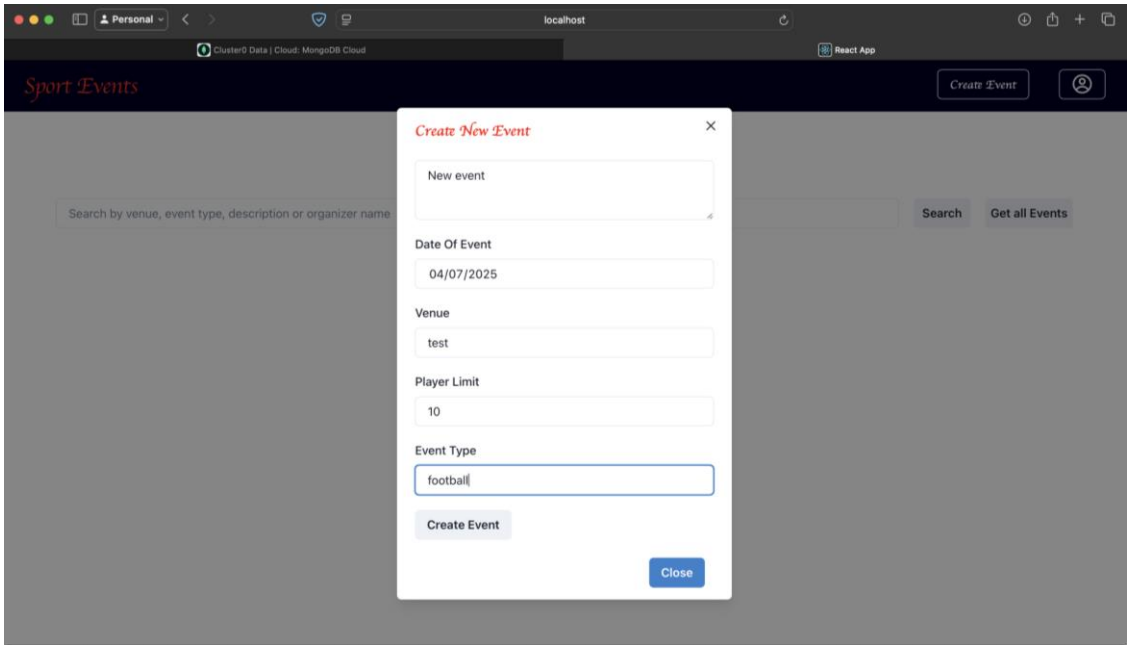

Рисунок 3.4 – Створення події

І після створення він з’явиться в списку подій (див. рисунок 3.5).

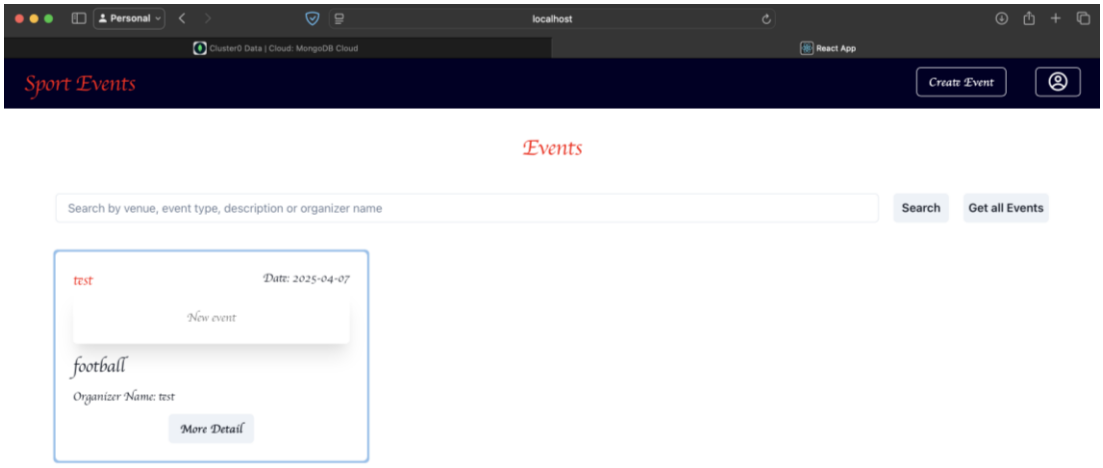

Рисунок 3.5 – Створена подія

Далі можна переглянути інформацію про подію натиснувши на відповідну кнопку (див. рисунок 3.6).

