МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота
другий (магістерський)
(рівень вищої освіти)

на тему **Особливості побудови серверної частини автоматизованої системи для організації та проведення спортивних змагань**

Виконав: студент 2 курсу, групи 8.1212-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
О.А. Білокобильский
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н. І.А. Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ Дискус
Р.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра Електроніки, інформаційних систем та програмного забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська___
"_01_" __вересня_____2023_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

____Білокобильському Олексію Андрійовичу_____
(прізвище, ім'я, по батькові)

1.Тема роботи __Особливості побудови серверної частини автоматизованої системи для організації та проведення спортивних змагань_____

керівник роботи ___Скрипник Ірина Анатоліївна, к.ф.-м.н., доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 09.10. 2023 р. №1577-с_
2. Строк подання студентом кваліфікаційної роботи 1 грудня 2023 р.
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження технологій, методів та засобів проектування веб-сайтів;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

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

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області шляхом комунікація із підприємствами для виявлення недоліків існуючих систем	02.09-12.09.23	виконано
2	Формулювання та узгодження із керівником тематики роботи	13.09-14.09.23	виконано
3	Вивчення аналогів, існуючих на ринку програмних продуктів	15.09-21.09.23	виконано
4	Визначення вимог до майбутньої інформаційної системи	22.09-25.09.23	виконано
5	Проектування бази даних та заповнення її інформацією	26.09-29.09.23	виконано
6	Проектування інтерфейсу користувача	30.09-05.09.23	виконано
7	Створення початкового застосунку та його зв’язування із БД	06.10-10.10.23	виконано
8	Реалізація спроектованого інтерфейсу	11.10-21.10.23	виконано
9	Проектування основних алгоритмів роботи системи	22.10-27.10.23	виконано
10	Розробка спроектованих алгоритмів	28.10-10.11.23	виконано
11	Зв’язування розроблених інтерфейсів із алгоритмами	11.11-18.11.23	виконано
12	Тестування інформаційної системи та виправлення недоліків	19.11-21.11.23	виконано
13	Оформлення звіту	22.11-27.11. 23	виконано

Студент _____ О.А. Білокобильський
(підпис) (ініціали та прізвище)

Керівник роботи _____ І.А. Скрипник
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (ініціали та прізвище

АНОТАЦІЯ

Сторінок: 84

Рисунків: 29

Джерел: 27

Білокобильський О.А. Клієнт-серверний веб-застосунок системи реєстрації, проведеня та менеджменту спортивних змагань : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І. А. Скрипник. Запоріжжя : ЗНУ, 2023. 84 с.

Мета роботи полягає у вивченні використовуваних підприємствами програмних застосунків, їхніх альтернатив та реалізації інформаційної системи для системи реєстрації, проведеня та менеджменту спортивних змагань що містить в собі всі необхідні вимоги.

Предметом дослідження є необхідність створення власного застосунку для задоволення потреб малих та середніх підприємств та зменшення їх фінансового навантаження шляхом об'єднання функціоналу сайту та аппаратної частини.

У процесі дослідження були розглянуті популярні системи для створення, мониторингу та проведення спортивних змагань. Як результат, була спроектована інформаційна система управління та мониторингу спортивних змаганнь. Даний застосунок призначений для проведення та мониторингу спортивних змаганнь, реєстрації учасників, можливості переглядати особисті результати у кабінеті учасника, та сповіщати про старт / фініш / та результати змагань або їх окремих етапів, та відображення результатів змагання / етапів у реальному часі.

Ключові слова: інформаційна система для керування змаганнями, клієнт-серверний застосунок, JavaScript, Socket.io, Redux, Node.JS, ExpressJS, ReactJS, Telegram Bot, SPA-застосунок, події, сховище, база даних, сокети.

SUMMARY

Pages: 84

Figures: 29

Source: 27

Bilokobylsky O.A. Client-server web application of the system of registration, conduct and management of sports competitions: master's qualification thesis of specialty 121 "Software engineering" / science. manager I. A. Skrypnyk. Zaporizhzhia: ZNU, 2023. 84 p.

The purpose of the work is to study the software applications used by enterprises, their alternatives and the implementation of an information system for the system of registration, holding and management of sports competitions, which contains all the necessary requirements.

The subject of the research is the need to create an own application to meet the needs of small and medium-sized enterprises and reduce their financial burden by combining the functionality of the site and the hardware part.

In the process of research, popular systems for creating, monitoring and conducting sports competitions were considered. As a result, an information system for managing and monitoring sports competitions was designed. This application is intended for conducting and monitoring sports competitions, registration of participants, the ability to view personal results in the participant's office, and to notify about the start / finish / and results of competitions or their individual stages, and displaying the results of competitions / stages in real time.

Keywords: competition management information system, client-server application, JavaScript, Socket.io, Redux, Node.JS, ExpressJS, ReactJS, Telegram Bot, SPA application, events, storage, database, sockets.

ЗМІСТ

ВСТУП

Спортивні змагання - це один з найпопулярніших заходів у світі, що збирає величезну кількість людей і привертає увагу з усіх куточків планети. При проведенні змагань багато факторів потрібно враховувати, починаючи від вибору місця, до системи обліку результатів.

З ростом компьютерних технологій, багато організацій, які займаються організацією змагань усе частіше використовують сучасні цифрові рішення для майже усіх етапів проведення змагання або турніру : реєстрація, проведення самого змагання, підрахунок результатів, видача призових місць.

Такі системи можуть автоматично фіксувати результати, показувати статистику та полегшити роботу суддів та організаторів, та вцілому роблять змагання більш привабливим для простого глядача, бо такі системи добре та зручно показують результати атлетів у реальному часі, що не дасіть нікому не занудьгувати.

Але навіть на сьогодніший час, організатори турнірів середнього звена до сих пір використовують застарілі та залежні від людського фактору методи : замір старту або фінішу за допомогою людини з секундомером, вписання результатів атлетів вручну, підрахунок усіх результатів вручну. Вони користуються даними методами через дорожнечу аренди або купівлі обладнання, яке може з точністю до мілісекунду зчитати старт або фініш атлета. Деякі апаратні рішення, навіть в аренді, доходять до десятків тисяч доларів, що унеможливлює їх придбання на більш локальних турнірах.

Актуальність створення системи обумовлена необхідністю створити систему , яка на відміну від аналогів буде мати можливість реєстрації особистого кабінету з функцією перегляду історії змагань, а також буде доступнішою за конкурентні, при цьому не втрачаючи якості.

Мета роботи

Мета роботи полягає у вивченні існуючих аналогічних систем проведення та керування спортивних змананнь, їхніх альтернатив та реалізації інформаційної системи для реєстрації, проведення та менеджменту спортивних вело-змагань, яка містить в собі весь необхідний функціонал.

Об’єкт дослідження

Об’єктом дослідження є процес організації, менеджменту та супроводу спортивних вело-змагань, а також вклад клієнт-серверного веб застосунку у цей процес.

Предмет дослідження

Предметом дослідження є необхідність створення власного застосунку для задоволення потреб малих та середніх бізнесів та зменшення їх фінансового навантаження шляхом об'єднання інформаційної сторони системи проведення змагань та апаратної частини таких систем.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз особливостей та існуючих рішень (аналогів) для проведення спортивних змагань в світі.
2. Аналіз технологій для реалізації такої системи.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що для вирішення задачі доцільне створення системи проведення змагань, яка вміщує в собі інформаційну частину (пророоблений веб-сайт з розподіленням ролей та можливістю переглядання результатів для звичайних користувачів), а також апаратну частину та надання послуг ренти застосунку.

Глосарій

JavaScript (JS) — це високорівнева мова програмування, яка використовується для розробки веб-застосунків та динамічного веб-контенту.

VS Code — це популярне середовище розробки з відкритим вихідним кодом, яке надає широкі можливості для редагування коду різних мов програмування.

Велосипедні змагання — це спортивні змагання, де учасники змагаються на велосипедах по певній трасі або у певних дисциплінах.

Redis — це система зберігання даних у форматі ключ-значення, яка часто використовується для кешування даних, прискорення доступу до них та зберігання сесій користувачів.

Next.js — це фреймворк JavaScript для розробки веб-додатків на базі React. Він надає можливість створювати універсальні додатки, що підтримують як клієнтську, так і серверну частини, з можливістю рендерингу на сервері.

Node.js — це середовище виконання JavaScript, яке дозволяє виконувати код JavaScript поза браузером. Воно часто використовується для розробки серверних застосунків, що дозволяє створювати швидкі та масштабовані програми.

Серверний застосунок — це програмне забезпечення, яке виконується на сервері і обробляє запити від клієнтської частини. Він може включати в себе логіку обробки даних, взаємодію з базою даних, обробку запитів користувачів тощо.

Клієнтська частина — це та частина програмного забезпечення, яка запускається на браузері чи мобільному пристрої користувача. Вона відповідає за відображення та взаємодію з користувачем, включаючи відправку запитів до серверної частини та отримання відповідей.

Телеграм-бот — це програма, розроблена для взаємодії з користувачами у месенджері Telegram. Вона може виконувати різні завдання, відповідати на запити користувачів, відправляти повідомлення, обробляти команди тощо, і використовується для автоматизації різних процесів.

Інтерфейс — це зовнішній вигляд програми або системи, що надає користувачеві можливість взаємодіяти з ними. Інтерфейс може містити елементи керування, які дозволяють користувачу взаємодіяти з програмою, такі як кнопки, поля введення, меню тощо.

Інформаційна система для керування підприємством — це комп'ютерна система, призначена для автоматизації різних процесів у підприємстві, таких як фінансове планування, управління персоналом, контроль запасів тощо.

Підприємство — це юридична особа, яка займається виробництвом або наданням послуг з метою отримання прибутку.

Спортивні компанії — це підприємства, що спеціалізуються на виробництві спортивного спорядження, одягу, обладнання для фітнесу, а також можуть займатися спонсорством спортивних заходів, організацією спортивних подій, продажем спортивних товарів та послуг. Вони активно працюють у спортивній індустрії, сприяють розвитку спорту, спонсорують команди, спортсменів та заходи, спрямовані на підвищення активності та здорового способу життя. Такі компанії можуть бути спеціалізованими на конкретному виді спорту або працювати у багатьох його напрямках

Програмний застосунок — це програма, яка створена для вирішення конкретної задачі або набору задач на комп'ютері.

WebSocket — це технологія, що забезпечує двостороннє з'єднання між клієнтом та сервером через одне TCP-з'єднання, яке дозволяє взаємодіяти в реальному часі. Веб-сокети використовуються для передачі даних між браузером користувача та веб-сервером, а також між сервером і веб-додатком, без необхідності постійного відправлення запитів з боку клієнта на сервер для оновлення інформації.

Socket.IO — це бібліотека для реалізації багатокористувацьких застосунків в реальному часі через веб-сокети. Вона дозволяє побудувати зв'язок між клієнтом і сервером, де обидві сторони можуть взаємодіяти, відправляти та отримувати дані в реальному часі, спрощуючи реалізацію потокових додатків, чатів, гри тощо.

Модель MVCS (або MVSC) — це архітектурна модель програмування, яка розбиває додаток на чотири основні компоненти: Model (Модель), View (Вид), Controller (Контролер) та Services (Сервіси). Модель відповідає за обробку даних, Вид - за їх відображення, Контролер - за логіку взаємодії між моделлю та видом, а Сервіси - за додаткову функціональність, яка може бути використана в додатку.

ACL (Access Control List) — це механізм контролю доступу, який використовується для управління правами доступу користувачів або ресурсів в інформаційній системі. ACL визначає, які користувачі чи групи мають доступ до певних ресурсів або операцій, а також які права вони мають на ці ресурси (наприклад, читання, запис, видалення). Це важливий механізм для забезпечення безпеки даних і ресурсів в програмному забезпеченні.

1 ОГЛЯД СИСТЕМ МОНІТОРІНГУ І ПРОВЕДЕННЯ ЗМАГАНЬ

1.1 Огляд проведених досліджень та наукових джерел

Однією з важливих складових ефективної розробки програмного застосунку для організації спортивних змагань є дослідження та аналіз наукових джерел, пов'язаних з цією тематикою. У даному розділі проводиться огляд проведених досліджень та наукових джерел, що стосуються організації спортивних змагань, а також програмних застосунків, використовуваних у даній сфері.

Огляд наукових досліджень включає докладне дослідження наукових праць, які стосуються організації спортивних змагань та суміжних областей. Були вивчені актуальні наукові роботи, які висвітлюють проблеми, виклики та можливі рішення у галузі організації спортивних змагань. При огляді досліджень було звернуто увагу на ключові висновки та рекомендації, які можуть мати значення для даного дослідження.

Також було проведено аналіз наукових джерел, таких як наукові статті, журнали, конференції, тези, книги тощо, що стосуються теми дослідження. Цей аналіз спрямовувався на визначення основних тенденцій, методів, підходів та інновацій, згаданих у цих джерелах. Особлива увага була приділена виявленню прогалин або невирішених проблем, які будуть підґрунтям для подальшого дослідження.

Крім того, було проведено аналіз існуючих програмних застосунків, використовуваних для організації спортивних змагань. Розглянуто їх функціональні можливості, обмеження, успіхи та недоліки. Цей аналіз дав змогу визначити, що вже було зроблено у галузі програмних рішень для організації спортивних змагань, а також виявити потенційні можливості, які ще потребують дослідження та вдосконалення.

На основі проведеного огляду досліджень та наукових джерел виділено прогалини та потенційні області, які стануть основою для подальшого дослідження у цій дипломній роботі.

Загальний огляд проведених досліджень та наукових джерел вказує на актуальність обраної теми дослідження та необхідність подальшого вивчення і вдосконалення програмних рішень для організації спортивних змагань

1.2 Огляд аналогічних систем.

Під час огляду аналогів, можна відокремити декілька систем проведення спортивних змагань :

Race Result –інтегроване рішенням для управління спортивними подіями та результатами змагань. Це програмне забезпечення спеціалізується на автоматизації обробки результатів спортивних заходів, управлінні змаганнями та наданні зручних інструментів для організаторів та учасників. [1].

Основні функції Race Result включають:

Реєстрація та управління учасниками: Це може включати в себе онлайн-реєстрацію учасників на подію, управління списками учасників, додавання та редагування їхньої інформації.

Управління результатами: Race Result дозволяє організаторам введення та обробку результатів змагань, автоматизуючи процес фіксації часу, дистанції та інших параметрів.

Часовий контроль та система таймінгу: Ця система може включати в себе використання електронних систем часу для фіксації результатів учасників під час змагань.

Маршрути та трекери: Відстеження маршрутів учасників, відображення треку та дистанції, що пройдені під час змагань.

Відображення результатів: Інтеграція із системами демонстрації результатів під час змагань для учасників та глядачів.

Аналітика та звіти: Надання організаторам спортивних подій аналітичних звітів, які допомагають у здійсненні аналізу результатів та інших даних, що стосуються події.

Race Result є інструментом, який спрощує та автоматизує багато аспектів організації та ведення спортивних заходів. Він допомагає забезпечити точність та доступність результатів для учасників та глядачів, а також полегшує роботу організаторів подій.

Серед недоліків системи можна виділити наступні:

- висока вартість обладнання;
- необхідність щомісячної сплати підписки;
- інтерфейс розрахований не розрахований на звичайних користувачів.

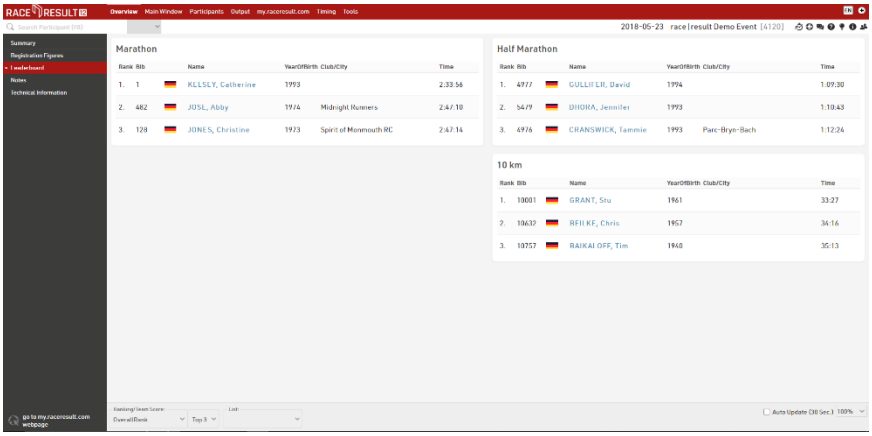

Рис. 1 – Інтерфейс дашборду веб-застосунку «Race Result»

Апаратна частина цієї системи, яка потрібна для проведення змагань і складається з великої кількості деталей та компонентів, може коштувати від декількох тисяч до десяток тисяч доларів (в залежності від типу змагання та кількості учасників). Якщо ви хочете створити змагання на базі їх веб-додатком, то за кожне розміщення та аналіз змагання потрібно заплатити певну суму (залежно від кількості учасників).

Challonge – це онлайн-платформа для організації та проведення турнірів і змагань будь-якого типу. Вона надає інструменти для створення розкладів, реєстрації учасників, керування матчами, відстеження результатів і багато

іншого[2].

Функціонал даної системи :

- Створення турнірів: Користувачі можуть створювати свої власні турніри на основі різних дисциплін та форматів. Це може бути все від електронних ігор до спортивних подій.
- Реєстрація учасників: Учасники можуть реєструватися на турнір через Challonge, додаючи свої дані та інформацію про участь.
- Структура соревнования: Платформа дозволяє налаштовувати формати змагань, включаючи одиночні та командні матчі, етапи соревнований, системи подвійного вибування, групові етапи тощо.
- Генерація розкладу: Challonge автоматично створює розклад турніру, включаючи пари для матчів та графік проведення.
- Управління матчами: Організатори та учасники можуть вносити зміни в розклад, підтверджувати результати, вести статистику тощо.
- Спільнота та спілкування: Через Challonge учасники можуть спілкуватися, обмінюватися думками та коментарями про події.
- Трансляція результатів: Результати турніру можуть бути транслювані в режимі реального часу для глядачів та інших зацікавлених сторін.
- Спонсорська підтримка: Challonge надає можливості для вступу спонсорів та партнерів у турніри для підтримки та просування їх брендів.

Система має гарний та інтуітивний інтерфейс, який дозволяє дуже легко створити змагання (див. Рис 2), але даний продукт не пропонує ніякого апаратного рішення для проведення змагань у полі або у горах.

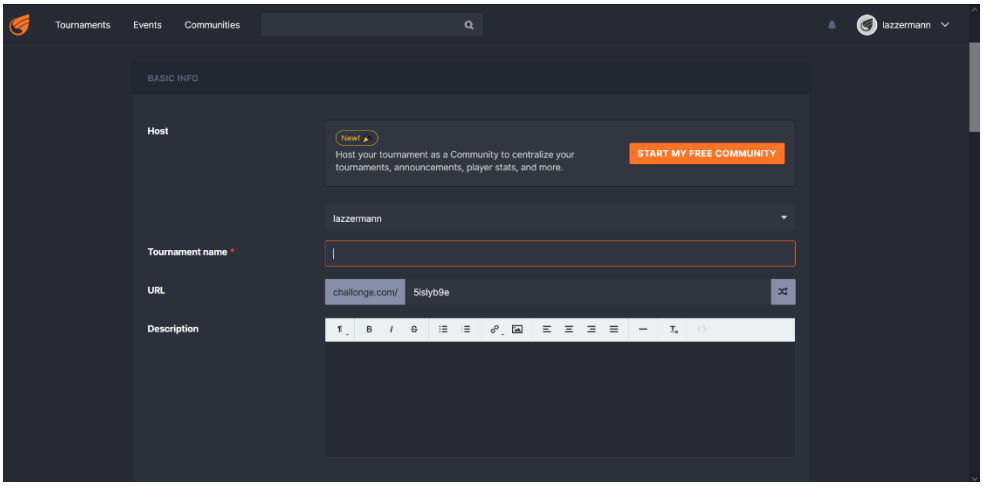

Рис. 2 – Інтерфейс сторінки створення змагання системи Challonge

IT’s your race — компанія що надає обладнання та веб-кабінет для створення та перегляду змагань у реальному часі з різноманітних дисциплін: треатлон, плавання, біг, вело-змагання [3]. Нижче подано загальний огляд основних можливостей системи IT’s Your Race:

Реєстрація на події: Учасники можуть зареєструватися на спортивні події через платформу, вибираючи тип події, дистанцію та інші параметри.

