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 десктоп-додатку для організації 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)
М.О. Мельник
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н. І.А. Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензен директор ТОВ Дискус
П.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

Мельнику Максиму Олександровичу
(прізвище, ім'я, по батькові)

1.Тема роботи Особливості побудови десктоп-додатку для організації 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)

11 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	02.09-10.09.22	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.22	виконано
3	Аналіз існуючих методів рішення	13.09-14.09.22	виконано
4	Дослідження RFID технології	15.09-20.09.22	виконано
5	Дослідження Alien рідера	21.09-26.09.22	виконано
6	Узгодження подальших дій з науковим керівником	27.09-28.09.22	виконано
7	З’єднання з Alien рідером через HTTP	29.09-13.10.22	виконано
8	З’єднання з сервером	14.10-16.10.22	виконано
9	Представлення отриманих результатів науковому керівнику та узгодження плану подальшого дослідження	17.10-19.10.22	виконано
10	Реалізація функціоналу для черги атлетів	20.10-09.11.22	виконано
11	Порівняння результату з аналогами	10.11-17.11.22	виконано
12	Реалізація користувацького інтерфейсу для комп’ютерної системи	18.11-22.11.22	виконано
13	Оформлення звіту	23.11-27.11.22	виконано

Студент _____ М.О. Мельник
(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: 100

Рисунків: 37

Таблиць: 0

Джерел: 16

Мельник М. О. Особливості побудови десктоп-додатку для організації спортивних змагань: кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І. А. Скрипник. Запоріжжя : ЗНУ, 2020. 100 с.

Мета і завдання дослідження полягають у створенні десктопного застосунку для організація спортивного змагання.

Спортивні змагання – були, є і будуть дуже популярними серед усіх людей. Але щоб їх організувати потрібно знати і враховувати багато нюансів. Наприклад: тип змагання, кількість учасників, погодні умови і т. д.

Тепер сучасні технології дуже полегшують даний процес. Не потрібно більше записувати результати на аркуші чи стояти з секундомером на фініші.

Але існує проблема - усі ці високо-технологічні рішення майже не використовуються організаторами турнірів середнього рівня. На покупку, налаштування і підтримання таких систем буде витратчатися забагато коштів.

Тому основна мета даного проекта була у створення повнофункціональної системи, яка не буде коштувати занадто дорого.

Ключові слова: десктопний застосунок, атлет, фініш, Electron, React, RFID, ALIEN.

SUMMARY

Pages: 100

Figures: 37

Tables: 0

Sources: 16

Melnik M.O. Features of Building a Desktop Application for Organizing Sports Competitions: Master's Qualification Work in the Specialty 121 "Software Engineering" / scientific supervisor I.A. Skrypnyk. Zaporizhzhia : ZNU, 2020. 100 p.

The aim and objectives of the research involve creating a desktop application for organizing sports competitions.

Sports competitions have been, are, and will remain very popular among people. However, to organize them, it is necessary to know and take into account many nuances, such as the type of competition, the number of participants, weather conditions, etc.

Nowadays, modern technologies greatly facilitate this process. There is no need to record results on paper or stand with a stopwatch at the finish line anymore.

But there is a problem - all these high-tech solutions are almost not used by organizers of medium-level tournaments. Spending too much money on the purchase, setup, and maintenance of such systems is a significant barrier.

Therefore, the main goal of this project was to create a fully functional system that would not be too expensive.

Keywords: desktop application, athlete, finish, Electron, React, RFID, ALIEN.

ЗМІСТ

ВСТУП

Актуальність теми

Актуальність теми моєї дипломної роботи не може бути переоцінена в контексті сучасного спортивного середовища. Змагання, будь то масові заходи чи високопрофесійні турніри, завжди привертають увагу та інтерес широкого загалу. Однак, несмотря на велику популярність, організація спортивних заходів залишається складним завданням, яке потребує не лише професійного підходу, але й використання сучасних технологій.

В контексті організації спортивних змагань, моє дослідження націлене на розробку десктопного застосунку, спрямованого на спрощення та автоматизацію процесів. Сучасні технології, такі як Electron та React, відкривають нові можливості для зручного створення програмних продуктів, спрямованих на полегшення організаційних аспектів спортивних заходів.