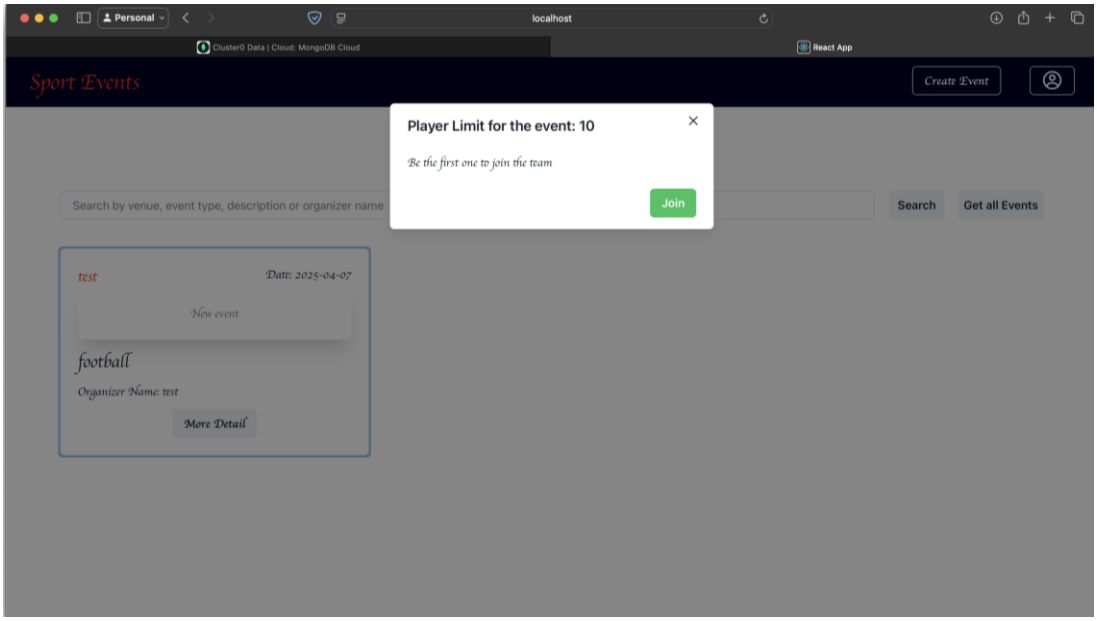

Рисунок 3.6 – Створена подія

Далі створимо ще один акаунт учасника і приєднаємось до даної події і відкривши інформацію про неї можемо побачити що додався один учасник (див. рисунок 3.7).

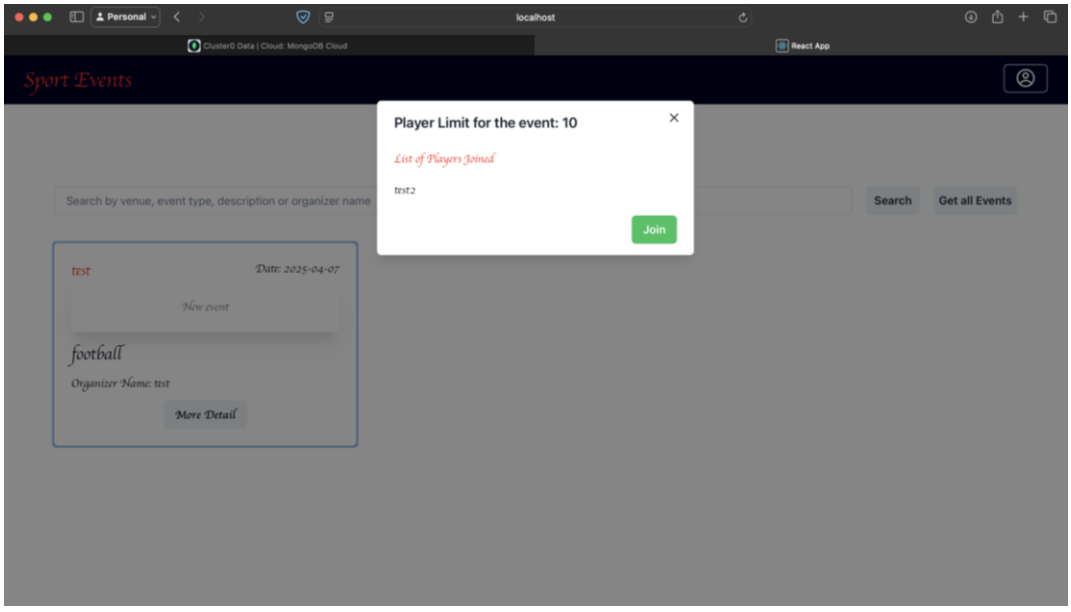

Рисунок 3.7 – Доданий учасник до події

Можна додати ще одну подію для того щоб протестувати пошук (див. рисунок 3.8).

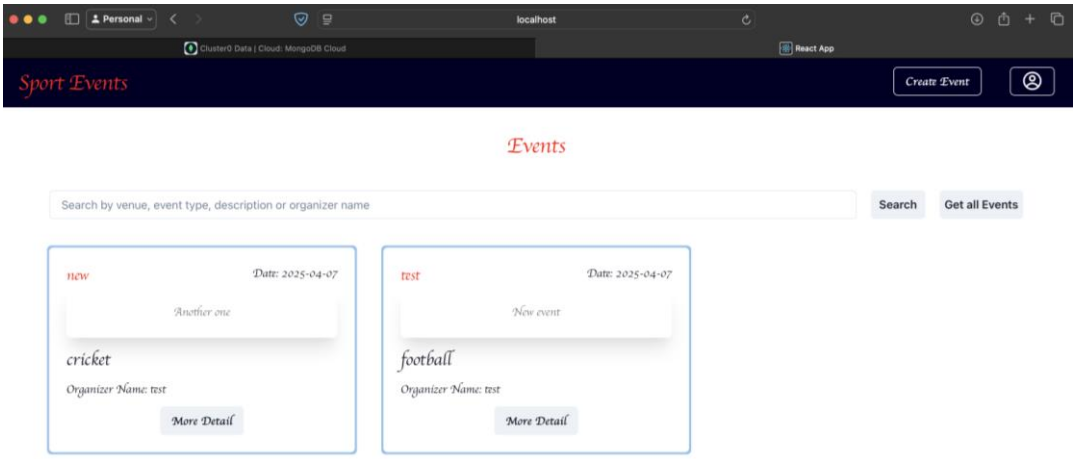

Рисунок 3.8 – Створена ще одна подія

Виконуємо пошук по категорії “football” (див. рисунок 3.9).