Управління подією: Організатори подій можуть використовувати IYR для створення, планування та керування подіями. Це може включати реєстрацію учасників, управління трасами, часом проведення, розподілом номерів, управління результатами тощо.

Мобільні додатки: Платформа надає мобільні додатки для учасників, які дозволяють переглядати інформацію про подію, результати, траси, а також спілкуватися з організаторами чи іншими учасниками.

Відстеження результатів: Учасники можуть відстежувати свої результати через IYR, переглядати свої часи, маршрути та підсумкові дані про участь.

Спілкування та соціальні мережі: Платформа дозволяє спілкуватися з іншими учасниками події, обмінюватися досвідом та результатами через інтеграцію з соціальними мережами.

Інтеграція з чіпами для спортивного відстеження: Деякі події можуть використовувати чіпи для відстеження часу та результатів учасників.

Аналітика та звіти: Організатори можуть отримувати аналітику та звіти про подію, включаючи кількість учасників, їхні результати, популярність та інше.

Безпека та підтримка: Платформа забезпечує заходи безпеки для захисту особистих даних учасників та надає підтримку користувачам у разі потреби.

Загалом, IT’s Your Race - це інструмент, який спрощує процес організації та участі у спортивних заходах, забезпечуючи зручність, доступність і спілкування для всіх зацікавлених сторін.

Серед переваг системи можна виділити наступне:

- перегляд результатів у реальному часі;
- легке налаштування змагань;
- масштабована ціна в залежності від кількості учасників;
- безкоштовний мобільний застосунок;

Серед недоліків системи можна виділити наступні:

- система доступна не у всіх регіонах;
- інтерфейс (див. Рис. 3) не відповідає вимогам “User Friendly” , хоча система передбачає користування звичайними користувачами;
- відсутність особистого кабінету та перегляду історії змагань;

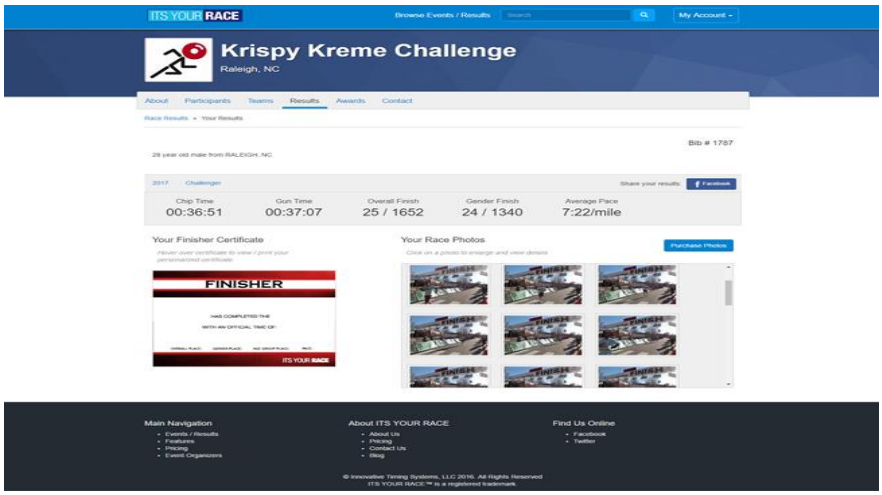

Рис. 3 – Інтерфейс сторінки перегляду змагання системи IT’s your race.

1.3 Які системи використовуються на території України

В Україні для організації спортивних змагань використовуються різноманітні системи та технології, які сприяють зручності та ефективності проведення змагань. Нижче наведено кілька прикладів таких систем:
Електронна реєстрація та керування учасниками: Часто використовується система електронної реєстрації, яка дозволяє учасникам зареєструватись онлайн для участі в спортивних змаганнях. Ця система забезпечує зручну та швидку обробку даних учасників та дозволяє забезпечити потрібну кількість місць для участі.

Системи таймерів та хронометражу: Для точного вимірювання часу під час спортивних змагань використовуються спеціальні системи таймерів та хронометражу. Вони дозволяють точно вимірювати час різних етапів змагань, таких як старт, фініш та проміжні точки, що є важливим для визначення переможців.

Системи відеонагляду та фотофінішу: Для більш точного та об'єктивного визначення результатів спортивних змагань використовуються системи відеонагляду та фотофінішу. Вони дозволяють зафіксувати момент фінішу учасників та забезпечують можливість перегляду та аналізу фінішних фотографій або відеозаписів для вирішення спірних ситуацій.

Електронна система розподілу інформації: Для забезпечення швидкого та ефективного розподілу інформації про змагання, розклади, результати та інші важливі дані використовуються електронні системи. Ці системи можуть включати веб-сайти, мобільні додатки, електронні табло тощо, які надають учасникам та глядачам легкий доступ до потрібної інформації.
Це лише кілька прикладів систем, які використовуються на території України для організації спортивних змагань.

1.4 Недоліки, які криються у використані таких систем

Існують деякі недоліки у використанні систем таймерів та хронометражу, відеонагляду та фотофінішу, а також у електронних системах розподілу інформації. Деякі з них:

Технічні неполадки: Електронні системи можуть піддаватися технічним збоям або несправностям, що може призвести до неправильного реєстрування часу або інших даних.

Залежність від електропостачання: Багато з цих систем потребують постійного живлення. Проблеми з живленням, перерви у роботі електромережі або відключення живлення можуть призвести до зупинки системи.

Вразливість до втручання: Електронні системи можуть бути підвернуті втручанню з боку зловмисників, що може призвести до маніпуляцій з результатами, зокрема, при використанні в спортивних змаганнях.

Обмеженість технічних можливостей: Деякі системи можуть мати обмеження в точності часу або здатності фіксувати інші параметри (наприклад, висота чи точність фотофінішу).

Вартість і підтримка: Висока вартість придбання, налаштування та підтримки таких систем може бути проблемою для менших організацій або спортивних заходів з обмеженим бюджетом.

Необхідність професійного обслуговування: Деякі системи вимагають спеціалізованого обслуговування та технічної підтримки для належної роботи, що може бути проблемою у віддалених або відокремлених місцях.

Враховуючи ці недоліки, важливо ретельно розглянути вибір системи та впевнитися в її надійності, безпеці та відповідності потребам конкретної ситуації чи події, на яку вона буде застосовуватися.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Перед розробкою системи для проведення спортивних змагань було проведено аналіз існуючих апаратних та програмних рішень для реалізації.

2.1 Серверний застосунок

При розробці серверного застосунку першим постало питання вибору фреймворку та мови розробки. Оскількі клієнтська частина буде розроблятися єдиною мовою, що розуміють веб-браузери, а саме — JavaScript[14] , було прийняте рішення зберегти єдину структуру коду та обрати ту ж саму мову для розробки серверної частини застосунку, саме тому було обрано фреймворк NodeJS для розробки серверу.

Node.js - це вільна та відкрита серверна бібліотека, яка дозволяє виконувати JavaScript на стороні сервера. Node.js розроблений на основі двигуна JavaScript V8 компанії Google та дозволяє розробникам використовувати JavaScript як мову серверної розробки [16].

Основні переваги Node.js:

- Швидкість та продуктивність: Node.js дозволяє швидко створювати додатки, які мають велику кількість одночасних підключень, так як Node.js має неблокуючу модель вводу/виводу, що дозволяє ефективніше використовувати ресурси сервера.
- JavaScript: Node.js дозволяє розробникам використовувати JavaScript на стороні сервера, що дозволяє розробникам використовувати знайому мову програмування на обох сторонах - клієнтській та серверній.
- Бібліотеки та фреймворки: Node.js має велику кількість бібліотек та фреймворків, які дозволяють розробникам зосередитись на функціональності додатків, а не на базових функціях розробки.
- Спільнота: Node.js має велику та активну спільноту розробників, яка постійно працює над розвитком та підтримкою бібліотек та фреймворків.

- Масштабованість: Node.js дозволяє легко масштабувати додатки та розширювати їх функціональність, що дозволяє розробникам швидко відповідати на потреби бізнесу та користувачів.

Також на серверній частині застосунку, було прийняте рішення щодо використання сесійного сховища Redis. Редіс (Redis) - це відкрите програмне забезпечення, що використовується для зберігання даних у вигляді ключ-значення у пам'яті. Він відноситься до баз даних типу NoSQL, але має також можливості кешування, атомарних операцій і роботи зі структурами даних, такими як списки, хеші, множини тощо.

Редіс часто використовується для швидкого зберігання та отримання даних, особливо в інтернет-застосунках, де швидкість доступу до даних є ключовою. Він може використовуватися як база даних, кеш пам'яті або посередник між різними системами для зберігання та обміну даними.

Однією з головних переваг Redis є його швидкодія, оскільки дані зберігаються у пам'яті, що дозволяє виконувати швидкі операції з ними. Також Redis має ряд функцій, які роблять його корисним інструментом для розробників програмного забезпечення, таких як можливість роботи з рядами, сортованими множинами, публікація/підписка на повідомлення та інші.

Цей інструмент є досить популярним у світі програмування через свою простоту використання і ефективність у вирішенні різноманітних задач, пов'язаних зі зберіганням та обробкою даних.

2.2 Огляд баз даних

При виборі бази даних ключовим фактором став вибір типу бази даних. Існують реляційні та нереляційні бази даних, тому в першу чергу було прийнято рішення ознайомитись з прикладами баз даних обох типів , оцінити переваги та недоліки.

Серед існуючих реляційних баз даних було обрано наступні бази:

- MySQL - це реляційна база даних, яка забезпечує зберігання та керування структурованими даними за допомогою SQL запитів. Вона є дуже популярною та використовується в багатьох веб-додатках [12].
- Microsoft SQL Server - це також реляційна база даних, розроблена компанією Microsoft. Вона має багатий функціонал, підтримує безпеку та масштабується [11].

Серед існуючих нереляційних баз даних було розглянуто наступний варіант:

- MongoDB - це документ-орієнтована нереляційна база даних, яка зберігає дані у вигляді документів у форматі BSON (Binary JSON). Вона широко використовується для зберігання та обробки великих обсягів даних та дозволяє більш гнучке зберігання та операції з даними [8].

При огляді баз даних вибір пав на MongoDB, через те що нереляційні бази даних показують більш ефективну работу при накопичуваному типу проекту яким і є система організації та керування змаганнями, а також мова запиту є JSON документом, що представляє собою фільтри у вигляді ключів у такому документі.

Серед інших переваг MongoDB перед SQL базами даних можна зазначити наступні:

- Гнучкість: MongoDB дозволяє легко змінювати структуру даних та розширювати її.
- Швидкодія: MongoDB працює швидше за більшість SQL баз даних, оскільки не використовує складні операції JOIN та інші вимоги до структури даних.
- Масштабованість: MongoDB може працювати в розподіленому середовищі, що дозволяє розміщувати дані на декількох серверах та збільшувати їх потужність зростаючи обсягу даних.
- Невимогливість: MongoDB не вимагає строгих схем даних, що дозволяє зберігати дані з різної структурою та форматом.

- Надійність: MongoDB дозволяє використовувати реплікацію та шардування для забезпечення високої доступності та стійкості до збоїв.

2.3 Клієнтський застосунок

Клієнтський застосунок представляє собою SPA (Single Page Application) [15] застосунок. Односторінкова програма (SPA) - це тип веб-додатку, який працює в браузері і взаємодіє з користувачем, не перезавантажуючи повністю сторінку під час взаємодії. В SPA весь код (HTML, CSS, JavaScript) завантажується під час першого відкриття сторінки, а далі для відображення нової інформації або зміни взаємодії використовуються AJAX-запити, які отримують або відправляють дані на сервер і оновлюють лише необхідні частини сторінки.

Основні переваги SPA:

Швидкість: Коли користувач взаємодіє з програмою, не потрібно завантажувати повністю нову сторінку, що забезпечує швидку реакцію та відмінний досвід користувача.

Менша потреба в серверних ресурсах: Односторінкові програми можуть зменшити навантаження на сервер, оскільки не кожен раз відправляється запит на повне завантаження сторінки.

Більша інтерактивність: SPA можуть бути більш інтерактивними, оскільки вони можуть динамічно оновлювати вміст без перезавантаження сторінки, забезпечуючи зручний і безперервний досвід.

Робота офлайн: Якщо дані були завантажені під час першого відкриття сторінки, SPA можуть працювати офлайн, зберігаючи та використовуючи локальні дані.

Однак у SPA є і деякі недоліки:

SEO-оптимізація: Можуть виникати питання з оптимізацією для пошукових систем, оскільки інформація динамічно завантажується за допомогою JavaScript, що може ускладнити індексацію вмісту пошуковими системами.

Перша загрузка: Час першого завантаження сторінки може бути тривалим, оскільки всі ресурси (JavaScript, CSS, дані) завантажуються одночасно при відкритті сторінки.

Безпека: Існує питання щодо безпеки, оскільки весь код вже знаходиться на клієнтській стороні, що може викликати питання з приватністю та безпекою даних.

SPA може бути реалізована за допомогою різних JavaScript-фреймворків та бібліотек, таких як React, Angular, Vue.js тощо, які надають різні зручні інструменти для створення односторінкових додатків..

ReactJS - це відкрите JavaScript-фреймворк для створення інтерфейсів користувачів (UI). Він був розроблений компанією Facebook та забезпечує швидку та ефективну розробку веб-додатків. ReactJS базується на концепції компонентів, які дозволяють розбити інтерфейс на окремі елементи та повторно використовувати їх в інших місцях додатку. Він також забезпечує взаємодію зі станом додатку та реагує на зміни [9].

Основні переваги ReactJS:

- Швидкодія: ReactJS забезпечує швидку відповідь на зміни стану додатку та швидке відображення інтерфейсу користувача.
- Модульність: ReactJS забезпечує створення модульних компонентів, що дозволяє легко розширювати та повторно використовувати їх у додатку.
- Стабільність: ReactJS забезпечує стабільну та просту структуру додатку, що дозволяє розробникам легко управляти інтерфейсом користувача.
- Інструменти: ReactJS має широкий вибір інструментів та бібліотек для розробки та тестування додатків.

Vue.js - це відкрите JavaScript-фреймворк для розробки веб-додатків та інтерфейсів користувача. Vue.js був створений в 2014 році і отримав значний розповсюдження завдяки своїй простоті та легкості вивчення.

Основна ідея Vue.js полягає у тому, щоб забезпечити легкий та простий для використання інструмент для створення інтерфейсів користувачів. Vue.js забезпечує такі можливості, як маршрутизація, підтримка плагінів та

компонентів, стан додатку, реактивне програмування та багато іншого [12].

Основні переваги Vue.js:

- Простота використання: Vue.js є легким та простим для вивчення фреймворком.
- Гнучкість: Vue.js дозволяє розробникам використовувати його як бібліотеку або як повноцінний фреймворк.
- Висока продуктивність: Vue.js забезпечує високу продуктивність завдяки використанню віртуального DOM.
- Реактивність: Vue.js забезпечує реактивність додатку, що дозволяє реагувати на зміни стану додатку.
- Широкий вибір інструментів: Vue.js має широкий вибір інструментів та бібліотек для розробки та тестування додатків.

Angular - це платформа та фреймворк розробки веб-додатків, розроблений і підтримуваний компанією Google. Він використовується для створення масштабованих та модерних односторінкових додатків.

Основні особливості Angular:

Компонентна архітектура: Angular базується на компонентах. Додаток будується з різних компонентів, які містять різні частини користувацького інтерфейсу.

Typescript: Основна мова розробки - TypeScript, який є типізованою версією JavaScript. Це дозволяє підвищити якість коду, зменшити помилки та покращити підтримку від IDE.

Двостороннє зв'язування даних: Механізм Angular, який дозволяє автоматично оновлювати дані в користувацькому інтерфейсі при їх зміні у коді і навпаки.

Dependency Injection (DI): Дозволяє впроваджувати залежності в компонентах, що полегшує тестування, забезпечує кращу модульність та підтримку змінності.

Роутинг: Angular має вбудований механізм маршрутизації, який

дозволяє створювати веб-додатки з багаторівневою навігацією, яка відображається без перезавантаження сторінок.

HTTP-запити: Angular надає модуль для взаємодії з веб-серверами через HTTP-запити.

Модульність: Додатки можуть бути розділені на модулі для кращої організації коду та зручності масштабування.

Тестування: Angular надає вбудовані інструменти для тестування коду, що полегшує створення єдиної інтеграції та модульного тестування.

Angular широко використовується для розробки веб-додатків будь-якої складності. Він забезпечує високий рівень організації коду, продуктивність та розширюваність, що робить його популярним фреймворком серед розробників.

Для розробки клієнтського застосунку було обрано використовувати ReactJS через те що окрім того що він як і VueJS має велику кількість розробників що ним користуються та підтримують, він має також і авторитетні компанії , що його розробляють такі як Facebook та WhatsApp, а також тому що цей фреймворк підходить для розробки десктопних та мобільних застосунків.

Основною проблемою ReactJS є неможливість роботи з SPA застосунками та виходячі з цього плоха індексація сайту. Аби вирішити цю проблему було прийняте рішення використовувати NextJS.

Next.js - це реактивний фреймворк для розробки веб-додатків на базі React. Він пропонує ряд переваг для розробників, зокрема, простішу налаштування, покращену продуктивність та можливості оптимізації для SEO.

Основні особливості Next.js:

Рендерінг на стороні сервера (SSR) та на стороні клієнта (CSR): Next.js підтримує як серверний рендерінг (Server-Side Rendering), так і можливість використання традиційного клієнтського рендерингу. Це дозволяє забезпечити швидку стартову загрузку і оптимізацію для SEO, а також динамічні зміни в залежності від взаємодії з користувачем.

Статичне сайтогенерування (SSG): Next.js дозволяє створювати

статичні сайти, що дозволяє забезпечити високу швидкість завантаження, просту розгортання та покращення продуктивності.

Універсальність маршрутизації: Маршрутизація в Next.js дозволяє створювати різні типи сторінок та маршрутів, забезпечуючи відмінний досвід користувача.

Підтримка TypeScript: Next.js відмінно інтегрується з TypeScript, що дозволяє писати більш безпечний і типізований код.

Розширені можливості CSS: Next.js підтримує CSS Modules, Styled JSX, SCSS, CSS-in-JS бібліотеки та інші способи організації та використання CSS в проектах.

Швидке розгортання: Проєкти на Next.js можуть бути легко розгорнуті на багатьох платформах, включаючи Vercel, Netlify та інші.

Зручна оптимізація для SEO: Завдяки SSR та SSG, Next.js надає зручні можливості для оптимізації вмісту для пошукових систем.

Next.js забезпечує простоту в розробці та оптимізований веб-додаток, а також надає широкий спектр можливостей для розширення функціоналу та забезпечення високої продуктивності. [10].

Також веб-застосунку необхідне локальне сховище даних для того щоб зменшити кількість запитів до серверу, та запрошувати лише необхідні дані тоді коли це насправді необхідно. Вибором стало найпопулярніше сховище, що використовується в зв’язці з ReactJS - Redux.