Однією з ключових проблем, яку вирішує моя дипломна робота, є забезпечення доступності високотехнологічних рішень для організаторів турнірів середнього рівня. Зазвичай, такі інноваційні системи вимагають значних витрат на їхню покупку, налаштування та обслуговування. Враховуючи це, створення ефективної та економічно доступної системи, яка допомагатиме в організації спортивних подій, стає важливою задачею для підтримки та розвитку спортивного середовища в Україні.

У цьому контексті, моя робота має стратегічне значення, оскільки вона спрямована на подолання перешкод у використанні сучасних технологій для всебічного поліпшення організації спортивних змагань, зробивши їх більш доступними та ефективними для широкого кола організаторів та учасників.

Додатковою вагомою причиною актуальності обраної теми є необхідність вдосконалення систем управління та моніторингу спортивних заходів у зручний та ефективний спосіб. В умовах сучасного світу, де ритм життя надто швидкий, а вимоги до точності та оперативності дедалі зростають,

наявність десктопного застосунку для автоматизації організаційних аспектів спортивних змагань стає запорукою успішності та привабливості таких подій.

До того ж, враховуючи сучасні тенденції в розвитку спортивного галузі, де важливу роль відіграють взаємодія та інтеграція різних технологій, моя робота також спрямована на створення системи, яка може впроваджувати та синхронізувати різні технічні рішення, такі як RFID та ALIEN, для забезпечення найвищого рівня точності та надійності в проведенні спортивних заходів.

Отже, не лише простота та доступність, але й відповідність сучасним вимогам до технологічних інновацій робить дану тему невід'ємною складовою удосконалення спортивної індустрії. Моя дипломна робота ставить за мету не лише розв'язання конкретних завдань організації спортивних заходів, а й вносить свій вклад у загальний розвиток інформаційних технологій у сфері спорту.

Мета і завдання дослідження

Метою даного дослідження є розробка та реалізація десктопного застосунку для ефективної організації та моніторингу спортивних змагань. Завдання включають в себе вивчення особливостей спортивних заходів, розробку програмного забезпечення з використанням сучасних технологій, таких як Electron та React, а також забезпечення доступності та високої ефективності системи для широкого кола користувачів.

Об’єкт дослідження

Об’єктом дослідження є RFID та Electron технології.

Предмет дослідження

Предметом дослідження є створення десктопного застосунку, який спростить та автоматизує процеси організації спортивних змагань,

використовуючи сучасні технології та забезпечуючи ефективність та доступність для різних організаторів та учасників.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз існуючих десктопних застосунків для організації спортивних змагань та визначення їхніх переваг і недоліків.
2. Вивчення моделей Electron та React для побудови користувацького інтерфейсу додатку з метою оптимізації взаємодії з організаторами та учасниками спортивних подій.
3. Аналіз можливостей використання технологій RFID та ALIEN для покращення системи ідентифікації та реєстрації учасників на спортивних заходах.
4. Аналіз існуючих датасетів із зображеннями спортсменів з метою використання їх для тренування та валідації системи.
5. Синтез та узагальнення отриманих даних та знань для розробки повнофункціональної системи для організації та ведення спортивних змагань.
6. Експериментальне застосування розробленого десктопного застосунку для проведення спортивних подій та оцінка його ефективності на реальних заходах.
7. Проведення експериментів зі зміною параметрів системи та їх впливу на продуктивність та зручність використання для користувачів.

Наукова новизна одержаних результатів

Наукова новизна отриманих результатів полягає в розробці та впровадженні десктопного застосунку для організації спортивних заходів, що сприяє значному поліпшенню ефективності та доступності організаційного процесу. Розроблений застосунок використовує оптимальні алгоритми для

розпізнавання та визначення положення учасників у режимі реального часу, забезпечуючи точність та оперативність відслідковування подій під час спортивних змагань. Цей підхід відкриває нові можливості для ефективного використання технологій у сфері спорту та сприяє оптимізації організаційних аспектів, зробивши їх більш доступними та привабливими для організаторів та учасників.

Практичне значення одержаних результатів

Практичне значення отриманих результатів дослідження у моєму випадку полягає в розробці та впровадженні десктопного застосунку для організації та моніторингу спортивних змагань. Розроблений застосунок в реальному часі надає можливість ефективно визначати різноманітні аспекти спортивних подій, включаючи реєстрацію учасників, визначення переможців, та обробку результатів.