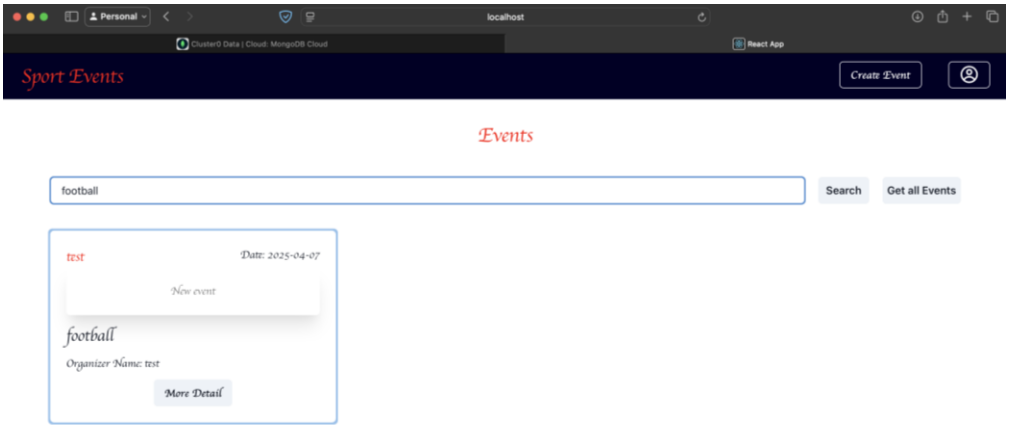

Рисунок 3.9 – Пошук по категорії “football”

Після виконання пошуку на сторінці відображається список подій.

3.4 Висновок до третього розділу

У третьому розділі дипломної роботи здійснено безпосередню реалізацію веб-сайту для огляду спортивних подій, що передбачало розробку як клієнтської

(фронтенд), так і серверної (бекенд) частини, а також тестування функціональності всієї системи.

На етапі реалізації фронтенд-частини було використано бібліотеку React.js, що забезпечило побудову модульної архітектури користувацького інтерфейсу. Компонентний підхід дав змогу легко управляти окремими елементами інтерфейсу, такими як форма реєстрації/авторизації, список подій, пошук та перегляд деталей подій. Була реалізована взаємодія з бекенд-сервером за допомогою HTTP-запитів (через fetch або axios), що дало можливість динамічно відображати дані в реальному часі. Також було забезпечено базову клієнтську валідацію введених даних, що покращило зручність користування сайтом.

На етапі реалізації бекенд-частини проєкту використовувались Node.js та фреймворк Express.js, які дозволили створити серверну логіку та REST API для обробки запитів з фронтенду. Було реалізовано такі маршрути: отримання списку подій, додавання нової події, оновлення даних (наприклад, додавання учасника до події), пошук подій за ключовими словами. Для зберігання інформації використовувалась база даних MongoDB, а для моделювання даних – ODM-бібліотека Mongoose, за допомогою якої були створені схеми користувачів та подій.

Особливу увагу було приділено реалізації аутентифікації та безпеки. Для реєстрації користувачів застосовано хешування паролів за допомогою бібліотеки argon2, що забезпечує високий рівень безпеки збереження облікових даних. Для авторизації користувачів використовувались JWT-токени (JSON Web Token), які дають змогу реалізувати захищений доступ до окремих функціональних частин сайту, зокрема для організаторів спортивних подій.

Також було реалізовано функціонал пошуку, який дозволяє користувачам знаходити події за ключовими словами в описі, назві організатора, місці проведення або типу події. Це значно підвищує зручність використання ресурсу, особливо при наявності великої кількості інформації.

На завершальному етапі було проведено тестування функціональності сайту. Перевірялась коректність відображення подій, реєстрації нових користувачів, додавання нових подій, пошуку та оновлення інформації. Було

перевірено обробку помилок на фронтенді й бекенді, правильність роботи з базою даних, валідність введених даних та коректність відповідей API. Тестування показало стабільну роботу веб-сайту та відповідність функціоналу поставленим завданням.

У підсумку, на основі сучасних технологій було створено повноцінний веб-сайт для огляду спортивних подій, який дозволяє користувачам переглядати події, реєструватися, додавати нові події (для організаторів), а також здійснювати пошук та взаємодіяти з інформацією у зручному форматі. Отриманий результат є підтвердженням доцільності обраної архітектури та технологічного стеку для розв’язання поставленої задачі.

РОЗДІЛ 4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Ергономічні проблеми безпеки життєдіяльності

У сучасних умовах розвитку інформаційних технологій та масової комп'ютеризації значна частина робочих процесів здійснюється за допомогою комп'ютерної техніки. Особливо це стосується діяльності спеціалістів з розробки програмного забезпечення та веб-додатків. Веб-розробники більшу частину робочого часу проводять за комп'ютером, виконуючи різні завдання: проектування архітектури програм, написання коду, тестування, адміністрування серверів тощо. Такий характер праці має низку особливостей, які пов'язані з тривалим перебуванням у вимушеній статичній позі, інтенсивною зоровою напругою, психоемоційними навантаженнями та впливом факторів виробничого середовища.