Redux - це відкрите програмне забезпечення, що забезпечує управління станом додатку в JavaScript-додатках. Redux дозволяє розробникам створювати додатки, які легко підтримувати та розширювати, і які можуть працювати в різних середовищах, включаючи браузер, сервер та мобільні пристрої.

Основна ідея Redux полягає в тому, щоб зберігати стан додатку в одному місці, називаному "store". У "store" зберігаються всі дані додатку, включаючи стан компонентів і інші дані, які використовуються в додатку. Redux використовує "action" та "reducer" для зміни стану додатку [4].

"Action" - це об'єкт, який містить інформацію про зміни, які необхідно виконати, такі як додавання елемента до списку або оновлення значення. "Reducer" - це функція, яка отримує поточний стан додатку та "action" і повертає новий стан додатку.

Основні переваги Redux:

- Простота використання: Redux є простим та легким для вивчення фреймворком.
- Забезпечує централізоване управління станом додатку: Redux дозволяє зберігати стан додатку в одному місці, що забезпечує легкість управління та підтримки додатку.
- Сумісність з багатьма фреймворками: Redux можна використовувати з різними фреймворками та бібліотеками, такими як React, Angular та Vue.
- Допомагає вирішувати проблему "callback hell": Redux дозволяє розробникам використовувати "middleware", який допомагає вирішувати проблему занадто багато вкладених функцій.
- Тестування: Redux забезпечує легкість тестування додатку за допомогою інструментів перегляду історії подій та зміни станів за допомогою розширення на браузер.

2.4 Обмін даними в реальному часі

Останнім питанням постало питання обміну даними між сервером та клієнтськими застосунками (веб-сторінка, застосунок на комп’ютері, мобільний застосунок) у реальному часі для відображення найактуальніших даних про стан змагання, результати тощо. Для цього було обрано систему обміну даними між двома пристоями за допомогою сокетів з використанням бібліотеки SocketIO.

Socket.io - це бібліотека JavaScript для реалізації двостороннього зв'язку між клієнтом і сервером через веб-сокети. Вона дозволяє розробникам створювати реально-часові додатки, які можуть передавати дані між браузером або

іншим клієнтом та сервером в режимі реального часу[5].

Основна ідея Socket.io полягає в тому, що вона дозволяє клієнтам та серверам взаємодіяти між собою в реальному часі через веб-сокети. Це означає, що якщо будь-який з клієнтів чи серверів відправляє дані, інші клієнти чи сервери отримують ці дані відразу ж.

Socket.io дозволяє розробникам створювати різні типи додатків, такі як чати, онлайн-ігри, потокове відео та інші, які потребують реального часу для взаємодії між користувачами та сервером.

Основні переваги Socket.io:

- Простота використання: Socket.io є простою та легкою для вивчення бібліотекою.
- Реальний час: Socket.io дозволяє передавати дані між клієнтами та сервером в режимі реального часу.
- Підтримка багатьох платформ: Socket.io може працювати на різних платформах, включаючи браузер, сервер та мобільні пристрої.
- Розширюваність: Socket.io дозволяє розробникам розширювати функціональність за допомогою плагінів та middleware

3 РОЗРОБКА СИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ СПОРТИВНОГО ЗМАГАННЯ

3.1 Бізнес обґрунтування та вимоги інформаційної системи

У рамках реалізації інформаційної системи системи потрібно було спроектувати систему, яка задовільняє таким вимогам:

- Автоматизація реєстрації учасників: Система повинна надати можливість легко та швидко реєструвати учасників спортивних змагань. Це допоможе уникнути ручного введення даних та забезпечить точність інформації про учасників.
- Управління графіком та розкладом змагань: Система повинна забезпечувати можливість створення, оновлення та відображення графіка проведення змагань. Це дозволить організаторам ефективно планувати та координувати різні етапи змагань.
- Результати та статистика: Система повинна надавати зручний спосіб відображення результатів змагань та статистичних даних. Це допоможе учасникам, глядачам та організаторам отримати доступ до актуальної інформації про показники успішності та прогрес учасників.
- Комунікація та співпраця: Система повинна забезпечувати можливість ефективної комунікації між різними сторонами, такими як організатори, тренери та учасники змагань. Це може включати обмін повідомленнями, нагадування про події та сповіщення про зміни у розкладі.
- Безпека даних та конфіденційність: Система повинна забезпечувати захист особистих даних учасників та інших конфіденційної інформації. Це включає захист від несанкціонованого доступу, резервне копіювання даних та використання захищених каналів комунікації.

3.2 Вимоги до програмного продукту

Перед початком проектування та розробки інформаційної системи було визначено функціональні та нефункціональні вимоги

3.2.1 Функціональні вимоги до програмного продукту

На поточному етапі розробки інформаційної системи було необхідно створити можливість:

- реєстрація користувачів через сайт або телеграм бота;
- веритикальна, наслідувальна ієрархія доступу ролей;
- створення редагування подій / компаній;
- відображення діючих подій на головній сторінці;
- оновлення таблиць на оперативних таблицях адмін-панелі в реальному часі;
- можливість перегляду оперативних результатів змагань за допомогою телеграм бота.

3.2.2 Нефункціональні вимоги до програмного продукту

- Надійність: Система повинна бути стійкою та надійною, здатною працювати без збоїв або відновлюватись в разі виникнення помилок.
- Продуктивність: Система повинна бути швидкою та ефективною, здатною обробляти великий обсяг даних та запитів без відчутних затримок. Це включає оптимізацію коду, використання швидких алгоритмів та оптимальне використання ресурсів.
- Складність інтерфейсу: Система повинна мати зрозумілий, інтуїтивний та легкий у використанні інтерфейс для користувачів. Це дозволить користувачам швидко освоїти систему та використовувати її без надмірних зусиль.
- Безпека: Система повинна забезпечувати високий рівень безпеки для захисту конфіденційної інформації та запобігання несанкціонованому

доступу. Це може включати шифрування даних, аутентифікацію користувачів та контроль доступу.

- Масштабованість: Система повинна бути здатною масштабуватись, тобто легко розширю

3.3 Розробка бази даних інформаційної системи

Для зберігання інформації буде використовуватися нереляційна база даних MongoDB на серверній частині. MongoDB - це документо-орієнтована база даних, яка використовує схему без схеми (schemaless) для зберігання даних у вигляді JSON-подібних документів. Вона відносно нова, але дуже популярна база даних, яка використовується у багатьох веб-додатках та системах зберігання даних.

Основні особливості MongoDB:

Документо-орієнтована модель даних: Дані зберігаються у вигляді документів BSON (Binary JSON), які схожі на звичайні об'єкти JSON. Це дозволяє легко зберігати та отримувати дані у вигляді складних об'єктів.

Схема без схеми (Schemaless): MongoDB не вимагає строго визначеної схеми для зберігання даних. Ви можете зберігати різні типи даних у тому ж самому наборі даних без необхідності визначення структури заздалегідь.

Гнучкість та масштабованість: MongoDB легко масштабується, дозволяючи додавати нові сервери для збільшення продуктивності та обсягу даних. Вона підтримує горизонтальне масштабування, що дозволяє розподілити навантаження на кілька серверів.

Можливості запитів: MongoDB має потужну мову запитів, яка дозволяє виконувати різні види операцій: пошук, сортування, агрегацію, фільтрацію та інші операції.

Висока швидкодія: MongoDB надає швидкий доступ до даних через використання індексів та кешування.

Підтримка реплікації та архітектури з дублюванням: Вона дозволяє створювати копії даних (реплікація) для забезпечення високої доступності та надійності системи.

MongoDB є потужним інструментом для зберігання та роботи з даними в різних типах додатків. Вона особливо підходить для веб-додатків, де потрібна гнучкість у зберіганні та роботі з різними типами даних. База даних інформаційної системи для керування спортивними змаганнями складається з восьми основних таблиць. Вся інформація про зв'язаність таблиць та типи даних, які в них міститься, наведена на рисунку (див. Рис. 4).

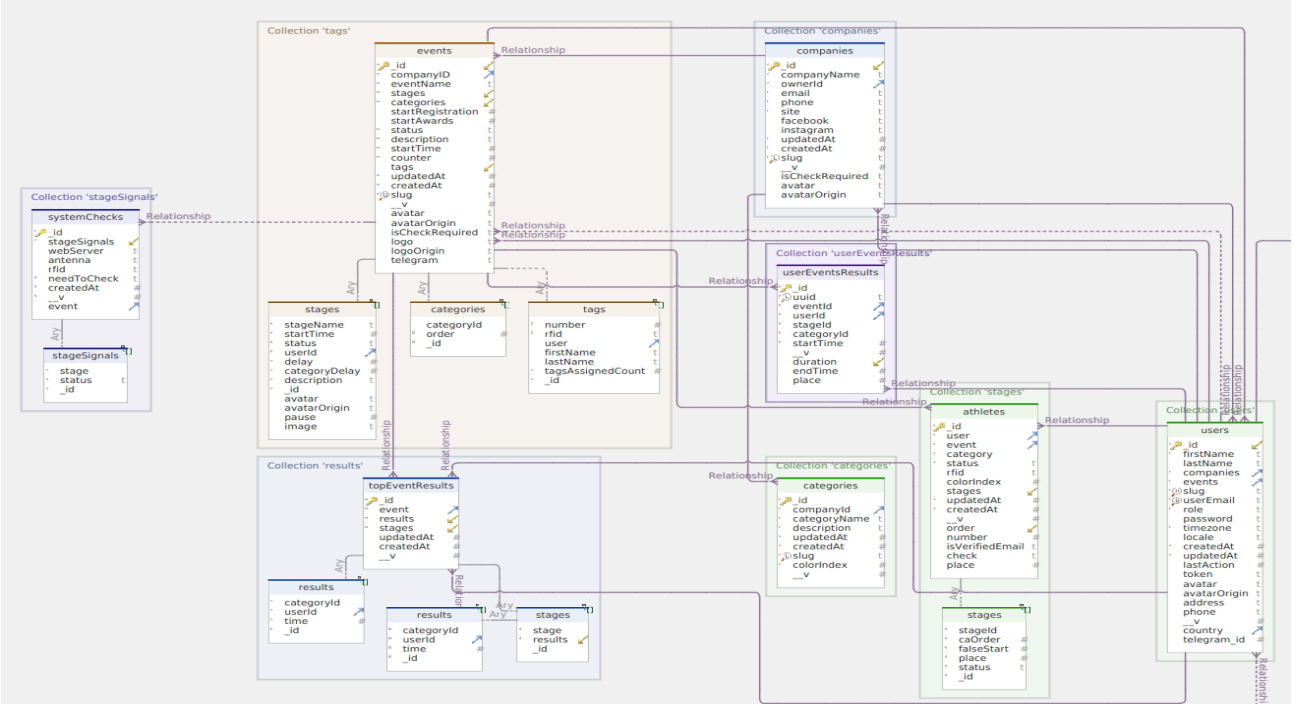

Рис. 4 Схема бази даних серверної частини

1. Users – таблиця, у якій містяться дані про людей, що зареєстровані (Ім'я, прізвище, роль, належність до певної компанії, тощо).

2. Companies – таблиця, у якій містяться дані про компанії зареєстровані в системі. У компаній є посилання на таблицю Users, на власника компанії.

3. Athletes – в ній зберігається інформація про атлетів зареєстрованих на певних змаганнях, вони мають посилання на основну таблицю Users .

4. Categories – таблиця, у якій містяться дані про категорії спортсменів що доступні в рамках конкретної компанії.

5. Events – таблиця, у якій містяться дані про створенні події в середині системи. У таблиці є посилання на компанію , від якої було створено подію. Також завдяки використанню бази даних MongoDB ми маємо можливість зберігати дочірні таблиці всередині батьківської. Таблиця Events має наступні дочірні таблиці: Stages – етапи події, що зберігають в собі посилання на оператора завідуючого етапом, час старту, закінчення тощо, Tags – список рфід-тегів атлетів, що зареєстровані в конкретній події, Categories – список категорій, що задіяні в цій конкретній події.

6. UserEventResults – таблиця, у якій містяться дані про виступ конкретного атлета у конкретній події на конкретному етапі, в певній категорії.

7. Results – таблиця, у якій містяться данні про загальні результати всередині конкретної події. Має дочірні таблиці: Stages – копія етапів з поточної події, в якій у вигляді масиву зберігаються найкращі результаті в цьому етапі по категоріям та масив Results, в якому зберігаються дані про найкращіх спортсменів в рамках усієї події.

8. StageSignals – таблиця, у якій зберігаються данні перевірки з’єднання стаціонарного обладнання між собою та веб-сервером.

Для зберігання інформації на клієнтській частині буде використовуватися Redux. Redux - це управлінська бібліотека стану для JavaScript, яка використовується для розробки односторінкових програм (SPA) або веб-додатків. Вона забезпечує простий та прогнозований спосіб керування станом додатку через використання одного централізованого стору (store).

Redux спрощує управління станом додатку, роблячи його більш передбачуваним і легким для відлагодження. База даних на клієнтській частині повністю повторює схему серверної бази даних. Це відбувається для того, щоб після діставання даних вони були нормалізовані, перед поміщенням до локального клієнтського сховища Redux (Рис 5).

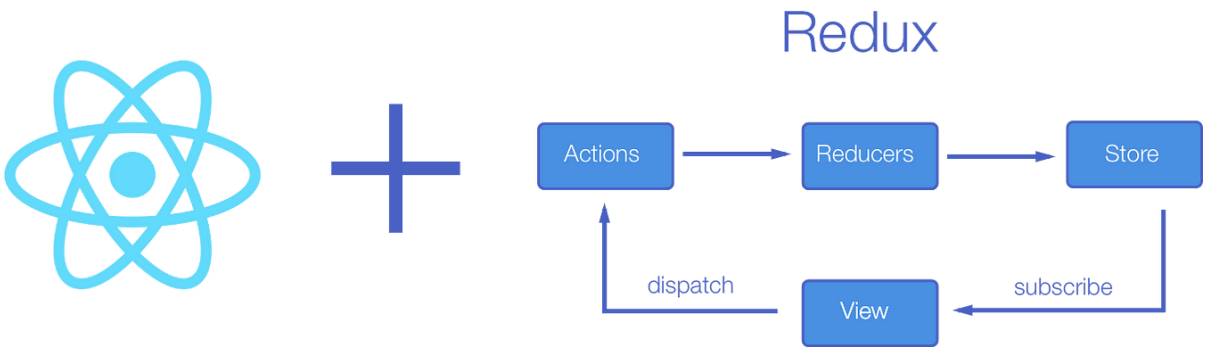

Рис. 5 *Схема роботи клієнтської бази даних*

На клієнтській частині використовуються наступні моделі:

- UserEntity
- MenuEntity
- TopEventResultEntity
- UserEventResultEntity
- CategoryEntity
- CompanyEntity
- AthleteEntity
- EventEntity

3.4 З’єднання клієнтського веб-застосунку з сервером

З'єднання клієнтського застосунку з сервером є критично важливим аспектом розробки програмного продукту. Для реалізації такого з'єднання було використанно глобальну утиліту Fetch API, яка надає інтерфейс JavaScript та TypeScript застосункам обробляти HTTP запити.

Fetch підтримує різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE і багато інших. Він також надає можливості для передачі параметрів запиту, встановлення заголовків, обробки відповіді сервера та обробки помилок. Fetch API використовується в батьківському класі клієнтських моделей Entity. Запити до Entity потрапляють через наслідуванні класси, кожен з яких репрезентує відповідну таблиці серверної бази даних. Дочірні

класи взаємодіють с сервером завдяки бібліотеці Redux-Saga, що надає подіє-орієнтований інтерфейс взаємодії із сервером.

3.5 Розробка інформаціної системи за допомогою Visual Studio Code

Розробка програмного застосунку відбувалась за допомогою Visual Studio Code - редактору вихідного коду, що був розроблений Microsoft для Windows, Linux та macOS. Позиціонується як «легкий» редактор коду для кросплатформної розробки веб- та хмарних програм.

Розробка програмного продукту була розподілена на 3 частини: проектування інформаційної системи, реалізація програмної частини та розробка інтерфейсу застосунку.

3.5.1 Проектування інформаційної системи

Інформаційна система побудована з дотриманням стандартів якісного програмування та задовольняючи всім вимогам до програмного продукту, які були визначені у пункту 3.2. Функціонал застосунку, який доступний користувачу, наведено на Use-case діаграмі (див. Рис. 6).

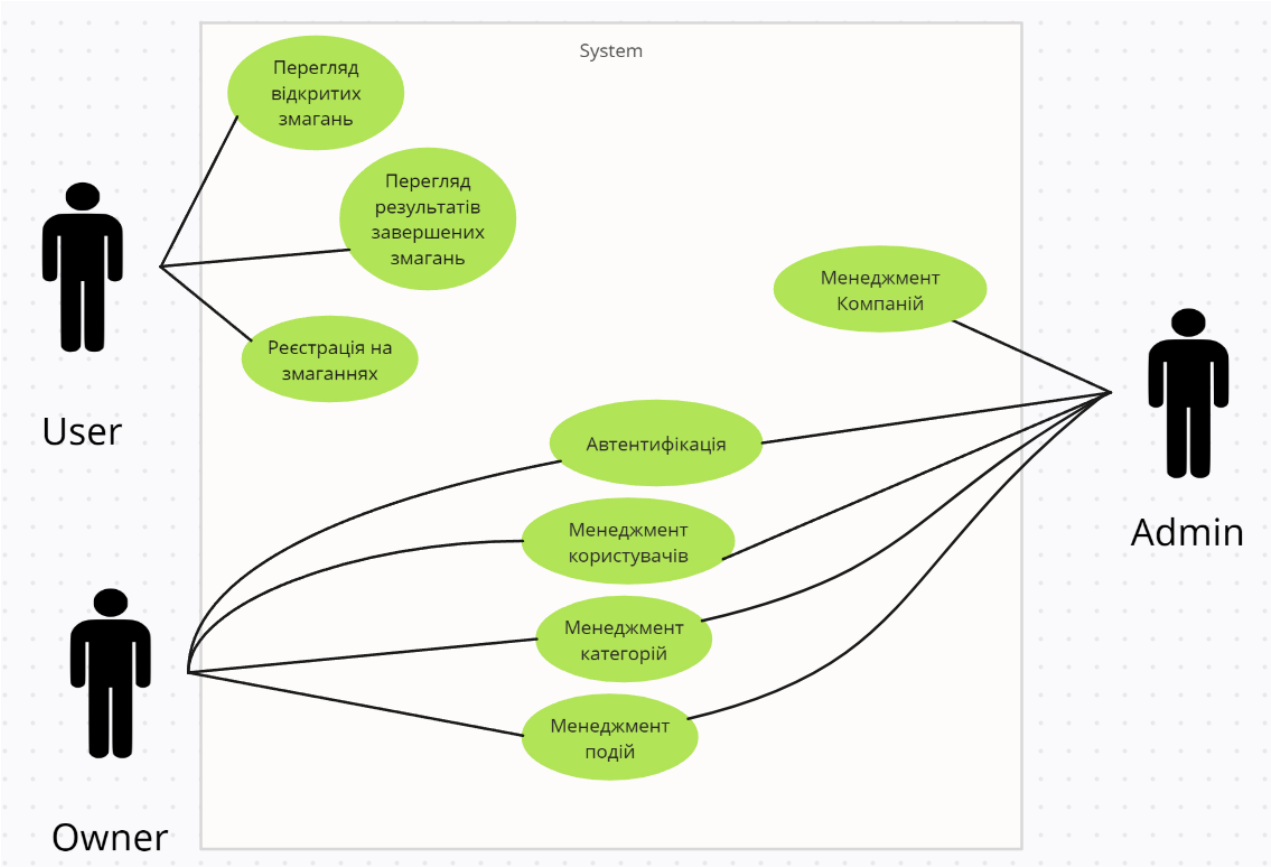

Рис. 6 *Use-case діаграма*

Use Case Diagram – це діаграма, яка використовується для моделювання та візуалізації можливих варіантів використання системи з точки зору користувача. Це один з ключових інструментів в аналізі вимог, який допомагає розібратися в тому, як система взаємодіє з її акторами (користувачами, зовнішніми системами або іншими компонентами) та які функціональні можливості вона надає.