Практична важливість цього дослідження проявляється у створенні доступної та ефективної системи для організаторів та учасників спортивних подій. Розроблений застосунок дозволяє оптимізувати процеси планування та ведення змагань, використовуючи сучасні технології, такі як Electron та React. Це робить організацію спортивних подій більш доступною та ефективною для широкого кола користувачів, необтяжуючи їх значними витратами часу та ресурсів.

Апробація одержаних результатів

Результати дослідження були представлені на XIII науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2023», а також на XXV науково-технічній конференції студентів, аспірантів, магістрантів і викладачів Інженерного навчально-наукового інституту Запорізького національного університету.

Глосарій

RFID (Radio-Frequency Identification) — технологія, що використовує радіочастотні хвилі для безконтактного зчитування та запису інформації на спеціальних мітках або картках, забезпечуючи швидку та ефективну ідентифікацію об'єктів.

Electron — це фреймворк для створення кросплатформених десктопних застосунків з використанням веб-технологій, таких як HTML, CSS та JavaScript, що дозволяє розробникам створювати програми для різних операційних систем з єдиною кодовою базою.

React — це JavaScript-бібліотека для розробки інтерфейсу користувача, створена компанією Facebook. Вона використовується для розробки веб-додатків з високою динамічністю та швидкодіючим інтерфейсом, дозволяючи розбивати користувацький інтерфейс на компоненти для зручності управління та обслуговування коду.

ALIEN — це бренд, пов'язаний з розробкою RFID-технологій, включаючи читачі, мітки та інші рішення, що забезпечують високий рівень точності та швидкості в ідентифікації об'єктів за допомогою радіочастотного взаємодії.

Decktop-застосунок — програмне забезпечення, розроблене для використання на персональних комп'ютерах та ноутбуках, яке дозволяє організаторам спортивних подій ефективно керувати та відстежувати результати змагань. Може використовувати технології, такі як RFID, для автоматизації процесів ідентифікації учасників.

Кросплатформений — Властивість програмного забезпечення або фреймворку, яка дозволяє йому працювати на різних операційних системах без необхідності відокремленого коду для кожної платформи. Для desktop-застосунків використовується фреймворк Electron, що забезпечує кросплатформеність застосунків.

Радіочастотна взаємодія — спосіб передачі даних безпосередньо через радіочастотні хвилі, що використовується в технології RFID для зчитування та запису інформації на мітках або картках без необхідності фізичного контакту.

HTML, CSS та JavaScript — ключові веб-технології, використовувані в десктоп-застосунку. HTML відповідає за структуру сторінки, CSS - за її стиль та оформлення, а JavaScript - за динамічні елементи та взаємодію з користувачем.

Redux — контейнер стану для JavaScript-додатків, що використовується для управління станом додатку та його зручного оновлення. У десктоп-застосунках дозволяє зберігати та відновлювати стан системи, навіть при втраті інтернет-з'єднання, що є важливим аспектом у реальних умовах проведення спортивних змагань.

Tailwind CSS — CSS-фреймворк, що дозволяє швидко та зручно оформлювати веб-елементи за допомогою готових класів, що прискорює процес розробки і забезпечує єдність дизайну усіх компонентів у десктоп-застосунку.

1 ДОСЛІДЖЕННЯ ОСОБЛИВОСТЕЙ ПОБУДОВИ ДЕСКТОП-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ СПОРТИВНИХ ЗМАГАНЬ

1.1 Огляд проблеми організації спортивних змагань

Організація спортивних змагань у сучасному світі є складним завданням, яке включає в себе широкий спектр викликів та проблем. По-перше, планування та координація таких подій вимагає великих зусиль і ресурсів. Забезпечення правильного хронометражу, реєстрація учасників, визначення переможців та оповіщення про результати - усе це стає завданням, яке вимагає високої організаційної ефективності.

По-друге, обробка та аналіз великого обсягу даних, пов'язаних із спортивними змаганнями, може бути складною задачею без використання сучасних технологій. Велика кількість учасників, різноманітні види спорту та різні режими проведення подій роблять необхідність точності та оперативності в обробці даних проблемою.

Відсутність доступних та інноваційних засобів для ведення та моніторингу спортивних заходів стає третьою важливою проблемою. Організаторам турнірів середнього рівня може бути важко знайти доступні та ефективні рішення, що не вимагають великих витрат на придбання та обслуговування.

Таким чином, огляд проблеми організації спортивних заходів підкреслює важливість розробки нових інструментів та технологій для полегшення цього процесу, з метою забезпечення його ефективності та доступності.