Ергономіка, як наука про оптимізацію взаємодії людини з технічними засобами та навколишнім середовищем, вивчає умови організації робочих місць і режимів праці з метою забезпечення безпеки, комфорту і продуктивності працівників. Неврахування ергономічних вимог при організації праці розробника може призводити до виникнення низки проблем, які негативно впливають на працездатність, самопочуття та загальний стан здоров'я [41].

Дві найпоширеніші ергономічні проблеми під час роботи за комп'ютером є надмірне навантаження на зоровий апарат та статичне навантаження на опорно руховий апарат. Під час тривалої роботи за монітором людина змушена постійно фокусувати погляд на екрані, що викликає зорову втому, сльозотечу, сухість очей, подразнення слизової оболонки. Згодом це може призводити до зниження гостроти зору, виникнення головного болю, порушення акомодації та розвитку офтальмологічних захворювань. Важливими факторами, які поглиблюють ці проблеми, є низький рівень освітленості приміщення, неправильне розташування монітора, надмірна яскравість екрану або його недостатній контраст.

Ще однією серйозною проблемою є статичне навантаження на опорно-руховий апарат. Розробники здебільшого працюють у сидячому положенні, що призводить до перенавантаження м’язів спини, шиї, попереку, а також до порушення кровообігу в нижніх кінцівках. За відсутності регулярних фізичних вправ та перерв у роботі це може спричинити розвиток остеохондрозу, сколіозу, артрозів, тунельного синдрому та інших захворювань опорно-рухового апарату. Надмірне навантаження на кисті рук та зап’ястя внаслідок роботи з клавіатурою та мишкою призводить до виникнення болю, оніміння та дискомфорту в суглобах [42].

Окремої уваги потребують психоемоційні навантаження, які виникають у процесі розробки програмного забезпечення. Висока інтелектуальна напруга, необхідність прийняття швидких і точних рішень, робота в умовах обмежених термінів, потреба в обробці великих обсягів інформації створюють підвищене емоційне навантаження на працівника. Постійне нервове напруження може призводити до перевтоми, стресу, зниження концентрації уваги, погіршення настрою, розвитку тривожності та професійного вигорання.

Також на безпеку життєдіяльності впливають несприятливі фактори мікроклімату приміщення: підвищена або знижена температура повітря, низька відносна вологість, недостатнє або надмірне освітлення, погана вентиляція. Тривале перебування у таких умовах може викликати загальне погіршення самопочуття, зниження працездатності, швидку стомлюваність, виникнення захворювань дихальної системи та алергічних реакцій.

З метою забезпечення ергономічної безпеки працівників, які виконують роботу за комп’ютером, необхідно створювати оптимальні умови праці. Це передбачає організацію правильного розташування робочого місця, регулювання висоти столу та стільця, забезпечення достатньої освітленості робочої зони, розміщення монітора на оптимальній відстані та висоті від очей, застосування підставок для рук, а також впровадження режиму праці та відпочинку, що передбачає регулярні фізкультурні паузи, гімнастику для очей і зміну виду діяльності. Всі ці заходи дозволяють знизити негативний вплив несприятливих

факторів, зберегти здоров’я працівника, підвищити продуктивність праці та поліпшити якість виконання професійних обов’язків.

4.2 Естетичне оформлення робочого місця оператора ПК

Естетичне оформлення робочого місця є важливим аспектом створення комфортних умов праці, що впливає не лише на продуктивність працівника, але й на його психоемоційний стан, мотивацію та загальне самопочуття. Особливо це актуально для фахівців ІТ-сфери, зокрема розробників веб-додатків, які тривалий час проводять за комп’ютером, виконуючи завдання, що потребують високої концентрації уваги, точності та інтелектуальних зусиль [43].

Затишне та гармонійне середовище сприяє підвищенню ефективності, зниженню рівня стресу та створенню приємної атмосфери для щоденних завдань. Крім того, правильне оформлення робочого місця допомагає знизити ризики розвитку професійних захворювань і перенапруги. Оформлення робочого простору оператора ПК є комплексним завданням, яке охоплює низку важливих складових.

Насамперед важливою є ергономіка – зручне та безпечне розміщення обладнання. Відповідно до ДСТУ 8604:2015, висота столу має становити 680–800 мм, а стілець має бути регульованим у межах 380–500 мм. Монітор розміщують на відстані 600–700 мм від очей, при цьому його верхній край має бути на рівні або трохи нижче рівня очей, з нахилом приблизно 15–20 градусів. Використання якісного стільця з м’яким сидінням, спинкою та підлокітниками допомагає зменшити фізичне навантаження на опорно-руховий апарат.

Освітлення також є критичним чинником комфорту. Робоче місце повинно бути добре освітленим з поєднанням природного світла та штучного освітлення. Яскравість освітлення має становити не менше 300–500 люкс. Джерела світла необхідно розташовувати таким чином, щоб уникнути відблисків на екрані монітора та створення різких тіней на робочій поверхні. Важливою є можливість регулювання інтенсивності світла відповідно до часу доби або характеру завдань.

Колірна гама робочого місця значною мірою впливає на психоемоційний стан працівника. Теплі кольори, як-от жовтий чи світло-оранжевий, сприяють активності та покращують настрій, тоді як холодні відтінки, наприклад блакитний або зелений, мають заспокійливий ефект. Надмірно яскраві або дуже темні кольори можуть викликати втому та дратівливість, тому оптимальним є використання пастельних тонів для стін і меблів із додаванням помірних кольорових акцентів для створення візуального балансу.