Для того, щоб було зрозуміло, як саме ці можливості користувача були втілені у інформаційну систему, наведена діаграма класів серверної частини (див. Рис. 7).

Class Diagram – це інструмент визначення та візуалізації структури моделі системи. Вона надає статичне представлення класів, типів даних та їх взаємозв'язків. Діаграма класів використовується для визначення структури системи, опису класів і їх атрибутів, методів, спадкування, асоціацій, агрегацій та

інших відношень між класами. Вона допомагає розуміти структуру системи, визначати взаємозв'язки між класами, їх характеристики та взаємодію у межах системи.

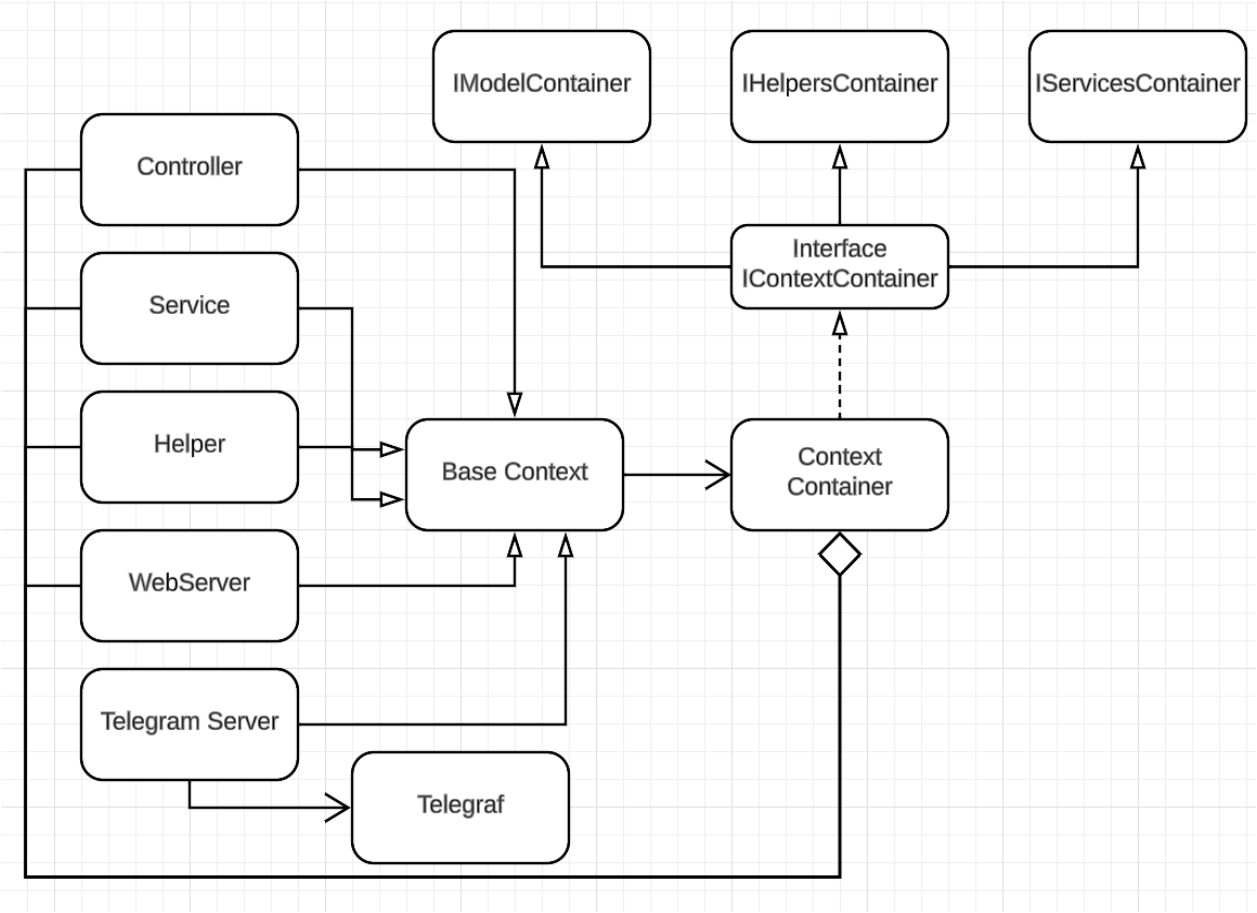

Рис. 7 *Діаграма класів серверного застосунку*

Також оскільки застосунок складається з веб-серверу та клінтського веб-застосунку, наведена діаграма класів клієнтського застосунку (див Рис. 8), що демонструє схему взаємодії класів на веб-сторінці на прикладі сторінки

логінації.

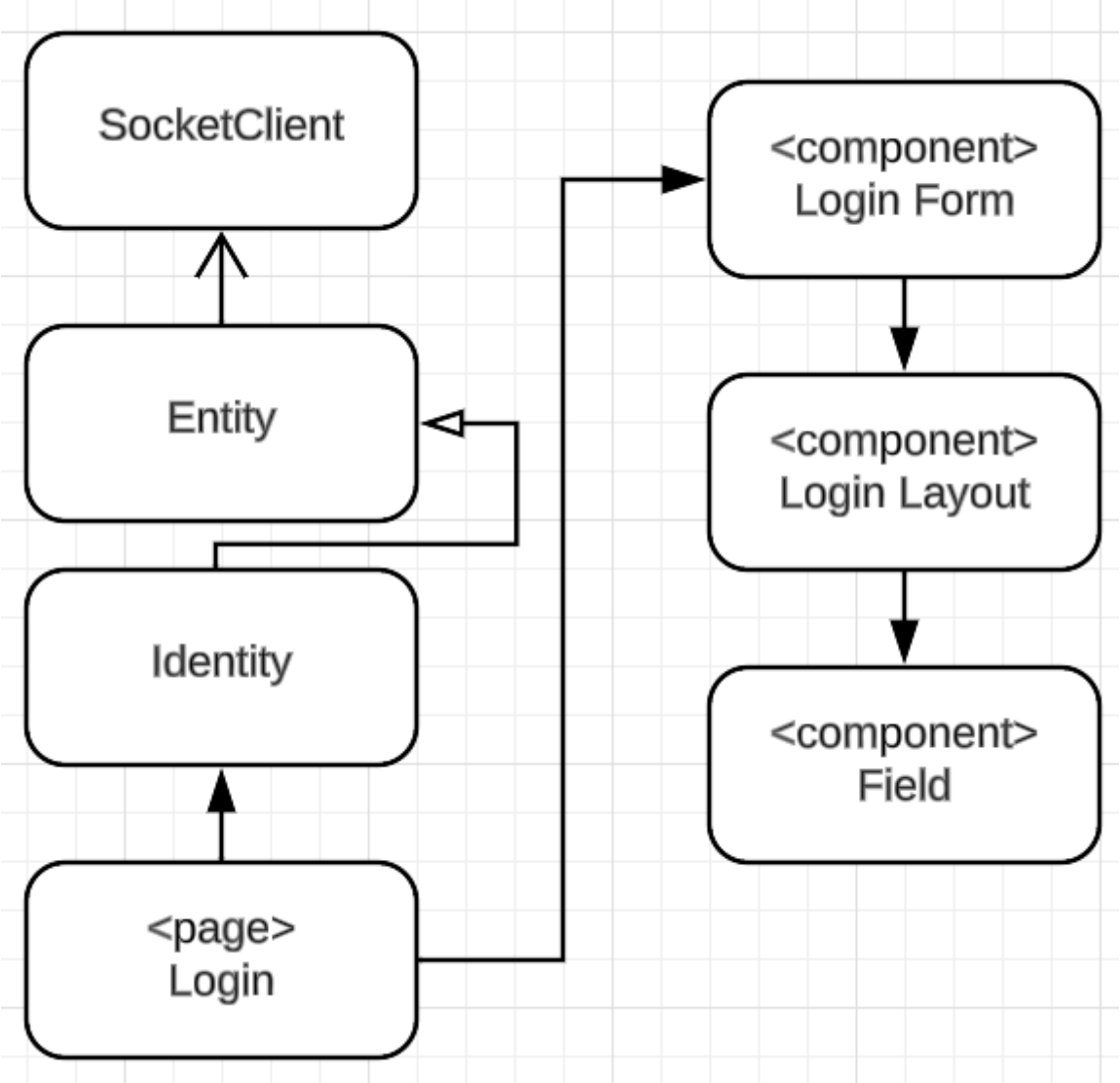

Рис. 8 *Діаграма класів клієнтського застосунку*

Більш детальний розгляд цих класів та опис деяких методів розглянуто далі (див. п. 3.4.2).

3.5.2 Реалізація програмної частини серверного застосунку

1. Клас BaseContext.ts головний клас від якого успадковуються інші класи, які регіструються у dependency injection контейнері.

2. Клас WebServer.ts слугує точкою входу до програми, та реалізує запуск веб-серверу на певному порті, з певною адресою, та підключення

різноманітних проміжних бібліотек, що обгортають стандартні запити, надаючи новий функціонал.

Лістинг 1 – *приклад функції start, що запускає веб-сервер, та під'єднує проміжні бібліотеки*

```
    public async start() {
     if (!config.jest) {
                await this.app.prepare();
            }
    this.expressServer = express();
    this.expressServer.use(compression());
    this.expressServer.use(cookieParser());
            this.expressServer.use(bodyParser.json({ limit:
'30mb' }));
            this.expressServer.use(bodyParser.urlencoded({
limit: '30mb', extended: false, parameterLimit: 50000 }));
    this.expressServer.use(requestMiddleware);
    this.expressServer.use(this.acl);
    this.expressServer.use(scopePerRequest(container));
            const files = 'controllers/**/*.' + (config.dev
? 'ts' : 'js');
            this.expressServer.use(loadControllers(files, {
cwd: __dirname }));
            this.expressServer.get('/socket.io', (req, res)
=> {
                res.send('ok');
            });
            this.expressServer.get('*', async (req, res) =>
{
                const parsedUrl = parse(req.url, true);
                handle(req, res, parsedUrl);
            });

            this.httpServer = this.expressServer
                .listen(config.port)
                .on('listening', () => {
                    console.log(`Web  server  listening  on
localhost:${config.port}`);
                }).on('error', err => {
                    console.log(`Error        on        the
server:${err}`);
                });
     }
```

3. TelegramServer.ts – клас, що запускає телеграм бота, та оброблює команди, що надходять від користувачів. В Лістингу 2 наведено приклад

роботи функції відображення деталей події, при натисканні відповідної кнопки в телеграм боті. Дані отримуються з спільного для всіх існуючих класів контексту. Якщо запит прийшов від користувача, що не обрав подію, результати якої він хоче переглянути, функція повертає перекладений, в залежності від мови системи телефону, рядок з проханням спочатку обрати бажану подію. Якщо подія була обрана заздалегідь, починається обробка події, зі збором всієї необхідної інформації у відповідь, у вигляді одного великого рядку.

Лістинг 2 – *приклад обробки дії користувача (відображення деталей події)*

```
    private showDetailEventInfo = async (ctx: TContext) => {
            const { config } = this.di;
            const { session: { timeZone }} = ctx;
            if (ctx.session.event) {
                const { categories, coverImage, description
} = ctx.session.event;
                const fileName = './uploads' +  coverImage;
                const   isImageExist   =   coverImage   &&
!coverImage.includes('undefined')                          &&
fs.existsSync(fileName);
                const       {       message       }       =
this.buildButtons(categories);
                let    caption    =    (description    &&
(description.substr(0,                500)                +
'...\n\n'))+ctx.i18n.t('available-
categories')+'\n'+message;

                const  header  =  ctx.i18n.t('choose-event-
btn', {
                    eventName: ctx.session.event.eventName,
                    startTime:
moment.tz(ctx.session.event.startEvent,
timeZone).locale(ctx.i18n.locale()).format('HH:mm,  MMM  DD,
YYYY'),
                    eventStatus:
statusesEmoji[ctx.session.event.status]
                });

                caption = header + '\n\n' + caption;

                if (isImageExist) {
                    let url = config.baseUrl + (config.port
&& config.dev ? ':' + config.port : '');
```

```
                    return ctx.replyWithPhoto(
                        { url: url + '/file' + coverImage },
                        {  caption:  caption,  parse_mode:
'HTML' },
                    );
                } else {
                    return   ctx.replyWithMarkdown(caption,
await this.mainMenu(ctx));
                }
            } else {
                return  ctx.reply(ctx.i18n.t('choose-event-
before'));
            }
        }
```

4. UserModel.ts – клас для взаємодії серверного застосунку з базою даних з таблицею Users з ціллю додавання, зміни та видалення даних з колекції

5. EventModel.ts – клас для взаємодії серверного застосунку з базою даних з таблицею Events з ціллю додавання, зміни та видалення даних з колекції. Завдяки використанню MongoDB як бази даних, ми маємо можливість спрощувати відносини «багато до багатьох», створюючи вкладені сутності. Клас події має в собі вкладену сутність, що репрезентує стадії події. У Лістингу 3, наведено приклад репрезентації вкладеної сутності.

Лістинг 3 – *приклад створення вкладеної сутності з використанням Typegoose*

```
export class EventStage {

    public _id: Schema.Types.ObjectId;

    @prop()
    public stageName: string;

    @prop()
    public startTime: number;

    @prop()
    public finishTime: number;

    @prop({ default: 0 })
    public pause: number;

    @prop()
```

```
    public pausedAt: number;

    @prop()
    public pausedStartAt: number;

    @prop({ default: StageStatus.OPENED })
    public status: StageStatus;

    @prop({ ref: () => UserSchema })
    public userId: Ref<UserSchema>; // oper of stage

    @prop()
    public delay: number;

    @prop()
    public categoryDelay: number;

    @prop()
    public description: string;

    @prop()
    public avatar: string;

    @prop()
    public avatarOrigin: string;

    @prop()
    public startPoint: string;

    @prop()
    public endPoint: string;

    @prop({ default: false })
    public ignoreResults: boolean;

    @prop({default: false})
    public massStart: boolean;

    @prop()
    public createdAt: number;

    @prop()
    public updatedAt: number;

    public currentResult: any;

    public athlete: any;
}
```

6. CategoriesModel.ts – клас для взаємодії серверного застосунку з базою даних з таблицею Categories з ціллю додавання, зміни та видалення даних з колекції.

7. CompanyModel.ts – клас для взаємодії серверного застосунку з базою даних з таблицею Companies з ціллю додавання, зміни та видалення даних з колекції.

8. AthleteModel.ts – клас для взаємодії серверного застосунку з базою даних з таблицею Athletes з ціллю додавання, зміни та видалення даних з колекції.

9. SocketServer.ts – клас, що запускає сокет-сервер та оброблює сокет-з'єднання та реалізує відправку даних через сокети.

Лістинг 4 – *приклад відправки даних через сокети*

```
private sendToKey(entityName: string, msg: any, key: string) {
        const rawJSON = JSON.parse(JSON.stringify(msg.data));
        const hash = md5(JSON.stringify(camelizeKeys(rawJSON)).toLowerCase());
        const buffer = this.connects[key]['buffer'];
        if (!buffer.hasOwnProperty(hash)) {
            buffer[hash] = { roomName: entityName, msg };
        }
        msg['hash'] = hash;
        msg['entity'] = entityName;
        // console.info('Socket request to client=', key,'hash=', hash, 'entity=', entityName);
        this.connects[key]['socket'].emit('data', msg);
                                }
```

10. UserService.ts – клас, що використовує клас UserModel.ts , передаючи в нього необхідні данні для внесення змін в базу даних.

11. EventService.ts – клас, що використовує клас EventModel.ts , передаючи в нього необхідні данні для внесення змін в базу даних. В сервісі подій реалізовано велику кількість найважливіших функцій системи. Однією з таких функцій є "findActual", яка повертає актуальні (повністю заповнені, та відкриті

для реєстрації) події, для відображення їх на головній, публічній сторінці сайту. Функція наведена у Лістингу 5.

Лістинг 5 – *функція розрахунку актуальних подій*

```
public async findActual() {
        const { EventModel } = this.di;

        const event = await EventModel.aggregate([
            {
                // $match: {
                //           $nin:  [EventStatus.DRAFT,
EventStatus.CLOSED]
                //   }
                $match: {
                    $expr: {
                        $and : [{
                            $not: {
                                $in:   ['$EventStatus',
[EventStatus.DRAFT, EventStatus.CLOSED]]
                            }}, {
                            $ne: ["$testMode", true]
                        }]
                    }
                }
            },
            {
                $addFields: {
                    "priority": {
                        $switch: {
                            "branches": [
                                {  "case":  {  $eq:  [
"$status", EventStatus.OPENED] }, "then": 3 },
                                {  "case":  {  $eq:  [
"$status", EventStatus.STARTED] }, "then": 2 },
                                {  "case":  {  $eq:  [
"$status", EventStatus.FINISHED] }, "then": 1 }
                            ],
                            default: 0
                        }
                    }
                }
            },
            {
                $sort: {
                    priority: -1,
                    updatedAt: -1
                }
            },
```

```
            {
                $project: {
                    priority: 0,
                }
            },
            {
                $lookup: {
                    "from": "companies",
                    "localField": "companyID",
                    "foreignField": "_id",
                    "as": "companyID"
                }
            },
            {
                $unwind: "$companyID"
            },
            {
                $limit: 1
            },
        ]);
        if (event) {
            return event[0]
        }
        return null;
    }
```

Наступною важливою функцією з сервісу подій, є функція “findCurrent”. Ця функція використовується керівниками компаній, аби автоматично вибирати поточну подію компанії і наповнювати певні сторінки відповідними даними по події. Функція наведена в Лістингу 6.

Лістинг 6 – *функція визначення поточної події*

```
    public async findCurrent(companyId = null) {
            const { EventModel } = this.di;
            const query = companyId ? {
                $match: {
                    companyID:  new  mongoose.Types.Ob-
jectId(companyId),
                    status: { $ne: EventStatus.CLOSED }
                }
            } : {
                $match: {
                    status: { $ne: EventStatus.CLOSED }
                }
            }
```

```
                const event = await EventModel.aggregate([
                    query,
                    {
                        $project: {
                            _id: 1,
                            eventName: 1,
                            slug: 1,
                            timeDifference: {
                                $abs: {
                                    $subtract:          [new
    Date().getTime(), "$startTime"]
                                }
                            }
                        }
                    },
                    {
                        $sort: {
                            timeDifference: 1
                        }
                    },
                    {
                        $limit: 1
                    }
                ]);

                return event.length ? event[0] : null;
        }
```

Ще двома важливими функціями з сервісу подій є функції, що підготовлюють дані для відображення їх у телеграм боті, по запиту користувачів. У Лістингу 7 наведена функція, що знаходить актуальну подію при запиті з телеграма та виводить її у чат-бот, відповідному користувачу. На Лістингу 8 наведена функція, що повертає форматовані повні данні події, для відображення їх на відповідний запит певному користувачу у телеграм-боті.