Організатори спортивних змагань стикаються з наступними проблемами:

Складність планування та координації спортивних заходів.

Необхідність точного визначення результатів та обробки великого обсягу даних.

Відсутність доступних та інноваційних засобів для ефективного ведення та моніторингу спортивних подій.

1.2 Технології для організації спортивних змагань

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

Race Result – інформаційна-апаратна система проведення великого спектру спортивних змагань, 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 можливостей системи можливо зазначити:

• можливість провести змагання на велосипеді, марафон та, навіть, тріатлон;

• має детальну аналітику змагань, яку можна подивитися на веб-сайті (див. Рисунок 1);

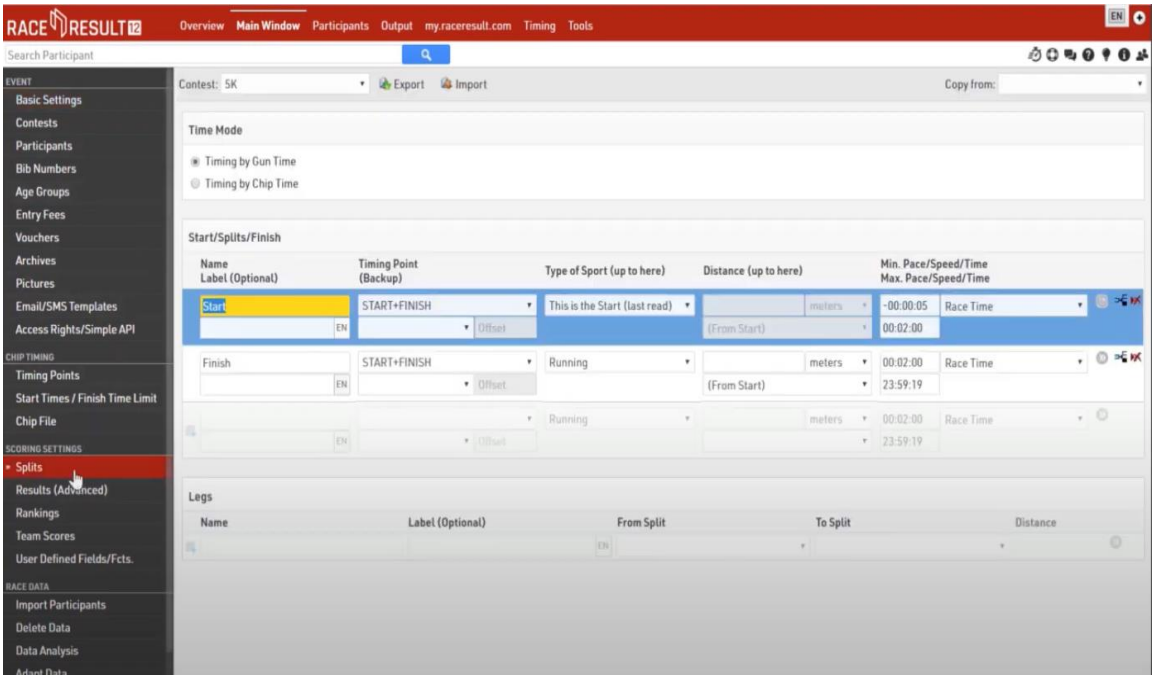

Рисунок 1 — Інтерфейс дешборду веб-застосунку Race Result

Апаратна частина цієї системи може коштувати до декількох тисяч долларів. Також потрібно доплачувати окремо за кожного учасника у середньому пів євро.

Challonge — це безкоштовний онлайн-сервіс для організації турнірів та змагань.

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.;

У системи дуже простий інтерфейс (див. Рисунок 2), що, авжеж, добре. Але простота інтерфейсу, в першу чергу, обумовлена малої кількістю функціоналу. Найбільша проблема, що система не вирішує головну проблему

організації турнірів, а саме зчитування часу старту і фінішу учасника. Все що ж є це турнірна сітка та учасники змагання.