Порядок і організованість робочого простору також мають велике значення. Надлишок предметів, безлад і хаотичне розміщення документів або техніки ускладнюють концентрацію уваги, спричиняють втому та підвищують рівень стресу. Ідеальне робоче місце повинно бути організованим за принципом мінімалізму – на поверхні столу розміщують лише найнеобхідніше, тоді як інші предмети слід зберігати в ящиках, на полицях або в шафах. Для підтримання порядку корисним є використання органайзерів, підставок, стелажів.

Декоративні елементи – ще один важливий аспект. Наявність кімнатних рослин, картин, постерів або сувенірів створює відчуття затишку, підвищує емоційний комфорт, а також сприяє зниженню рівня стресу. Рослини, зокрема, очищують повітря та покращують мікроклімат. Проте кількість декоративних елементів повинна бути помірною, аби не спричиняти візуального перевантаження.

Сучасні технологічні інновації дозволяють зробити робоче місце більш функціональним. Бездротові пристрої зменшують кількість дротів і сприяють кращій організації простору. Інтелектуальні системи освітлення та клімат-контролю дають змогу автоматично регулювати умови відповідно до потреб працівника, що підвищує комфортність робочого процесу [44].

З точки зору оформлення робочого місця оператора ПК є важливою складовою безпеки життєдіяльності, оскільки безпосередньо впливає на працездатність, емоційний стан та загальне самопочуття працівника. Розумне поєднання кольорів, освітлення, організації простору, декоративних елементів та якісного меблювання створює сприятливу робочу атмосферу, яка сприяє

ефективному виконанню професійних обов’язків і зниженню ризиків виникнення професійних захворювань та психоемоційних перевантажень.

4.3 Висновок до четвертого розділу

Розділ, присвячений безпеці життєдіяльності та основам охорони праці, розглядається як комплексний у своїй сутності та значущості. На основі аналізу двох ключових аспектів – ризику як кількісної оцінки небезпек та естетичного оформлення робочого місця оператора ПК – проаналізуємо фундаментальні аспекти забезпечення безпеки та комфорту працівника.

Проаналізовано естетичне оформлення робочого місця оператора ПК, як ключовий чинник, що впливає на комфорт та продуктивність працівника. Було детально розглянуто фактори ергономіки, відповідно до ДСТУ 8604:2015, включаючи висоту робочої поверхні стола (680-800 мм) та регульованність стільця (380-500 мм), розташування монітора (на відстані 600-700 мм від очей) та інші параметри. Зазначено важливість належного освітлення – не менше 300-500 люкс, підбору колірної гами, яка сприяє спокою та концентрації.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи освітнього рівня “Бакалавр” на тему “Розробка інформаційного веб-сайту на тематику огляду спортивних подій” було досягнуто поставленої мети – створено сучасний, функціональний веб-сайт для відображення та перегляду оглядів спортивних подій, що відповідає сучасним вимогам до інформаційних платформ.

У першому розділі роботи здійснено аналіз предметної області. Було розглянуто існуючі популярні веб-сайти, присвячені спортивним подіям, визначено їх основні функціональні можливості, переваги та недоліки. Проведено огляд сучасних веб-технологій, що застосовуються для створення подібних проєктів, серед яких особливу увагу приділено React.js, Node.js, Express.js, що стали основою для реалізації власного проєкту. Також було проаналізовано архітектурні підходи до побудови веб-додатків, що дозволило обґрунтувати вибір архітектури клієнт-серверної моделі.

У другому розділі виконано проєктування архітектури майбутнього веб-сайту. Визначено структурну схему взаємодії між клієнтською та серверною частинами, розроблено модель бази даних, яка передбачає зберігання інформації про спортивні події, огляди, авторів та категорії оглядів. Було запропоновано архітектурний підхід на основі клієнт-серверної моделі з REST API, що дозволяє забезпечити гнучкість у масштабуванні та спрощення подальшої підтримки системи. Особливу увагу приділено безпеці взаємодії між клієнтом і сервером, правильній організації обробки HTTP-запитів та захисту даних.

У третьому розділі безпосередньо реалізовано веб-сайт для огляду спортивних подій. Фронтенд-частину створено з використанням бібліотеки React.js, що забезпечила адаптивність, інтерактивність та високу швидкість роботи користувацького інтерфейсу. Для реалізації бекенд-частини застосовано Node.js у поєднанні з фреймворком Express.js, що дозволило забезпечити швидке та стабільне обслуговування клієнтських запитів, а також обробку даних на сервері. Проведено тестування веб-сайту, що підтвердило коректну роботу усіх

основних функцій: відображення спортивних оглядів, пошуку, фільтрації за категоріями, додавання нових матеріалів та їх редагування.

У четвертому розділі було проаналізовано питання безпеки життєдіяльності та охорони праці під час роботи оператора персонального комп’ютера. Окрему увагу приділено ергономічним проблемам, які виникають при тривалій роботі за ПК, та запропоновано заходи щодо їх усунення. Також розглянуто принципи естетичного оформлення робочого місця, що сприяють покращенню умов праці та підвищенню продуктивності працівника.