Лістинг 7 – *функція пошуку актуальної події для чату телеграм*

```
        public    async    findForTelegramByStatus(status:
EventStatus | EventStatus[], companyID?: string) {
                const { EventModel } = this.di;
                status = Array.isArray(status) ? status :
[status];
```

```
                const match: any = {
                    status: {$in: status},
                    testMode: {$ne: true}
                }
                if(companyID){
                    match.companyID          =          new
mongoose.Types.ObjectId(companyID);
                }
                return EventModel.aggregate([
                    {
                        $match: { ...match }
                    },
                    {
                        $project: {
                            _id: 1,
                            eventName: 1,
                            slug: 1,
                            startTime: 1,
                            status : 1,
                            timeDifference: {
                                $abs: {
                                    $subtract:         [new
Date().getTime(), "$startTime"]
                                }
                            }
                        }
                    },
                    {
                        $sort: {
                            timeDifference: 1
                        }
                    }
                ]);
            }
```

Лістинг 8 – *функція форматування повних даних про подію для чату телеграм*

```
    public async getDataForTelegram(eventSlug: string)
: Promise<ITEvent> {
                const  {  EventService,  CompanyModel  }  =
this.di;
                const         event         =         await
EventService.findBySlug(eventSlug);
                if (!event) { return null; }
```

```
            const company = await
CompanyModel.findById(event.companyID);
            const coverImage =
'/companies/'+company.slug+'/events/'+event.slug+'/avat
ar/'+event.avatar;
            let btnGroup = 1;
            let isOdd = event.categories.length % 2 ===
0 ? true : false;
            const categories: ITEventCategory[] =
event.categories.reduce((a, item:any, i) => {
                const { categoryName, description,
slug, _id } = item.categoryId;
                if (btnGroup == 2 || ( !isOdd && i ==
0)) {
                    btnGroup += 1;
                }
                a.push({
                    slug,
                    name: categoryName,
                    order: item.order,
                    description,
                    group: btnGroup,
                    id: _id.toString(),
                    action: 'category['+slug+']',
                });
                return a;
            },new Array<ITEventCategory>());
            btnGroup = 1;
            isOdd = event.stages.length % 2 === 0 ? true
: false;
            const stages = event.stages.reduce((a,
stage, i) => {
                const { _id, stageName, description,
startTime, avatar, massStart } = stage;
                if (btnGroup == 2 || ( !isOdd && i ==
0)) {
                    btnGroup += 1;
                }
                a.push({
                    id: _id.toString(),
                    name: stageName,
                    order: i,
                    description,
                    group: btnGroup,
                    startTime: startTime,
                    massStart,
                    action: 'stage['+stage._id+']',
```

```
                    avatar:
'/companies/'+company.slug+'/events/'+event.slug+'/stag
es/avatar/'+avatar
                })
                return a;
            }, new Array<ITEventStage>());
            return {
                slug: event.slug,
                eventId: event._id.toString(),
                eventName: event.eventName,
                startEvent: event.startTime,
                status: event.status,
                startRegistration:
event.startRegistration,
                startAwards: event.startAwards,
                categories,
                stages,
                coverImage,
                description: event.description,
                isCheckRequired: event.isCheckRequired,
                paymentDetails: event.paymentDetails,
                telegram: event.telegram,
            }
    }
```

Одним з найважливіших аспектів при роботи з базами даних є контроль цілісності. Так як сама MongoDB не надає інструментів для такого контролю, відповідальність за правильне видалення даних лежить на плечах програмістів. У Лістингу 9 наведено приклад видалення вкладеної сутності “Категорії” з події .

Лістинг 9 – *видалення елементу категорії з події*

```
    public async deleteFromEvent(body) {
            const { EventModel } = this.di;

            let event = await
EventModel.findOneAndUpdate({ slug: body.eventSlug }, {
           $pull: {
                    categories: { categoryId :
body.categoryId }
                }
            }, { new: true });
```

```
            event.categories.forEach(function (item,
index) {
                item.order = index + 1;
            });

            return event.save();
    }
```

12. CompanyService.ts – клас, що використовує клас CompanyModel.ts, передаючи в нього необхідні дані для внесення змін в базу даних. Сервіс використовується для видалення, редагування та додавання компаній. У Лістингу 10 та Лістингу 11 наведено приклади функцій по внесенню змін у профіль компанії та видалення компанії відповідно.

Лістинг 10 – *приклад функції внесення змін у профіль компанії*

```
public async changeProfile(body) {
        const { CompanyModel, UserModel, telegram } =
this.di;
        const company = await Company-
Model.findById(body.id);

        const telegramBot = company?.telegramBot;
        // const telegramApiKey = company?.telegramA-
piKey;
        const isBotStarted = company?.isBotStarted;
        if(body.ownerId !== company.ownerId.toString())
{
            const userNew = await Us-
erModel.findById(body.ownerId);
            const userOld = await Us-
erModel.findById(company.ownerId);
            userOld.companies.pull(company);  //=
userOld.companies.filter(company => company.own-
erId.toString() !== userOld._id.toString());
            userNew.companies.push(company);
            userOld.save();
            userNew.save();
        }

        company.set(body)

        const companySave = await company.save();
        return companySave;
    }
```

Лістинг 12 – *приклад функції видалення компанії з бази даних*

```
public async deleteCompany(companyId: string) {
        const { UserModel, log } = this.di;
        const company = await this.findCompanyById(com-
panyId);

        if (!company) {
            return  Promise.reject('common:company-not-
found');
        }
        const  owner  =  await  UserModel.findById(com-
pany.ownerId);
        if (!owner) {
            await company.remove({});
        } else {
            const  companyIdx  =  owner.companies.find-
Index(c => c._id === company._id);
            delete owner.companies[companyIdx]
            owner.save();
            await company.remove({});
        }
        log.info(`Company  "${company.companyName}"  de-
leted`)
        return null;
    }
```

13. CategoryService.ts – клас, що використовує клас CategoryModel.ts , передаючи в нього необхідні дані для внесення змін в базу даних. Сервіс використовується для редагування, видалення та додавання категорій в базу даних для обраних подій. У Лістингу 13 наведено приклад функції витягу категорій з бази даних за відповідними фільтрами, для відображення їх в таблиці.

Лістинг 13 – *приклад функції витягу категорій з бази даних*

```
public page(pageNum = 1, perPage = PAGE_SIZE_10, filter:
any = null, sorts: any = null, count = 0) {
        const { CategoryModel } = this.di;
        let query = {} as any;
        const querySearch = {} as any;

        if (filter) {
            if                  (hasOwnProperty(filter,
'nameOrDescription')                                    &&
!isEmpty(filter.nameOrDescription)) {
```

```
                querySearch['$or'] = [
                    {
                        categoryName: {
                            $regex:
filter.nameOrDescription, $options: 'i',
                        },
                    },
                    {
                        description: {
                            $regex:
filter.nameOrDescription, $options: 'i',
                        },
                    },
                ];
            }

            if (hasOwnProperty(filter, 'companyId') &&
!isEmpty(filter.companyId)) {
                query['companyId']   =   {   $eq:   new
mongoose.Types.ObjectId(filter.companyId) }
            }
        }

        query = [
            { $match: query },
            { $match: querySearch },
        ];

        const countItems = concat(cloneDeep(query), [{
$count: 'countItems' }]);
    // 2 : sort
        const sort = {} as any;
        if (sorts) {
            if (sorts.sort === Sort.ASC || sorts.sort
=== Sort.DESC) {
                sort[sorts.field] = sorts.sort;
                sort['_id'] = Sort.ASC;
            }
        }

        if (!isEmpty(sort)) {
            query = concat(query, [{ $sort: sort }]);
        }

        // 3 : limit
        query = concat(query, [
            { $skip: (pageNum - 1) * perPage },
            { $limit: perPage },
```

```
        ]);

        if (!count || count === 0) {
            return CategoryModel.aggregate(countItems)
                .then((cnt) => CategoryModel
                    .aggregate(query)
                    .collation({ locale: 'en', strength:
2 })
                    .then((items) => {
                        return { items, count: cnt[0] ?
cnt[0].countItems : 0 };
                    })
                );
        }
        return CategoryModel.aggregate(query)
            .collation({ locale: 'en', strength: 2 })
            .then((items) => {
                return ({ items, count });
            });
    }
```

14. AthleteService.ts – клас, що використовує клас AthleteModel.ts , передаючи в нього необхідні данні для внесення змін в базу даних. Цей сервіс використовується для таких дій як: реєстрація атлета, оновлення даних атлета, присвоювання номерку та RFID-тегу. Також через цей сервіс атлети отримують статус “Verified” після перевірки їх даних керівниками компаній, та відсилаються нотифікації про кожного зареєстрованого або веріфікованого атлету до телеграм-чату боту, власнику компанії. У Лістингу 14 наведений приклад роботи функції з веріфікації атлета.

Лістинг 14 – *функція веріфікації атлета*

```
public async verifyAthlete(athleteData: {
        event: string;
        user: string;
        status: string;
    }, approverId: string) {
        const {AthleteModel, UserModel, telegram, log,
EventService} =
            this.di;
        const event = await EventService.findById(ath-
leteData.event);
        if (event) {
```

```
            for (let stage of event.stages) {
                await this.clearAthletesQueueRedis(
                    athleteData.event,
                    stage._id.toString(),
                );
            }
        }

        const user = await UserModel.findOneAndUpdate(
            {_id: athleteData.user},
            {
                $addToSet:        {events:        [ath-
leteData.event]},
            },
        );

        let athlete = await AthleteModel.findOne({
            event: athleteData.event,
            user: athleteData.user,
        });

        const isVerified = athlete.isVerifiedEmail; //
if athlete has already received the verified email
        if (
            !athlete.isVerifiedEmail &&
            athlete.status == AthleteStatus.APPLICANT &&
            athleteData.status  ==  AthleteStatus.VERI-
FIED
        ) {
            athlete.isVerifiedEmail = true;
        }

        if (athlete.status == AthleteStatus.APPLICANT &&
            athleteData.status  ==  AthleteStatus.VERI-
FIED) {
                const    approver    =    await    Us-
erModel.findById(approverId);
                this.sendAthleteVerifiedToManagersTg(
                    athlete.event,
                    approver,
                    {
                        firstName: user.firstName,
                        lastName: user.lastName,
                    },
                );
    }

        athlete.set(athleteData);
```

```
            await athlete.save();
            await athlete.populate({
                path: 'user',
                select: '-password -token -secret',
            });

            if (!isVerified && athlete.isVerifiedEmail) {
                //this.notifyAthleteAboutVerifica-
tion(oldAthlete, user);
                if (user.telegram_id) {
                    telegram.sendMessageAboutVerification-
ToPrivateChat(
                        user.telegram_id,
                        athlete._id.toString(),
                    );
                }
            }

            return Promise.resolve(athlete);
        }
```

Функція приймає параметрами об’єкт athleteData , що містить у собі властивості: статус, подія на якій зареєструвався атлет а також унікальний номер користувача з колекції користувачів. Другий параметр функції це унікальний номер користувача, що веріфікував атлета, тобто користувача, який надіслав запит.

15. Класи типу Controller, що надають певні Api-поінти для взаємодії з сервером, та передають отриманні в запиті дані у класи сервісів.

Також на сервері використовуються певні валідатори даних, що обгортають апі-поінти серверу завдяки використанню декораторів, та реалізують контроль даних, та валідацію доступу до певних апі-поінтів.

1. emailValidator.ts – клас, що реалізує валідацію шаблонів email-повідомлень. Цей клас є важливим компонентом для перевірки валідності шаблону відправки пошти. Він містить властивості, що відповідають параметрам доступним в шаблонах:

1) title (назва):

- exists (існує): Перевіряє наявність поля.

- notEmpty (не порожнє): Вимагає, щоб поле не було порожнім.
- isString (є рядком): Вимагає, щоб значення було рядком.
- errorMessage (повідомлення про помилку): Вказує повідомлення про помилку у разі невідповідності вимогам.
- isLength (довжина рядка): Перевіряє довжину рядка.
- errorMessage (повідомлення про помилку): Вказує повідомлення про помилку у разі невідповідності вимогам.
- options (опції): Налаштування, такі як максимальна довжина рядка (у цьому випадку - максимум 50 символів).

2) description (опис):

- exists (існує): Перевіряє наявність поля.
- notEmpty (не порожнє): Вимагає, щоб поле не було порожнім.
- isString (є рядком): Вимагає, щоб значення було рядком.
- errorMessage (повідомлення про помилку): Вказує повідомлення про помилку у разі невідповідності вимогам.

Цей об'єкт використовується для перевірки вхідних даних - назви та опису. Він допомагає гарантувати, що вони відповідають встановленим вимогам перед їх подальшим використанням чи зберіганням, зменшуючи ризик отримання неправильних або недостовірних даних у програмі чи системі.

Лістинг 15 – *валідатор шаблонів email-повідомлень*

```
export const emailVal: Record<string, ParamSchema> =
{
    'title': {
        exists: true,
        notEmpty: true,
        isString: true,
        errorMessage: 'required ',
        isLength: {
            errorMessage: 'max 50 characters',
            options: { max: 50 }
        }
    },
```

```
        'description': {
            exists: true,
            notEmpty: true,
            isString: true,
            errorMessage: 'required ',
        },
    };
```

3.5.3 Реалізація програмної частини клієнтського застосунку

1. Entity.ts – батьківський клас для класів UserEntity, EventEntity тощо, що являє собою точку з'єднання клієнтського застосунку з сервером.

Лістинг 16 - *приклад функцій, що відпрацьовують різні методи відправки запитів*

```
public xSave = (uri: string, data: any = {}) => {
        return   this.actionRequest(uri,   CRUD.UPDATE,
HTTP_METHOD.POST, data);
    }

    public xRead = (uri: string, data: any = {}, method:
HTTP_METHOD = HTTP_METHOD.GET) => {
        return    this.actionRequest(uri,    CRUD.READ,
method, data);
    }

    public xDelete = (uri: string, data: any = {}) => {
        return   this.actionRequest(uri,   CRUD.DELETE,
HTTP_METHOD.DELETE, data);
   }
```

2. Файл reducers.ts з Redux редьюсерами. В Лістингу 6 наведено функію що є редуктором, яка відповідає за обробку змін в стані (state) для управління сутностями (entities) в програмі. Цей редуктор обробляє різні типи дій (crud) для маніпулювання сутностями у стані програми. Він виконує дії, такі як видалення об'єктів, очищення списків сутностей або оновлення їх даними, що надходять з дій (actions). Кожен case у switch-блоку відповідає певному типу дії (CREATE, UPDATE, DELETE, CLEAR, CLEAR_ALL) та виконує відповідні маніпуляції зі списками сутностей в стані програми.

Лістинг 17 - *основний редьюсер проекту*

```
function entities(state = initialEntities, action: any)
{
    if ('glob' in action) {
        const {
            glob: { crud, entity },
        } = action;

        switch (crud) {
        case IMethod.DELETE:
            if            (action.response            &&
action.response.entities) {
                let list = state.get(entity.entityName);

                if (list) {
                    const       deletedObjectId       =
Object.keys(action.response.entities[entity.entityName]
)[0];
                    list                                =
list.remove(deletedObjectId);
                    state                               =
state.set(entity.entityName, list);
                }
            }
            break;
        case IMethod.CLEAR:
            if (entity && state.has(entity.entityName))
{
                state   =   state.set(entity.entityName,
fromJS({}));
            }
            break;
        case IMethod.CLEAR_ALL: {
            state = initialEntities;
            break;
        }
        default:
        case IMethod.UPDATE:
            if            (action.response            &&
action.response.entities) {
                const {
                    response: { entities },
                } = action;
                if (entities) {

Object.keys(entities).map((entityName) => {
```

```
                            let            list            =
state.get(entityName);
                            if (list && list.size > 0) {

Object.keys(entities[entityName]).map(
                                    (id)    =>    (list    =
list.remove(id))
                                );
                            }
                            state  =  state.set(entityName,
list);
                        });
                        state                                =
state.mergeDeep(fromJS(entities));
                    }
                }
                break;
            }
        }
        return state;
}
```

3. Файл action.ts. В Лістингу 18 наведена функція-декоратор, що репрезентує собою наявні в різних Entity методи, які забезпечують зв’язок із веб-сервером.

Лістинг 18 - *Action декоратор*

```
const action = () => {
    return (target: any, propertyKey: string) => {
        const entityName = target.construc-
tor.name;
        const entityItem = entityName in En-
tity.mActions ? Entity.mActions[entityName] : {};
        if (!(propertyKey in entityItem)) {
        entityItem[propertyKey] = {
            watcher: propertyKey,
            actionFunc: (data) => actions.ac-
tion(propertyKey, data),
            isAdded: false,
        };
        }
        Entity.mActions[entityName] = entityItem;
    };
};
```

4. Файл saga.ts. В Лістингу 19 наведено приклад функції, яка використовується для визначення спостерігачів (саг) для об'єктів у програмі, які очікують певні події (actions). Вона створює безкінечний цикл, який чекає на виникнення певних подій та виконує відповідні дії над отриманими даними з сервера.

Лістинг 19 - *Saga декоратор*

```
const saga = (entityName: string, actions: string[] =
[]) => {
    const entity = container.resolve(entityName);
    return (constructor: Function = null) => {
        if (entity) {
            const entityName = entity.constructor.name;
            actions.forEach((actionName) => {
                if (entityName in Entity.mActions) {
                    const            acts            =
Entity.mActions[entityName];
                    if (actionName in acts) {
                        const          act           =
Entity.mActions[entityName][actionName];
                        if (!act.isAdded) {
                            //console.log('Add:   '   +
entityName + '.' + actionName + '()');
                            // @ts-ignore
                            const     watcherFunc     =
entity[act.watcher].bind(entity);
                            let func = function* () {
                                while (true) {

//console.log('Start: ' + entityName + '.' + actionName
+ '()');
                                    const data = yield
take(actionName);
                                    delete data.type;

//console.log('Call: ' + entityName + '.' + actionName +
'()');
                                    yield
fork(watcherFunc, data);
                                }
                            };
                            //   set   name   to   above
function. use for debug goals
                            // func = new Function(
```

```
                                //          `return function
(call) { return function ${entityName}_${actionName} ()
{ return call(this, arguments) }; };`
                                //
)()(Function.apply.bind(func));

Entity.mSagas.push(fork(func));
                                act.isAdded = true;
                            }
                        }
                    } else {
                        throw       new       Error(`Action
[${actionName}] does not belong to the entity`);
                    }
                });
            } else {
                Entity.mSagas = [];
            }
        }
};
```

5. SocketClient.ts – клас, що реалізує під’єднання клієнтського застосунку до серверу, шляхом сокетів. Цей клас використовується для отримання оперативних даних від сервера, та оновлення зовнішнього вигляду сторінок у реальному часі.

В Лістингу 20 наведено метод, що реалізує під’єднання клієнтської частини до серверу, шляхом сокетів. Ця функція починає з'єднання з сервером WebSocket, використовуючи інформацію про ідентифікацію користувача (identity: IIdentity). Вона перевіряє наявність з'єднання (this.socket) та прав доступу за допомогою об'єкта guard. Якщо з'єднання відсутнє і дозвіл на використання WebSocket є, то формується URL сервера WebSocket (залежно від режиму розробки) та ініціюється з'єднання через бібліотеку Socket.IO. Після встановлення з'єднання обробляються події "connect" (подія успішного з'єднання) та 'data' (прийом даних через WebSocket). Ця функція дозволяє клієнту з'єднатися з сервером через WebSocket для обміну даними в реальному часі.

В Лістингу 21 наведено приклади методів, що реалізують обробку подій з Лістингу 20. Функція onSocketData використовується для обробки отриманих даних через WebSocket та виклику відповідних функцій, що обробляють ці

дані. При наявності певного хешу в надісланих даних, вона викликає функцію acceptRequest, яка підтверджує збереження даних з цим хешем на сервері.

Функція acceptRequest отримує значення ключа з локального сховища браузера та відправляє сигнал на сервер WebSocket з отриманим хешем та ключем для підтвердження збереження даних на сервері.