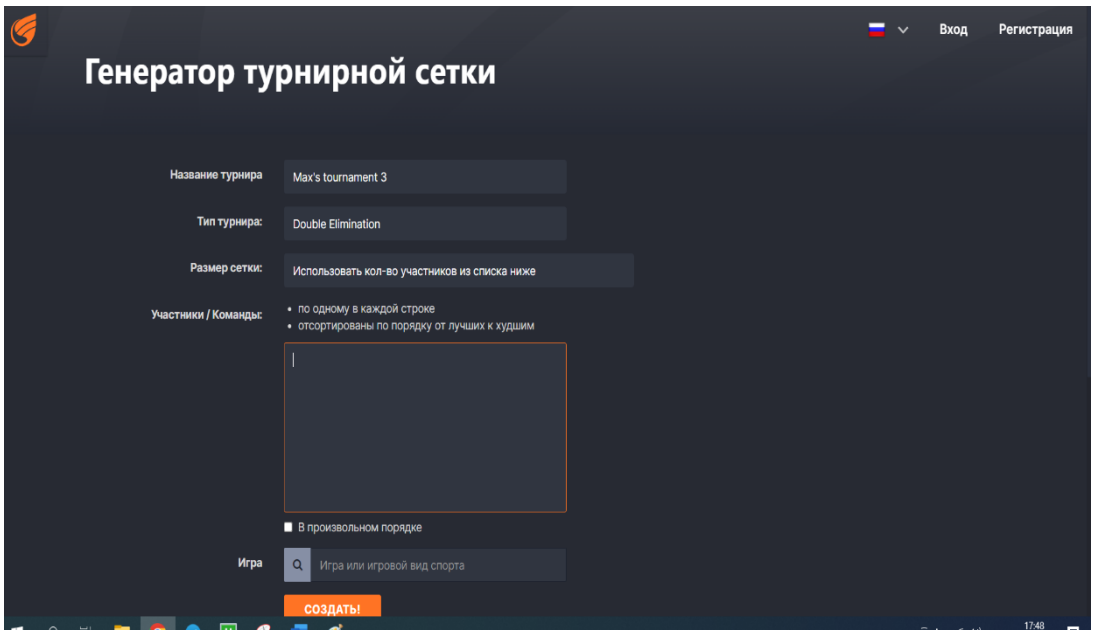

Рисунок 2 — Інтерфейс сторінки створення змагання системи Challonge

Race Roster — це також платформа для організації змагання. Сервіс керує додаванням та оновленням списків атлетів. Сайт має мінімалістичний інтерфейс (див. Рисунок 3), що дозволить навить зробити змагання або зареєструватися у ньому. Знову проблема з тим, що система не пропонує апаратного рішення. Але, з іншого боку, є можливість інтеграції з іншими системами зчитування часу.

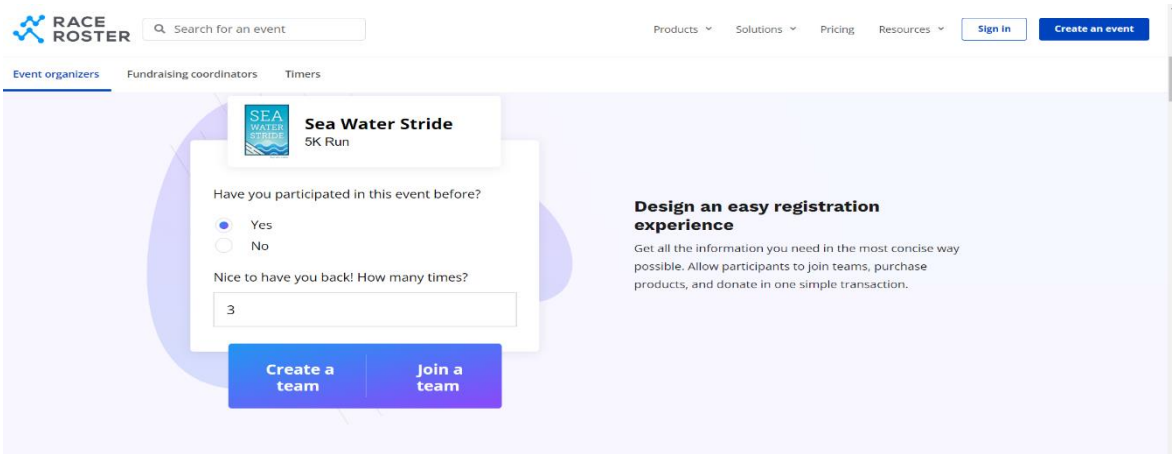

Рисунок 3 — Сторінка дешборду сайту Race Roster

Bikereg — це система, яка дозволяє організовувати та просувати велосипедні події, включаючи гонки, марафони та інші змагання. Сайт надає інструменти для створення та налаштування заходів, обробки реєстрацій учасників та управління списками стартових листів. Серед переваг можна виділити мінімалістичний та інтуітивний інтерфейс сайту (див. Рисунок 4), просту систему створення змагання та дуже подрібну і глибоку аналітику змагань. Усі результати змагань не розраховуються автоматично, тому потрібно усі результати на сайті вбивати вручну.