У підсумку, виконана кваліфікаційна робота бакалавра має практичну цінність, оскільки розроблений веб-сайт може бути використаний як готовий проєкт для інформаційної підтримки спортивних подій або слугувати основою для подальшої розробки повнофункціональних спортивних порталів. Вибір сучасних технологій React.js, Node.js та Express.js забезпечив високу продуктивність, масштабованість та зручність подальшої підтримки розробленого веб-додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ESPN [Електронний ресурс]. – Режим доступу: https://www.espn.com/ (дата звернення: 08.05.2025).

2. BBC Sport [Електронний ресурс]. – Режим доступу: https://www.bbc.com/sport (дата звернення: 08.05.2025).

3. Sky Sports [Електронний ресурс]. – Режим доступу: https://www.skysports.com/ (дата звернення: 08.05.2025).

4. CBS Sports [Електронний ресурс]. – Режим доступу: https://www.cbssports.com/ (дата звернення: 08.05.2025).

5. Yahoo Sports [Електронний ресурс]. – Режим доступу: https://sports.yahoo.com/ (дата звернення: 08.05.2025).

6. Bleacher Report [Електронний ресурс]. – Режим доступу: https://bleacherreport.com/ (дата звернення: 08.05.2025).

7. The Athletic [Електронний ресурс]. – Режим доступу: https://theathletic.com/ (дата звернення: 08.05.2025).

8. Goal [Електронний ресурс]. – Режим доступу: https://www.goal.com/ (дата звернення: 08.05.2025).

9. Creating a React, Node, and Express App [Електронний ресурс]. – Режим доступу: https://dev.to/techcheck/creating-a-react-node-and-express-app-1ieg (дата звернення: 08.05.2025).

10. Create a React app served by Express.js & Node.js (and add TypeScript) [Електронний ресурс]. – Режим доступу: https://leejjon.medium.com/create-a-react-app-served-by-express-js-node-js-and-add-typescript-33705be3ceda (дата звернення: 08.05.2025).

11. How to create a React frontend and a Node/Express backend and connect them together [Електронний ресурс]. – Режим доступу: https://www.freecodecamp.org/news/create-a-react-frontend-a-node-express-backend-and-connect-them-together-c5798926057c/ (дата звернення: 08.05.2025).

12. Full-Stack Web App Using React and Express [Електронний ресурс]. – Режим доступу: https://github.com/Techtonica/curriculum/blob/main/pair-

programming/week-7/react-express-app/react-expressjs.md (дата звернення: 08.05.2025).

13. React Express Example [Електронний ресурс]. – Режим доступу: https://github.com/nburgess/react-express-example (дата звернення: 08.05.2025).

14. React Express [Електронний ресурс]. – Режим доступу: https://www.react.express/ (дата звернення: 08.05.2025).

15. React and Express.js tutorial [Електронний ресурс]. – Режим доступу: https://docs.aserto.com/docs/quickstarts/react/overview (дата звернення: 08.05.2025).

16. How to Connect Express with React: A Comprehensive Guide [Електронний ресурс]. – Режим доступу: https://www.linkedin.com/pulse/how-connect-express-react-comprehensive-guide-eslam-zaid (дата звернення: 08.05.2025).

17. Full Stack Open [Електронний ресурс]. – Режим доступу: https://fullstackopen.com/en/ (дата звернення: 08.05.2025).

18. React + Express Full Stack Example [Електронний ресурс]. – Режим доступу: https://github.com/crsandeep/simple-react-full-stack (дата звернення: 08.05.2025).

19. How to Build a MERN Stack App [Електронний ресурс]. – Режим доступу: https://medium.com/@sriram.se21/step-by-step-tutorial-building-a-mern-stack-application-from-scratch-d281010715e4 (дата звернення: 08.05.2025).

20. Deploying React with Express on Render [Електронний ресурс]. – Режим доступу: https://render.com/docs/deploy-react-express-app (дата звернення: 08.05.2025).

21. React Express Example [Електронний ресурс]. – Режим доступу: https://github.com/nburgess/react-express-example (дата звернення: 08.05.2025).

22. Vite React Express Boilerplate [Електронний ресурс]. – Режим доступу: https://github.com/joeynguyen/vite-react-express-boilerplate (дата звернення: 08.05.2025).

23. Full-Stack Web Development Project with React and Express [Електронний ресурс]. – Режим доступу:

https://medium.com/@sushantkadam15/setting-up-a-full-stack-web-development-project-with-react-and-express-e835c52e3c31 (дата звернення: 08.05.2025).

24. React Express Starter [Електронний ресурс]. – Режим доступу: https://github.com/philnash/react-express-starter (дата звернення: 08.05.2025).

25. Kickstarting Your Full-Stack Journey with React, Vite, and Express [Електронний ресурс]. – Режим доступу: https://www.joelspriggs.com/kickstarting-your-full-stack-journey-with-react-vite-and-express/ (дата звернення: 08.05.2025).

26. React Express Boilerplate for SaaS [Електронний ресурс]. – Режим доступу: https://www.reddit.com/r/node/comments/p2uivn/i_am_building_a_nodejs_express_react_boilerplate/ (дата звернення: 08.05.2025).

27. React and Express.js Tutorial [Електронний ресурс]. – Режим доступу: https://docs.aserto.com/docs/quickstarts/react/overview (дата звернення: 08.05.2025).

28. Understanding React and Express: A Comprehensive Guide [Електронний ресурс]. – Режим доступу: https://www.simplilearn.com/tutorials/react-tutorial/guide-to-understanding-react-and-express (дата звернення: 08.05.2025).