Лістинг 20 – *встановлення сокет-з'єднання із сервером*

```
public start(identity: IIdentity) {
        const guard = initGuard(identity);
        if  (!this.socket  &&  guard?.allow('socket/*',
GRANT.READ)) {
            const url = !IS_DEV_MODE ?
                SOCKET_HOST :
                SOCKET_HOST  +  (SOCKET_PORT  ?  ':'  +
SOCKET_PORT : '');
            this.socket = io(url, {
                query: {
                    sessionKey: this.sessionKey
                }
            });
            this.socket.on("connect", this.onConnect);
            this.socket.on('data', this.onSocketData);
        }
    }
```

Лістинг 21 – *отримання даних, та додавання їх до локального сховища*

```
private onSocketData(data) {
        Object.keys(this.rooms).map(entityCallback => {
            this.rooms[entityCallback](data);
        });
        if (data.hash) {
            this.acceptRequest(data.hash);
        }
    }
public acceptRequest(hash: string) {
        const                    key                    =
window.localStorage.getItem('sessionKey');
        console.log('Accept Request: key=',key, 'hash=',
key);
        this.socket.emit('accept', { hash, key });
    }
```

3.5.4 Розробка інтерфейсу застосунку

Для розробки клієнтської частини використовувались бібліотеки React.js та Tailwind.css.

Tailwind CSS - це потужний інструмент для створення веб-інтерфейсів, який базується на наборі написаних заздалегідь класів. Ця бібліотека CSS надає можливість швидко створювати і налаштовувати вигляд елементів веб-сторінок без необхідності написання власних CSS-стилів.

Основна ідея Tailwind CSS полягає у тому, щоб використовувати набір малих класів, які надають стилі для різних елементів, наприклад, відступи, розміри шрифтів, кольори та інші властивості. Замість написання CSS-коду з нуля, ви можете використовувати ці класи безпосередньо в HTML-розмітці, швидко конструюючи потрібний вам вигляд.

Технологія Tailwind дозволяє розробникам створювати різноманітні інтерфейси, використовуючи гнучкий інструментарій, зручний для швидкої роботи. Це дозволяє прискорити процес розробки веб-сайтів і додатків, забезпечуючи водночас високий рівень контролю над стилями та виглядом компонентів.

Tailwind CSS відрізняється від інших CSS-фреймворків тим, що не має готових компонентів і сконцентрований на створенні стилів шляхом комбінування класів.

Розроблений інтерфейс є зрозумілим та простим у використанні.

На головній сторінці всі користувачі можуть побачити діючу подію та натиснувши перейти на сторінку самої події, де в них є можливість переглянути детальну інформацію про подію, зареєструватися в події або подивитись результати, якщо це змагання вже було завершено (див Рис. 9) Також користувач може побачити дату на яку заплановано подію, скільки людей зареєстровано, та окрему статистику кількості учасників по всіх категоріях події. При

натисканні кнопки реєстрації, користувача буде направлено на сторінку реєстрації.

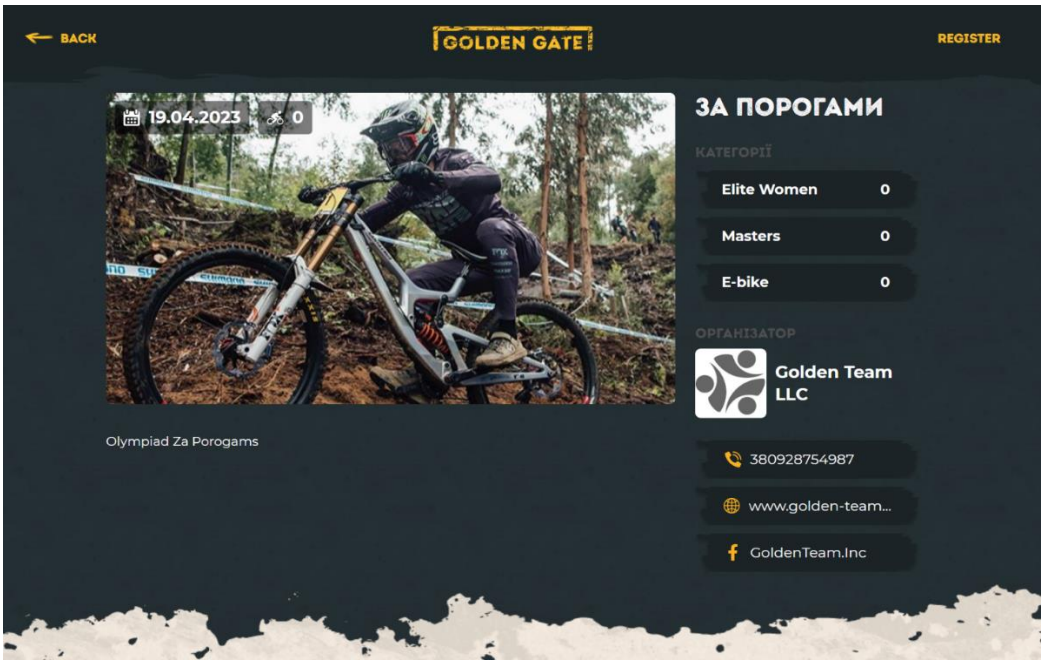

Рис. 9 – приклад публічної сторінки змагання

Сторінка реєстрації містить форму для заповнення обов’язкових даних для проходження реєстрації (Рис 10), а також згенерований QR-Code, що

дозволяє користувачам, використовуючи телефонний додаток Telegram, швидко перейти
до існуючого бота компанії. Сама форма на сторінці складається з наступних компонентів:

1. **TextInput** – компонент для введення простого тексту. На формі присутні два компоненти: для вводу ім’я та прізвища:

1) **Select** – компонент для обирання серед наведених опцій. На формі реєстрації опціями виступають доступні для обору категорії змагання.

2) **PhoneInput** – компонент для вводу номеру телефону з випадаючим списком, що містить в собі всі наявні в світі телефонні коди країн.

3) **CheckBox** – компонент для обору однієї з наведених опцій, в формі надає користувачу можливість обрати свою стать.

4) **Button** – компонент, що виконує певну функцію при натисканні, на формі надає користувачу можливість завершити реєстрацію в змаганні та відправити свої данні на сервер, при умові, що всі введенні данні пройшли валідацію.

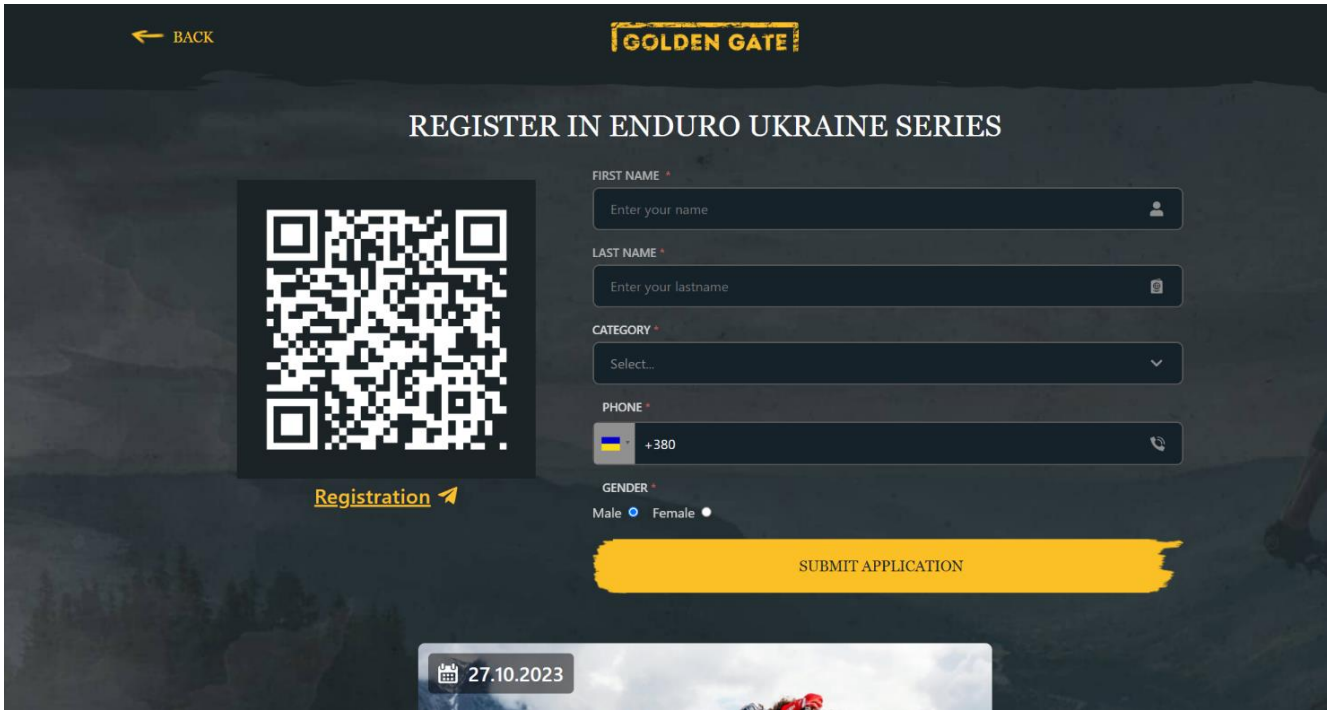

Рис 10 – приклад публічної сторінки реєстрації на змаганні.

На адмін панелі всі сторінки виконані в звичайному-бізнес стилі та своєю більшістю являють собою сторінки з таблицями. На сторінці менеджменту користувачів, адмін або власник компанії можуть переглянути список доступних

їм користувачів, задіяти фільтри, аби отримати певні данні, редагувати існуючих або додати нових користувачів (див Рис. 11). На цій сторінці існує можливість фільтрувати за ім’ям або поштою, шляхом введення тексту в відповідне поле.

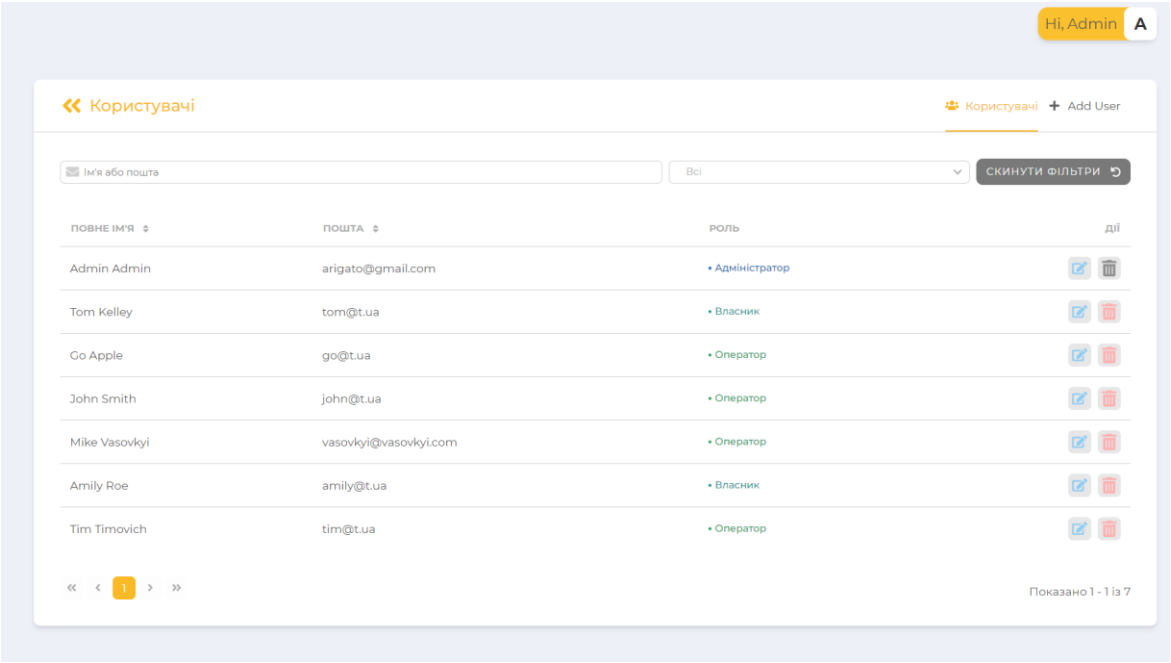

Рис. 11 – приклад сторінки менеджменту користувачів з адмінської сторони

Вид адмін панелі може доповнюватись новими функціями, якщо адмін використає бокове меню, щоб обрати компанію та змагання. Тоді в нього з’явиться функція швидкого доступу до сторінки компанії або змагання, що були обрані (див Рис 12)

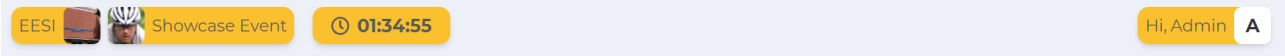

Рис. 12 – приклад вигляду заголовку сайту, при обраних компанії та події

При обраній події, на сторінці атлетів адмін зможе побачити вже існуючих спортсменів в рамках поточної події та редагувати їх, або зареєструвати

нових атлетів на події. На сторінці існують фільтри за ім'ям та поштовою скринькою, також можливість сортування за певними даними (дата реєстрації, номерок, місце у події). Також адмін зможе переглянути результати спортсменів натиснувши на будь-якого зі спортсменів (див Рис. 13).

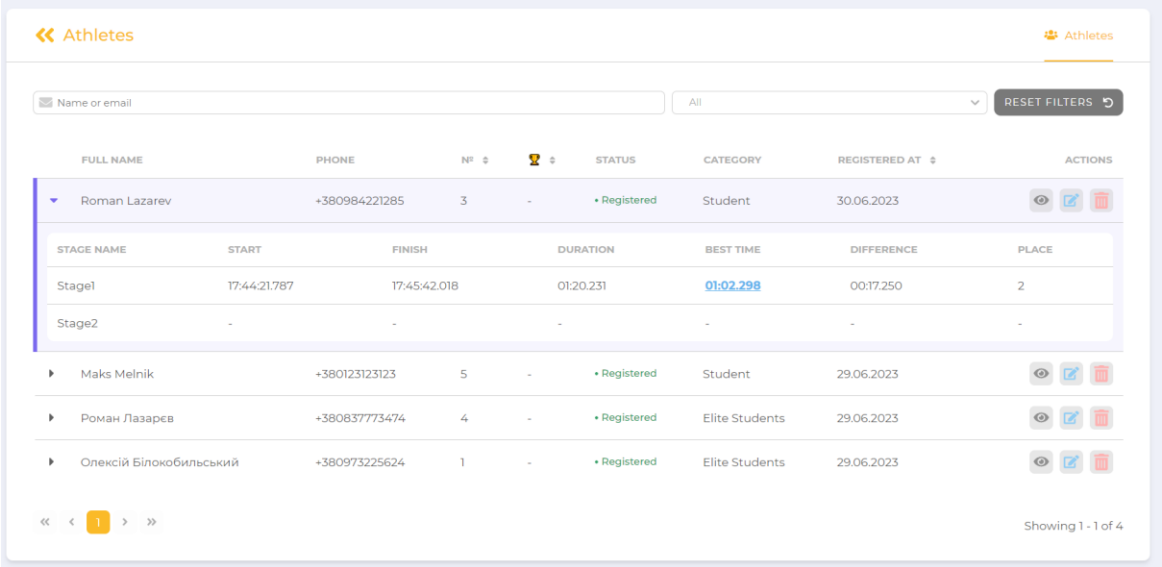

Рис. 13 – приклад вигляду сторінки спортсменів

На сайті є головне меню. З його допомогою можна переходити на потрібні сторінки. В компактному виді воно виглядає як набір іконок, тому не заважає користувачу і не займає місця на лендінгу. При навединні меню повністю випливає, показуючи список всіх сторінок на які можливо перейти (див. Рис. 14).

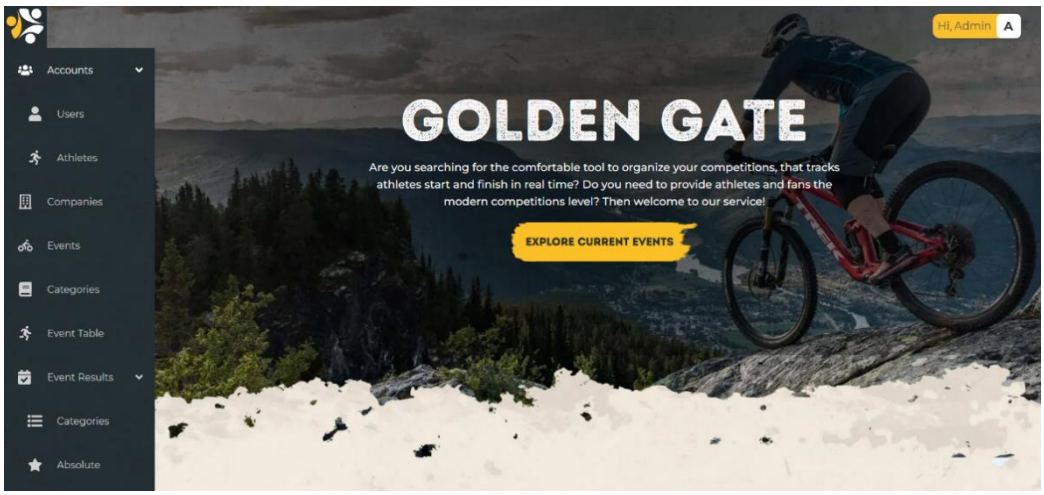

Рис. 14 – приклад вигляду меню сайту

В список входять такі сторінки:

1. **Accounts**:
   - **Users** - сторінка зі списком усіх зареєстрованих користувачів застосунку.
   - **Athletes** - сторінка зі списком усіх зареєстрованих спортсменів та деякі їй данні.
2. **Companies** - сторінка зі списком усіх зареєстрованих компаній.
3. **Events** - сторінка зі списком наявних чи майбутніх подій.
4. **Categories** - сторінка зі списком можливих варіацій змагань.
5. **Event Table** - сторінка для показу даних за поточним змаганням.
6. **Event Results**:
   - **Categories** - сторінка результатів події за категоріями.
   - **Absolute** - сторінка результатів всіх можливих подій.

На сторінці **Categories** можна побачити списки категорій подій (див. Рис. 15). В цьому розділі є можливість додавати та налаштовувати категорії для подій. Налаштувати можливо назву, опис категорії та колір, за яким вона буде відображатись на таблиці. Для редагування назви та опису потрібно натиснути на ввідний ряд **Title** чи **Description** та вести потрібний текст.

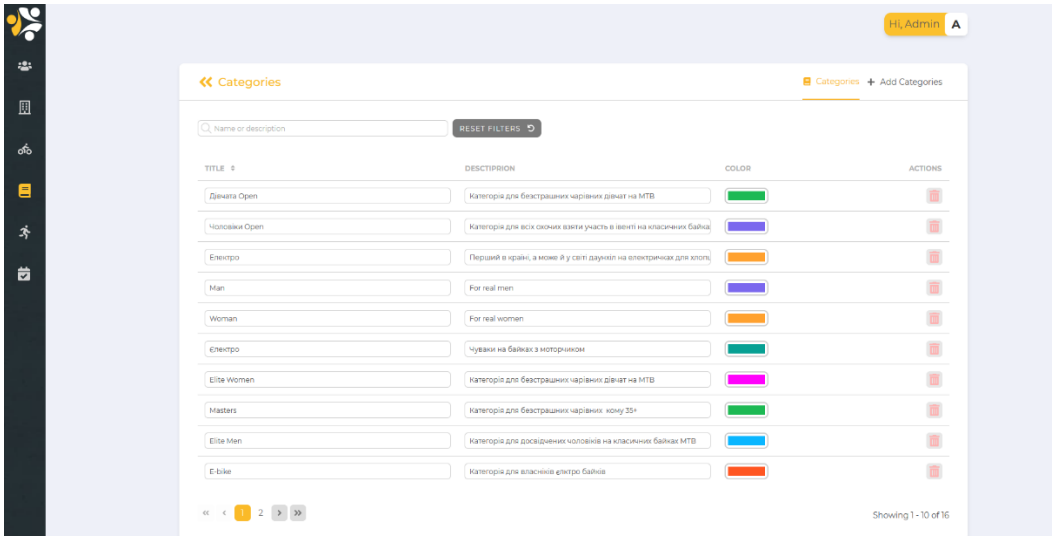

Рис. 15 – приклад вигляду сторінки **Categories**

Для того, щоб налаштувати колір потрібно натиснути на кнопку **Color**, після чого з'явиться модальне вікно в якому можна буде вибрати потрібний колір зі списку та підтвердити вибір натиснувши кнопку **Save Color** (див. Рис. 16).