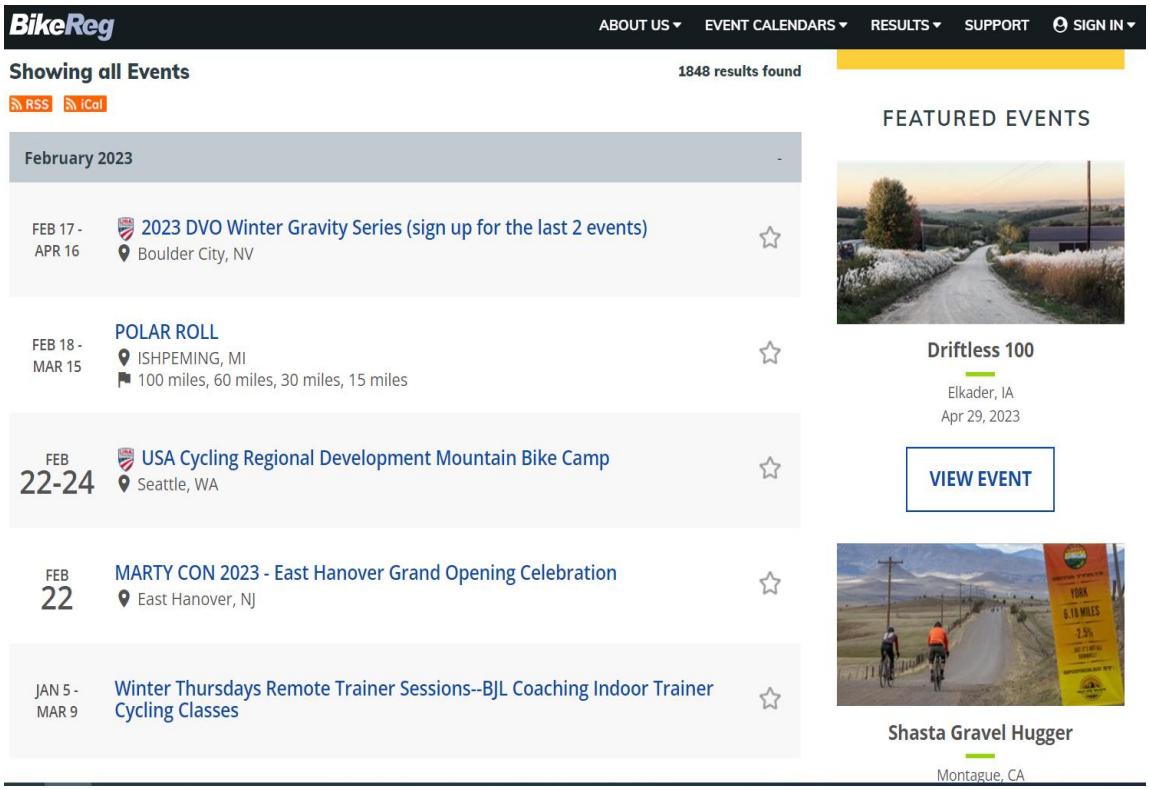

Рисунок 4 — Приклад інтерфейсу сайту Bikereg

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. 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. Рисунок 5) не відповідає вимогам “User Friendly” , хоча система передбачає користування звичайними користувачами;
- відсутність особистого кабінету та перегляду історії змагань;