29. How to Use React with Express [Електронний ресурс]. – Режим доступу: https://medium.com/@eslmzadpc13/how-to-use-react-with-express-5f832dc74105 (дата звернення: 08.05.2025).

30. React Express [Електронний ресурс]. – Режим доступу: https://www.react.express/ (дата звернення: 08.05.2025).

31. Building a Full-Stack Notes Application with Node.js, Express, React, and Tailwind CSS [Електронний ресурс]. – Режим доступу: https://dev.to/moibra/building-a-full-stack-notes-application-with-nodejs-express-react-and-tailwind-css-2642 (дата звернення: 08.05.2025).

32. Building Full-Stack Applications with TypeScript, Node.js, and React.js [Електронний ресурс]. – Режим доступу: https://medium.com/@vishalchouhan2012/building-full-stack-applications-with-typescript-node-js-and-react-js-32105310b244 (дата звернення: 08.05.2025).

33. Building a Full Stack App with React 16 and Express 4 [Електронний ресурс]. – Режим доступу: https://www.pluralsight.com/courses/react-express-full-stack-app-building (дата звернення: 08.05.2025).

34. Recipes for Express & React Full-Stack Apps in TypeScript [Електронний ресурс]. – Режим доступу: https://javascript.plainenglish.io/recipes-for-express-and-react-fullstack-apps-in-type-8bcd4644bed0 (дата звернення: 08.05.2025).

35. Node.js and Express [Електронний ресурс]. – Режим доступу: https://fullstackopen.com/en/part3/node_js_and_express (дата звернення: 08.05.2025).

36. Harchenko, A., Bodnarchuk, I., & Yatcyshyn, V. (2012). The modeling and optimization of software engineering processes. Proceedings of International Conference on Modern Problem of Radio Engineering, Telecommunications and Computer Science, 326.

37. Харченко, О., Яцишин, В., & Боднарчук, І. (2013). Експертна система проектування архітектури програмного забезпечення. Комп'ютерні Технології Друкарства, (29), 10–26.

38. Lytvynenko, I. V., Maruschak, P. O., & Lupenko, S. A. (2014). Processing and modeling of ordered relief at the surface of heat-resistant steels after laser irradiation as a cyclic random process. Automatic Control and Computer Sciences, 48, 1-9.

39. Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022). Information technology for estimating city gas consumption during the year. International Conference on Smart Information Systems and Technologies.

40. Fedonuyk, A., Yunchyk, V., Mukutuyk, I., Duda, O., & Yatsyuk, S. (2021). Application of the hierarchy analysis method for the choice of the computer mathematics system for the IT-sphere specialists preparation.

41. Безпека життєдіяльності та охорона праці : підруч. / В.В. Сокуренко, О.М. Бандурка та ін. – Харків : ХНУВС, 2021. 308 с.

42. Атаманчук П.С. Безпека життєдіяльності: навч. посіб. Київ : Центр учбової літератури, 2020. 276 с.

43. Організація робочого місця Оператора ПК [Електронний ресурс] – Режим доступу: https://studcon.org/organizaciya-robochogo-miscya-operatorapk?page=5 (дата звернення 08.05.2025).

44. ДСТУ 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги [Електронний ресурс] – Режим доступу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=71028 (дата звернення: 08.05.2025).