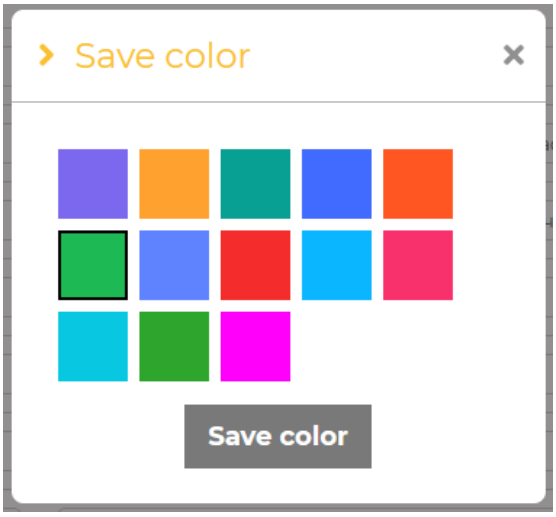

Рис. 16 – приклад вигляду модального вікна **Save Color**

Є можливість різного відображення списку даних категорій, пошуку потрібних та їх видалення. Для сортування списку потрібно натиснути стрілочні кнопки біля **Title**. Щоб знайти категорію, потрібно написати у пошукову строку назву чи опис. Поруч з пошуком є кнопка **Reset Filters**, яка очищує рядок пошуку.

Щоб видалити категорію, потрібно натиснути на кнопку видалення напроти обраного об'єкту та після цього з'явиться модальне вікно **Delete Category**, в якому можна підтвердити видалення кнопкою **Remove** чи відмінити його кнопкою **Cancel** (див. Рис. 17).

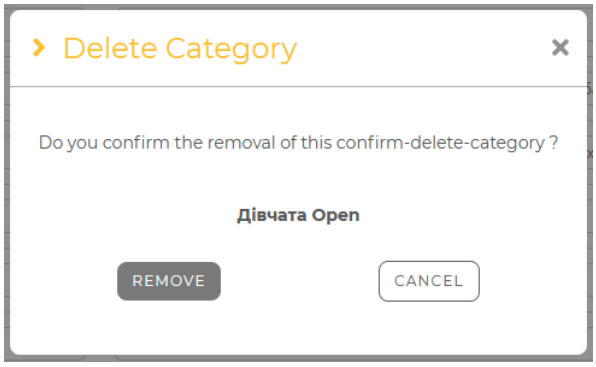

Рис. 17 – приклад вигляду модального вікна **Delete Category**

На сторінці **Companies** можливо подивитись список компаній, в якому є дані: назва компанії, ім'я власника, контактний номер та дата, коли були додані дані до списку (див. Рис. 18).

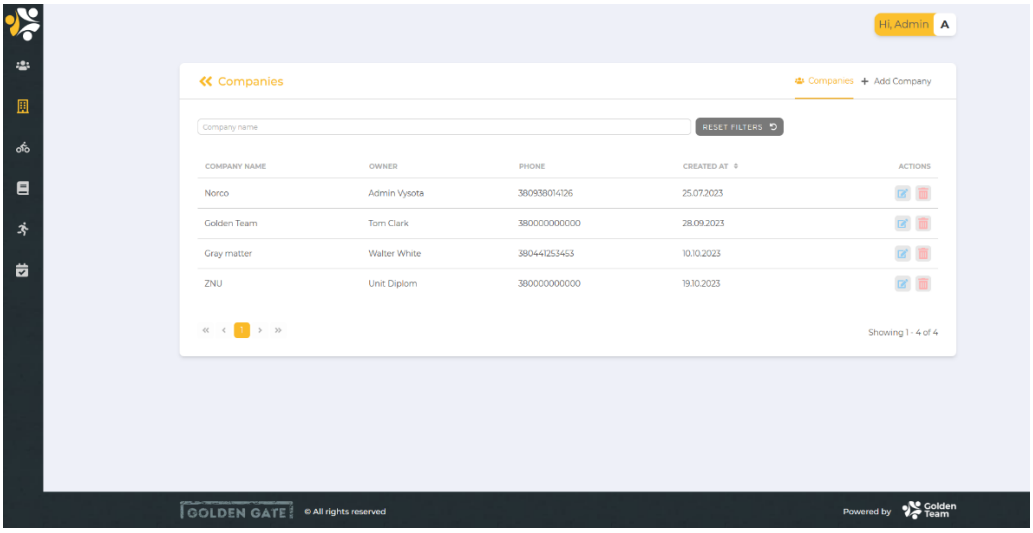

Рис. 18 – приклад вигляду сторінки **Companies**

Присутня можливість додати компанію. Для цього потрібно натиснути кнопку **Add Companies**, яка переведе нас до форми заповнення даних компанії. На сторінці додавання компанії потрібно заповнити: назву, власника, електрону пошту, посилання на сайт компанії, контактний телефон, сторінку Facebook та Instagram. Після заповнення потрібно натиснути кнопку **Add** для того щоб додати компанію (див. Рис. 19).

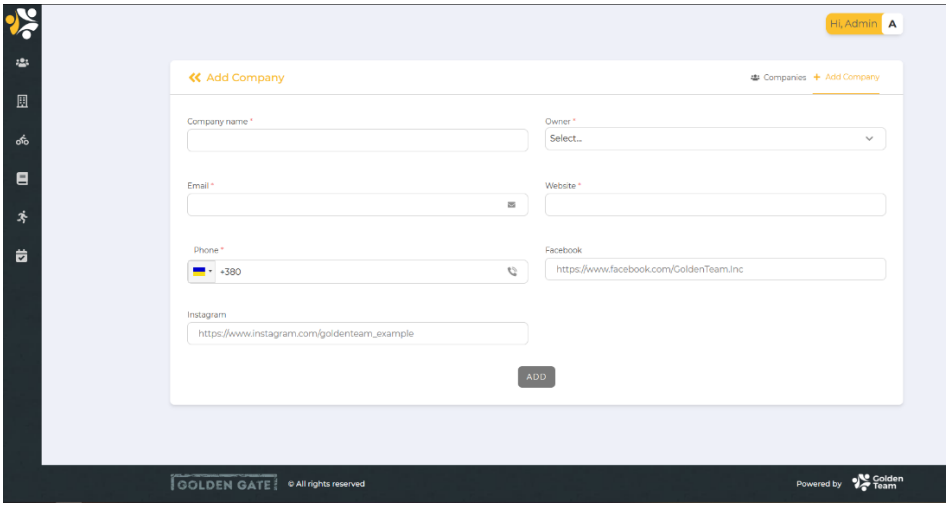

Рис. 19 – приклад вигляду сторінки **Add Companies**

Для пошуку потрібної компанії потрібно ввести назву компанії у строку пошуку. Очистити строку пошуку можна за допомогою кнопки **Reset Filters** (див. Рис. 18).

Для того щоб редагувати дані компаній, потрібно натиснути на кнопку напроти вибраної компанії, після цього відбудеться перехід на сторінку **Edit Company**. В першій вкладці сторінки, керівник компанії або адмін можуть побачити поля з заповненою інформацією компанії. Кожне текстове поле має іконку “Копіювати”, при натисканні на яку, відповідну інформацію буде скопійовано до буферу обміну користувача. Також на цій сторонці можна змінити логотип за допомогою кнопки **Change Logo** та почати редагувати дані при натисканні кнопки **Edit Info**. Після цього з'явиться кнопка **Save** для того, щоб зберегти введені дані (див. Рис. 20).

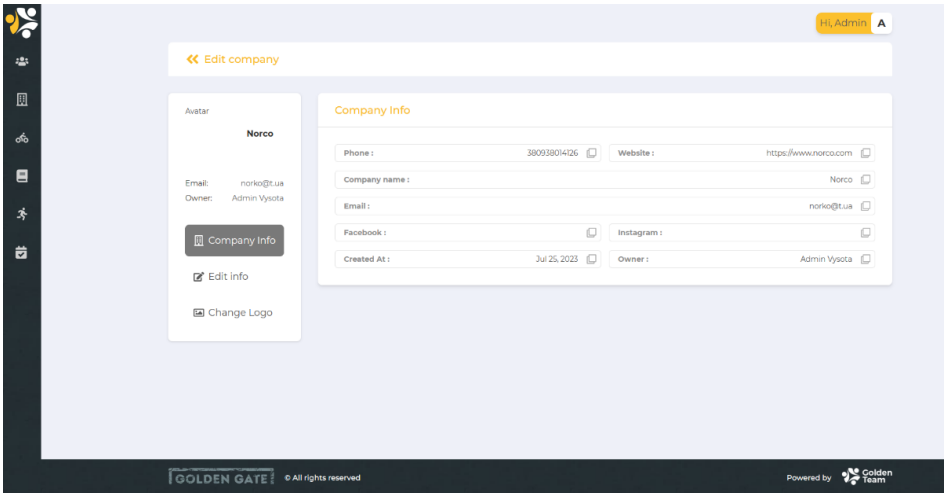

Рис. 20 – приклад вигляду сторінки **Edit Company**

Також, на сторінці **Companies** є кнопка видалення компанії, яка знаходиться напроти кожної компанії. При натисканні з'явиться модальне вікно, в якому можна підтвердити видалення кнопкою **Remove** чи відмінити видалення кнопкою **Cancel** (див. Рис. 21).

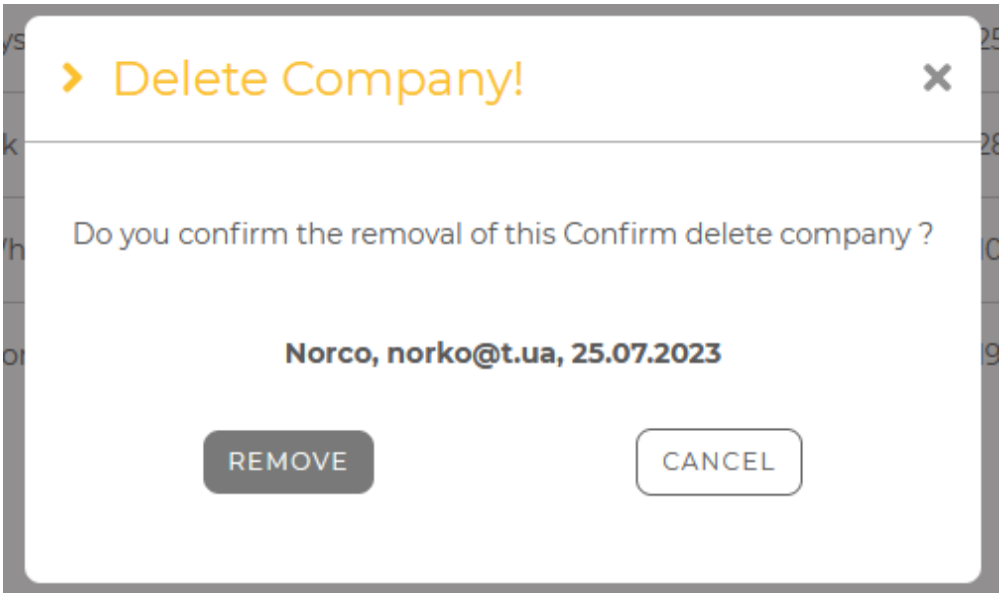

Рис. 21 – приклад вигляду модального вікна **Delete Company**

На сторінці **Events** можливо подивитись список подій, в якому є дані: назва події, дата та час початка події, статус та кількість етапів (див. Рис. 22).

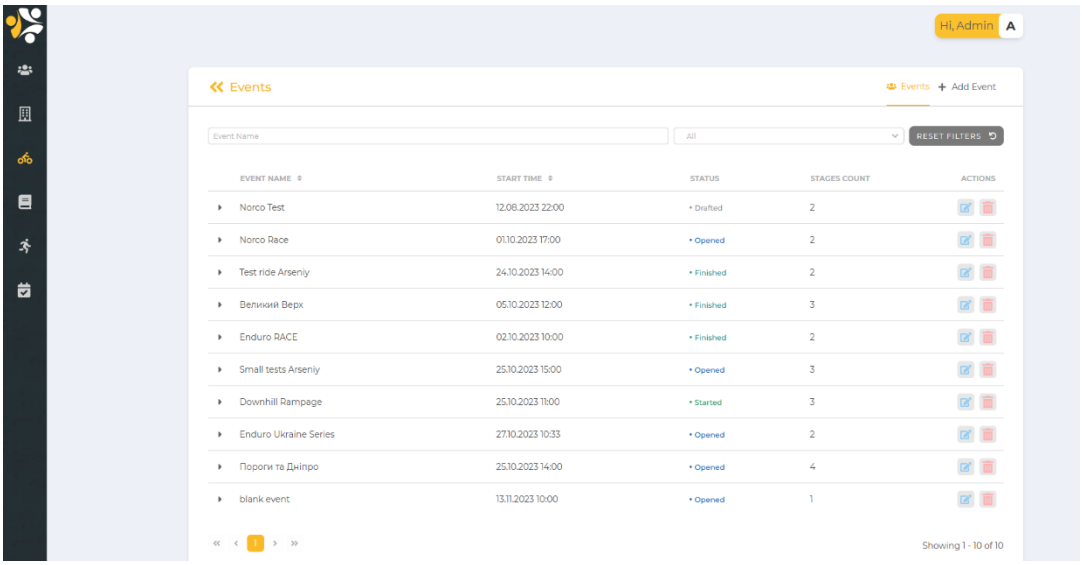

Рис. 22 – приклад вигляду сторінки **Events**

Для пошуку потрібної компанії потрібно ввести назву компанії у строку пошуку та можливо обрати статус події. Очистити строку пошуку та скинути статус можна за допомогою кнопки **Reset Filters**.

Також, на сторінці **Events** є кнопка видалення компанії, яка знаходиться напроти кожної компанії. При натисканні з'явиться модальне вікно, в якому можна підтвердити видалення кнопкою **Remove** чи відмінити видалення кнопкою **Cancel** (див. Рис. 23).

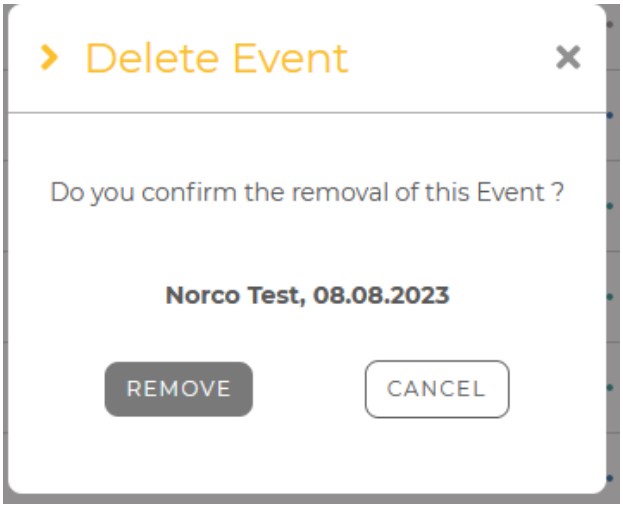

Рис. 23 – приклад вигляду модального вікна **Delete Company**

Для того щоб редагувати дані компаній, потрібно натиснути на кнопку напроти вибраної компанії, після цього відбудеться перехід на сторінку **Edit Event**. На цій сторонці є меню з різними модальними вікнами(див. Рис. 24).

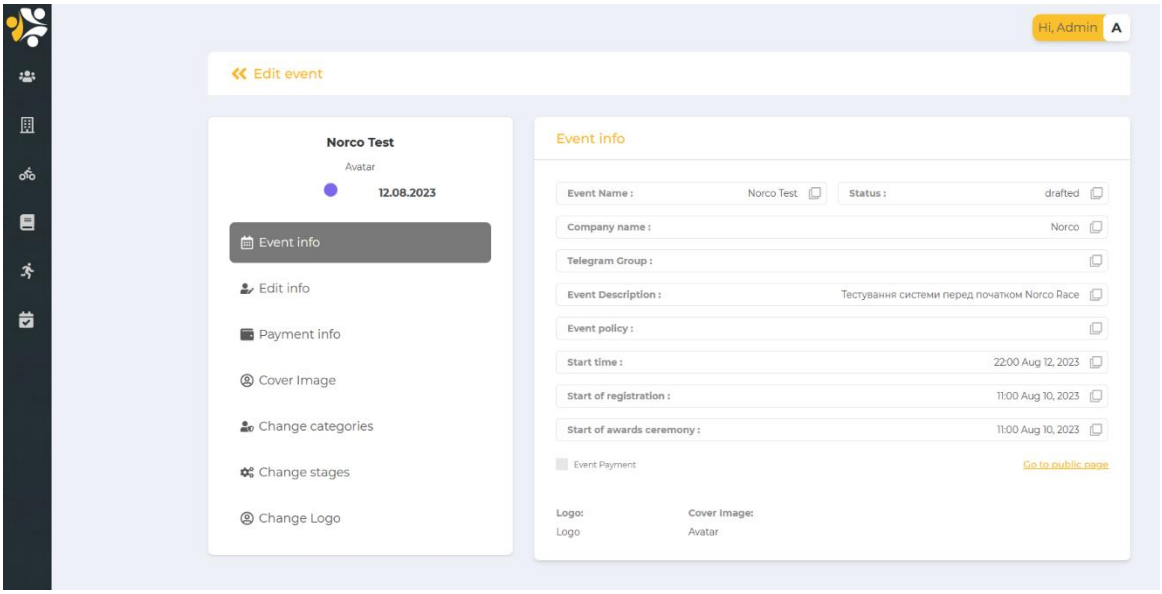

Рис. 24 – приклад вигляду сторінки **Edit Event**

Event Info дозволяє подивитися інформацію відповідно вибраній події. **Edit info** дозволяє відкривати редагування інформації події та змінити потрібні поля. Після чого натиснути на кнопку **Save** для зберігання змін (див. Рис. 25). **Payment info** дозволяє заповнити платіжну інформацію.

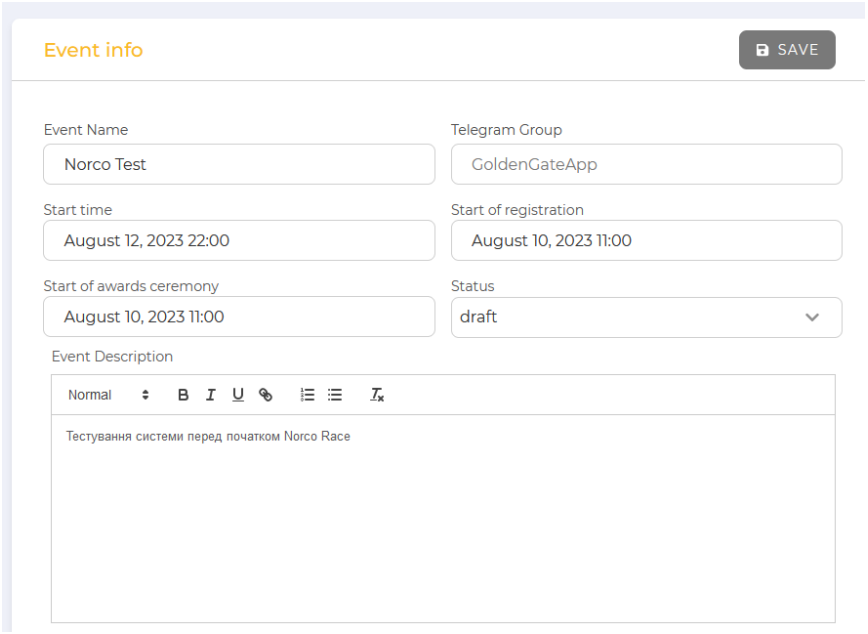

Рис. 25 – приклад вигляду сторінки **Edit info**

Cover image та **Change logo** дозволяє змінити дозволяє змінити банер події та логотип події відповідно (див. Рис. 26).

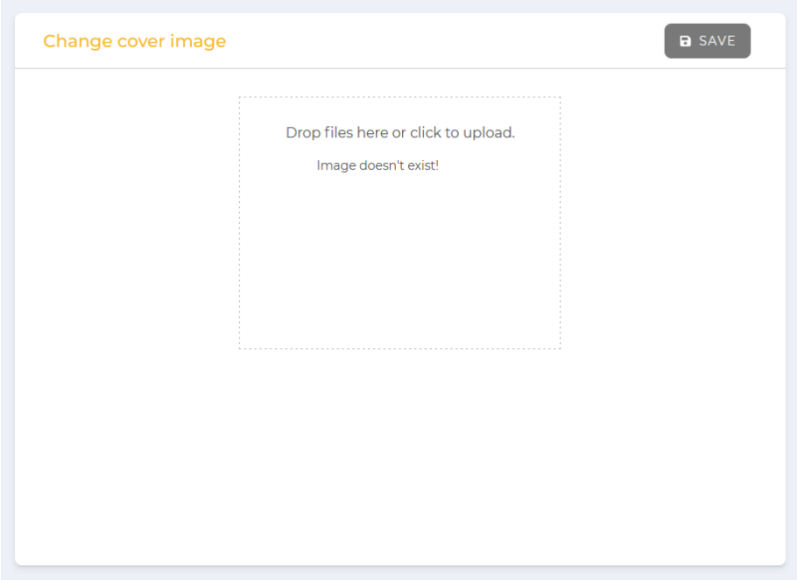

Рис. 26 – приклад вигляду сторінки **Cover image** та **Change logo**

Change categories дозволяє додати категорію за допомогою кнопки **Add** до обраної події (див. Рис. 27).

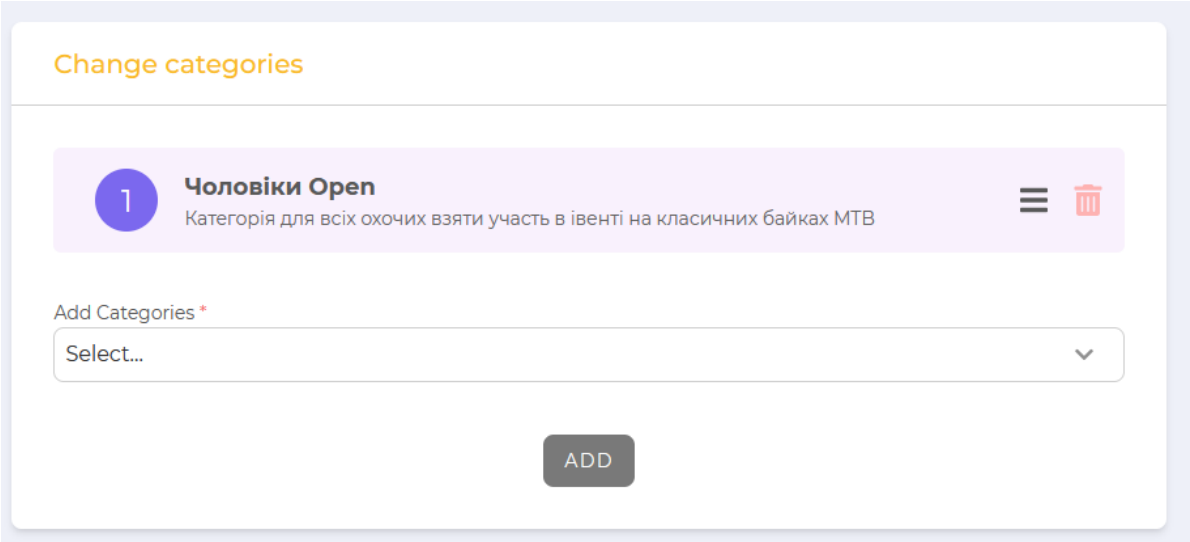

Рис. 27 – приклад вигляду сторінки **Change categories**

Change stages (див. Рис. 28) дозволяє додати стадію, або редагувати та видалити вже існуючі стадії в події, а також переглянути наявну заповнену інформацію про існуючі вже створенні. Також господар компанії або адмін можуть переглядати фотографію закріплену за стадією, при натисканні кнопки з іконкою ока. Індикатор іконки буде зеленим, якщо фотографія наявна, та сірим якщо відсутня.

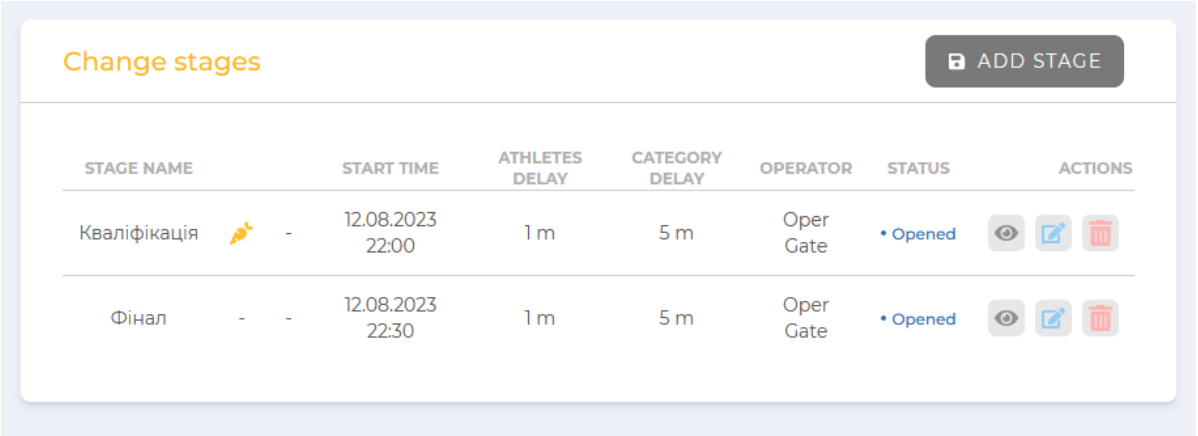

Рис. 28 – приклад вигляду сторінки **Change stages**

За допомогою кнопки **Add Stage** відкривається модальне вікно для заповнення даних стадії. На модальному вікні існує поле для завантаження фотографії, що буде слугувати обкладинкою стадії події. Також існують наступні важливі поля: Stage Name – поле для вводу назви етапу, Stage Description – поле для вводу опису етапу, Operator – поле з вибором користувача ролі «Оператор», який буде відповідальним за даний етап, де всі опції вибору є статичними і репрезентують собою зареєстрованих в компанії користувачів з роллю «Оператор». Також існує поле вибору дати “Start Time”, що дозволяє задати дату та час початку етапу. Після введення всіх даних у модальному вікні зміни можливо зберегти за допомогою кнопки **Save** (див. Рис. 29).

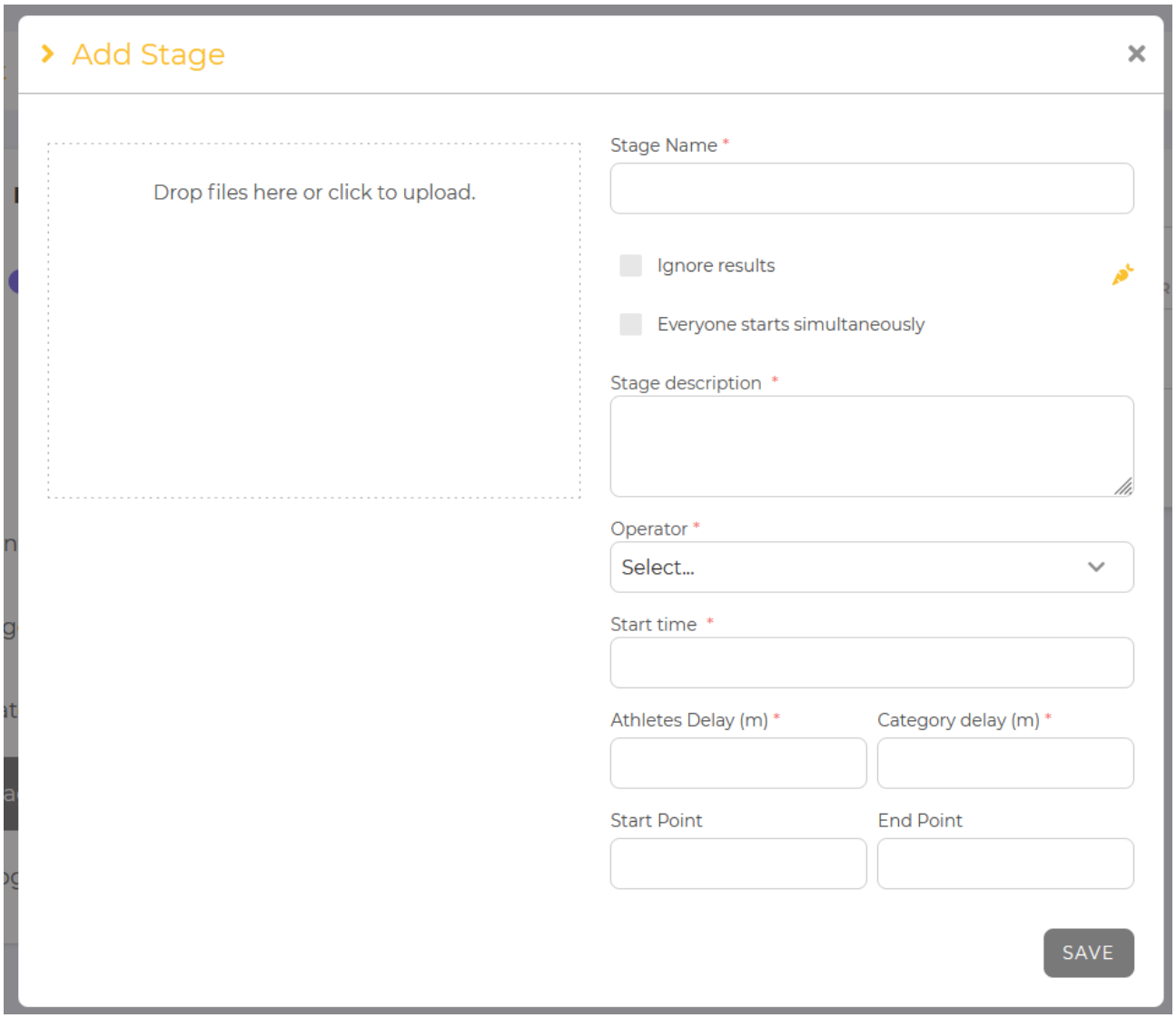

Рис. 29 – приклад вигляду модального вікна **Add Stage**

4 ПРОВЕДЕННЯ ТА МЕНЕДЖЕМНТУ СПОРТИВНИМИ ЗМАГАННЯМИ

4.1 Єдина архітектура як міцний скелет проекту

В результаті створення інформаційної системи, весь архітектурний вигляд системи як на серверній так і на клієнтський частині були досягнуті великі переваги в області обслуговування коду, зрозумілості коду, а також можливості додавання нових сутностей і масштабування системи. Система має уніфікований код, який не є прив’язаним до назв сутностей, а є спільним. Такого результату вдалось досягти завдяки використовуванню методології програмування Dependency Injection [22]. Така методологія дозволяє об’єднати код зробивши його доступним в будь-яких класах, що наслідують BaseContext, що значно полегшує та пришвидшує роботу системи.

Переваги використання Dependency Injection на сервері:

Зменшення залежностей: Використання DI дозволяє знизити залежності між компонентами програми, що робить систему більш гнучкою та легше піддається змінам.

Спрощення тестування: Завдяки DI та використанню інверсії керування (Inversion of Control, IoC), тестування окремих компонентів стає простіше через можливість підставляти моки або замінювати залежності для тестів.

Покращення читабельності та обслуговування: DI дозволяє покращити читабельність коду та його обслуговування, оскільки зменшується кількість «жорстко» прописаних залежностей в коді.

Нормалізація даних на клієнтській стороні та їх підведення під однакові схеми:

Забезпечення консистентності даних: Нормалізація даних та підведення під однакові схеми дозволяють зберігати інформацію консистентною та однорідною, спрощуючи її обробку та аналіз.

Уникнення дублювання інформації: Спільне використання однакових схем на клієнтській стороні дозволяє уникнути дублювання та зберігання однакових даних у різних форматах, що сприяє економії місця та підвищенню продуктивності.

Покращення ефективності використання даних: Однакові схеми дозволяють швидше та легше обробляти дані на клієнтській стороні, що пришвидшує роботу вьюшек та забезпечує більш ефективну роботу із схожими типами даних.

4.2 Важливість правильної побудови архітектури бази даних

При першому аналізі ТЗ, та первинній побудові архітектури бази даних, було використано підхід побудови реляційної схеми бази даних, з подальшим спрощенням відносин «багато до багатьох» шляхом вміщення дочірніх сутностей в середині батьківських.

Але на більш пізньому етапі розробки застосунку, через обмеженість аналізу з точки зору обміну даними в реальному часу через сокети, постала проблема обміну даними, при якій, після зміни або додаванні сутності EventResult (результат атлета в події на певному етапі, або глобально), через зберігання результатів в середині події, через сокети доводилось відправляти всю сутність події, а також вносити зміни до бази всю сутність, замість маленьких частин. В свою чергу це призвело до зменшення швидкості виконання обміну даними та неефективному використанню запитів до БД. Після ретельного вивчення проблеми та аналізу помилок, було зроблено рефакторинг коду відповідно до змін в базі даних, та прийнято рішення винести сутності Атлетів та Результатів Атлетів з сутності Події до окремих таблиць. Через цю помилку, було переписано близько 30% вихідного коду проекту, що зайняло дуже багато часу, та ще раз підтверджує важливість і необхідність правильної постанови ТЗ, та ретельного аналізу при побудові архітектури БД.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було поставлено та виконано наступні задачі:

1. Визначена предметна область та проаналізована існуюча проблематика. В ході спілкування з атлетами та керівниками підприємств, що займаються велоспортом, було з’ясовано, що наявність систем автоматизації змагань дуже обмеженна і існує здебільшого за кордоном, де коштує багато грошей або не задовільняє всім вимогам. Таким чином, питання створення системи автоматизації та проведення змагань для України є дуже поширеним і конкурентно здатним.

2. Були розглянуті різні платформи та технології для розробки інформаційно системи. При вивченні переваг та недоліків перевага була віддана мові JavaScript, як на серверній, так і на клієнтській частині, для стандартизації коду та в цілях використання єдиної мови для всіх підсистем загальної системи. Оптимальними засобами реалізації програмного застосунку були обрані: JavaScript (TypeScript), Visual Studio Code, MongoDB, Redis, ReactJS, Redux, Typegoose.

3. Було розроблено бізнес-обгрунтування та вимоги до програмного продукту, після чого спроектовані та розроблені інформаційна система керування та серверна база даних до неї. Також описано її інтерфейс та найважливіші модулі.

4. В результаті проведеного дослідження було визначено, що систем створення, проведення та менеджменту спортивних змагань дуже мало, а ті, що існують, мають або дуже велику ціну за оренду обладнання та за використання іх системи як платформи для проведення, або не мають повного функціоналу, щоб ними могли користуватись їх основна аудиторія — спортсмени.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. About Product. raceresult.com : веб-сайт. URL: https://www.raceresult.com/en-us/contact/_index.php (дата звернення: 10.05.2023).
2. About Challonge. challonge.com: веб-сайт. URL: https://challonge.com/about (дата звернення: 07.04.2023).
3. About IT’s Your Race: веб-сайт. URL: (дата звернення: 13.02.2023)
4. Getting Started with Redux redux.js.org веб-сайт. URL: https://redux.js.org/introduction/getting-started (дата звернення: 19.02.2023).
5. What Socket.io is. socket.io веб-сайт. URL: https://socket.io/docs/v4/ (дата звернення: 05.03.2023).
6. SockJS.ably.com веб-сайт. URL: https://ably.com/periodic-table-of-realtime/sockjs (дата звернення: 05.03.2023).
7. A single platform to Develop and deliver real-time interactivity. pubnub.com. веб-сайт. URL: https://www.pubnub.com/products/pubnub-platform/#overview (дата звернення: 06.03.2023).
8. MongoDB documentation веб-сайт. URL: https://www.mongodb.com/docs/atlas (дата звернення: 21.01.2023)
9. Getting started with ReactJS веб-сайт. URL: https://ru.reactjs.org/docs/getting-started.html (дата звернення: 07.12.2022)
10. Getting started NextJS веб-сайт. https://nextjs.org/docs/getting-startedURL: (дата звернення: 07.12.2022)
11. Microsoft Clusters Database documentation веб-сайт. https://www.msclusters.com/docs/sqlserver/ URL: (дата звернення: 21.01.2023)
12. MySQL documentation веб-сайт. URL: https://dev.mysql.com/doc/ (дата звернення: 07.12.2022)
13. Getting started with VueJS веб-сайт. URL: https://vuejs.org/guide/introduction.html (дата звернення: 21.01.2023)

14. Getting started with JavaScript веб-сайт. URL: https://learn.javascript.ru/ (дата звернення: 07.12.2022).
15. About SPA applications веб-сайт. URL: https://wezom.com.ua/blog/chto-takoe-spa-prilozheniya (дата звернення: 07.12.2022).
16. About NodeJS веб-сайт. URL: https://nodejs.org (дата звернення: 02.09.2023).
17. Carlson J. Redis in action. Simon and Schuster, 2013. Книга.
18. Macedo T., Oliveira F. Redis cookbook: Practical techniques for fast data manipulation. " O'Reilly Media, Inc.", 2011. Книга. (дата звернення: 05.09.2023).
19. Banker K. et al. MongoDB in action: covers MongoDB version 3.0. Simon and Schuster, 2016. Книга. (дата звернення: 10.03.2023).
20. Padolsey J. Clean Code in JavaScript: Develop reliable, maintainable, and robust JavaScript. Packt Publishing Ltd, 2020. Книга. (дата звернення: 12.03.2023).
21. Prasanna D. Dependency injection: design patterns using spring and guice. – Simon and Schuster, 2009. Книга. (дата звернення: 05.03.2023).
22. Edward S. G., Sabharwal N. Practical MongoDB: Architecting, Developing, and Administering MongoDB. Apress, 2015. Книга. (дата звернення: 09.02.2023).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Білокобильський Олексій Андрійович, студент 2 курсу, денної форми навчання,

Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz118bd-02@stu.zsea.edu.ua,

- підтверджую, що написана мною кваліфікаційна робота на тему: «**Особливості побудови серверної частини автоматизованої системи для організації та проведення спортивних змагань**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст. 42 Закону України «Про освіту», зі змістом яких ознайомлений;
- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;
- згоден на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою Інтернет-системи, в також архівування моєї роботи у базі даних цієї системи.

Дата _____ Підпис _____Білокобильський Олексій Андрійович
(студент)

Дата _____ Підпис _____Скрипник Ірина Анатоліївна
(науковий керівник)