ДОДАТКИ

Додаток А

Програмний код головної сторінки веб-сайту «Home.jsx»

```
import React, { useContext, useEffect, useState } from 'react'
import { Box, Button, Input, Modal, ModalBody, ModalCloseButton,
ModalContent, ModalFooter, ModalHeader, ModalOverlay, SimpleGrid,
Text, useDisclosure, useToast } from '@chakra-ui/react'
import { GetEventContext } from '../../App'
import axios from 'axios'

const initState = {
  players: []
}

const Home = () => {
  const {eventList, getEventList, setEventList} =
useContext(GetEventContext);
  const { isOpen, onOpen, onClose } = useDisclosure();
  const [rowSeleted, setRowSelected] = useState(initState);
  const profile = JSON.parse(localStorage.getItem("sportEvent"))
|| "";
  const toast = useToast();
  const [keyword, setKeyWord] = useState("");

  const handleJoinBtn = async(id)=>{
    if(!profile){
      return toast({
        title: 'Login first.',
        status: 'warning',
        duration: 5000,
        isClosable: true,
        position: "top",
      })
    }
    if(profile.role === "organizer"){
      return toast({
        title: 'Organizers cannot join.',
        status: 'warning',
        duration: 5000,
        isClosable: true,
        position: "top",
      })
    }
    await axios({
      url: `http://localhost:8080/event/${id}`,
      method: "PUT",
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json;charset=UTF-8"
      },
      data: {
```

```
        "player" : profile.user
      }
    })
      .then((res) => {
          console.log(res.data);
          getEventList();
          onClose();
          toast({
            title: 'Joined Successfully.',
            status: 'success',
            duration: 3000,
            isClosable: true,
            position: "top",
          })
      })
      .catch((e) => {
          toast({
            title: 'Something went worng try again.',
            status: 'warning',
            duration: 3000,
            isClosable: true,
            position: "top",
          })
        }
      );
  }

  const handleSearch = async(e)=>{
    e.preventDefault();
    await
axios.get(`http://localhost:8080/event/search?keyword=${keyword}`)
      .then((res)=>{
        setEventList(res.data)
      })
      .catch((e)=>{
        console.log(e)
      })

      setKeyWord("")
  }

  const getAllEvents = (e)=>{
    e.preventDefault();
    getEventList();
  }

  useEffect(()=>{
    getEventList();
  },[])

  return (
    <Box w={'90%'} m={'auto'} mt={'100px'}>
      <Text fontFamily={'cursive'} fontSize={'3xl'} mb={10}
color={'red'}>Events</Text>
```

```
      <Box display={'flex'} gap={5} mb={10}>
        <Input
          name='keyword'
          value={keyword}
          onChange={(e)=>setKeyWord(e.target.value)}
          placeholder='Search by venue, event type, description or
organizer name'
        />
        <Button onClick={handleSearch}>Search</Button>
        <Button onClick={getAllEvents}>Get all Events</Button>

      </Box>
        <SimpleGrid columns={[1,1,1,3]} gap={10}>
          {
            eventList?.map((el)=>(
              <Box key={el._id} boxShadow='outline' p='6'
rounded='md' bg='white' fontFamily={'cursive'}>

                <Box display={'flex'} justifyContent={'space-
between'}>
                  <Text color={'red'}
fontSize={'xl'}>{el.venue}</Text>
                  <Text>Date: {el.date}</Text>
                </Box>

                <Box textAlign={'center'} boxShadow='xl' p='6'
rounded='md' bg='white' color={'gray'}>{el.desc}</Box>

                <Text textAlign={'left'} mt={3}
fontSize={'2xl'}>{el.type}</Text>
                <Text textAlign={'left'} mt={3}>Organizer Name:
{el.org_name}</Text>

                <Button mt={3} onClick={()=>{
                  onOpen();
                  setRowSelected(el)
                  }}>More Detail</Button>
              </Box>
            ))
          }
        </SimpleGrid>
        <Modal isOpen={isOpen} onClose={onClose}>
              <ModalOverlay />
              <ModalContent>
                <ModalHeader>Player Limit for the event:
{rowSeleted.playersLimit}</ModalHeader>
                <ModalCloseButton />
                <ModalBody fontFamily={'cursive'} >
                  {
                    rowSeleted.players.length === 0 ?
                    "Be the first one to join the team" :
                    <Box>
                      <Text mb={5} color={'red'}>List of Players
Joined</Text>
```

```
                        <Box display={'flex'} gap={10}
flexWrap={'wrap'}>
                          {
                            rowSeleted.players?.map((el,i)=><Box
key={i}>{el}</Box>)
                          }
                        </Box>
                      </Box>
                    }

                  </ModalBody>

                  <ModalFooter>
                  <Button
                    onClick={() => {
                      handleJoinBtn(rowSeleted._id)
                    }}
                    colorScheme="whatsapp"
                  >
                    Join
                  </Button>
                  </ModalFooter>
                </ModalContent>
              </Modal>
    </Box>
  )
}

export default Home
```

Додаток Б

Програмний код для реєстрації нового користувача «Signup.jsx»

```
    import { Button, Checkbox, Input, useToast } from '@chakra-
ui/react'
    import axios from "axios";
    import { useState } from 'react';

    const initState = {
        name: "",
        email: "",
        password: "",
        role: "user"
    }

    const Signup = () => {
        const [form, setForm] = useState(initState);
        const toast = useToast()

        const handleChange = (e)=>{
            const { name, value, type } = e.target;

            const valueToUpdate = type === "checkbox" ?
"organizer" : value;

            setForm({
                ...form,
                [name]: valueToUpdate
              });;
        }

        async function createAccount(e){
          e.preventDefault();

          await axios({
            url: "http://localhost:8080/user/signup",
            method: "POST",
            headers: {
              Accept: "application/json",
              "Content-Type": "application/json;charset=UTF-8"
            },
            data: form
          })
            .then((res) => {
                console.log(res.data);
                toast({
                  title: 'Account Created Successfully.',
                  status: 'success',
                  duration: 3000,
                  isClosable: true,
                  position: "top",
                })
```

```
            })
            .catch((e) => {
              if(e.response.data === "This email is already in
use try with other email."){
                toast({
                  title: 'This email is already in use try with
other email.',
                  status: 'warning',
                  duration: 3000,
                  isClosable: true,
                  position: "top",
                })
              }else{
                toast({
                  title: 'Something went worng try again.',
                  status: 'warning',
                  duration: 3000,
                  isClosable: true,
                  position: "top",
                })
              }
            });

          setForm({
            name: "",
            email: "",
            password: "",
            role: "user"
          });
        }

      return (
        <>
          <form onSubmit={createAccount}>
            <Input
              type={"text"}
              onChange={handleChange}
              value={form.name}
              name="name"
              required="required"
              placeholder='Name'
            />
            <Input
              type={"email"}
              onChange={handleChange}
              value={form.email}
              name="email"
              required="required"
              placeholder='Email'
              mt={10}
              />
            <Input
              type={"password"}
              onChange={handleChange}
```

```
                value={form.password}
                name="password"
                required="required"
                placeholder='Password'
                mt={10}
              />

              <Checkbox
                  mt={10}
                  name={'role'}
                  onChange={handleChange}
                  checked={form.role}
              >Are you a Event Organizer</Checkbox>

              <Button type='submit' mt={20} colorScheme='teal'
size='md'>
                  Create Account
              </Button>

            </form>
          </>
        )
      }

      export default Signup
```