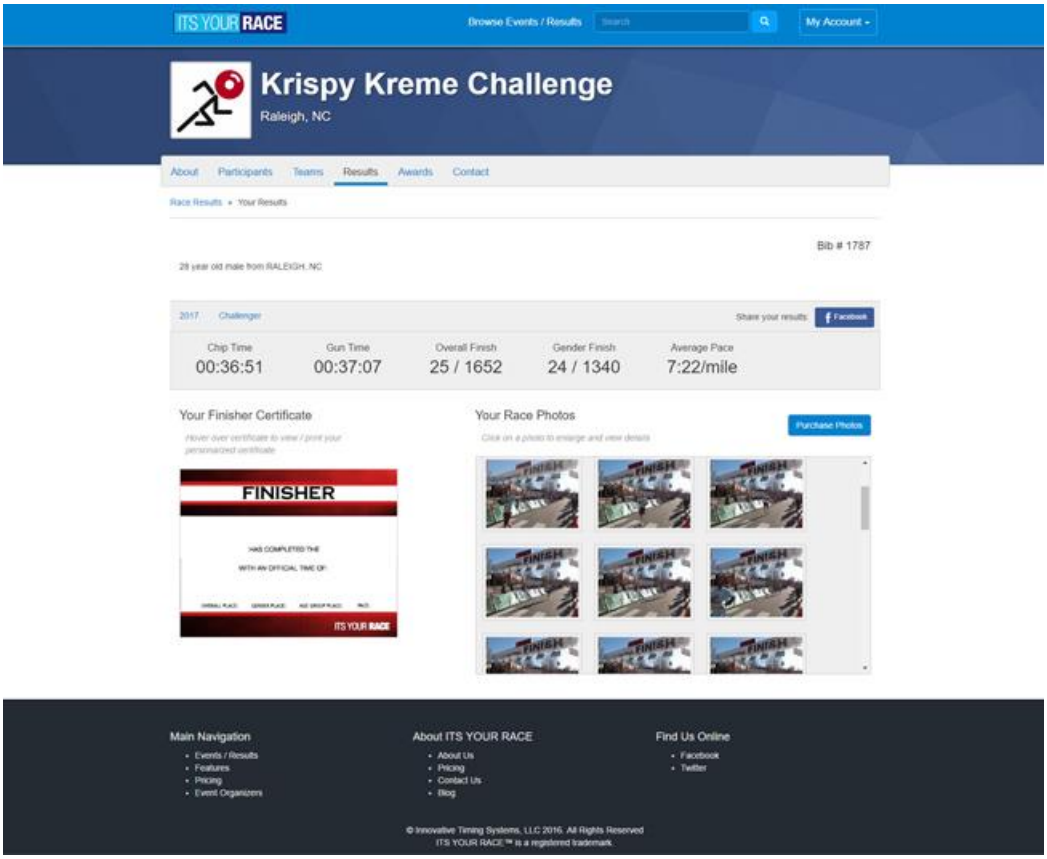

Рисунок 5 — Дизайн IT’s Your Race

Якщо робити висновок аналогічних систем, то вони або коштують занадто багато, або не мають критично важливого функціонала. Тому рішення зробити свою систему націлену на українскький ринок є актуальним.

1.3 Сфери застосування RFID технологій

RFID (Radio-Frequency Identification) технології знайшли широке застосування в різних сферах діяльності, сприяючи автоматизації та

підвищенню ефективності багатьох процесів. Нижче розглянемо деякі з сфер застосування RFID технологій:

1. Логістика та Спряжені ланцюги постачання:

RFID технології використовуються для відстеження руху товарів та управління запасами в логістичних та постачальних ланцюгах. Вони дозволяють точно визначити місцезнаходження товарів, спрощуючи інвентаризаційні та доставкові процеси.

2. Транспорт і автомобільна промисловість:

У транспорті RFID використовується для автоматичної оплати дорожнього квитка, відстеження транспортних засобів та управління парковкою.

3. Медична галузь:

В медицині RFID використовується для ідентифікації пацієнтів, ведення медичних карток та відстеження руху лікарських засобів та обладнання.

4. Роздрібна торгівля:

В роздрібній торгівлі RFID технології використовуються для відстеження товарів, забезпечення антикрадіжного захисту та поліпшення обслуговування клієнтів.

5. Виробництво:

У виробництві RFID може використовуватися для відстеження робочого процесу, управління інвентарем та покращення якості контролю.

6. Сільське господарство:

В сільському господарстві RFID може бути використано для відстеження вирощених культур, контролю за станом обладнання та оптимізації використання ресурсів.

7. Системи безпеки та доступу:

RFID часто використовується для створення систем контролю доступу та ідентифікації осіб, що забезпечує високий рівень безпеки в різноманітних областях, від корпоративних приміщень до громадських заходів.

8. Бібліотеки та освітні заклади:

В бібліотеках RFID використовується для автоматизації обліку та контролю за книгами, а також для ідентифікації відвідувачів.

RFID технології продовжують розвиватися, і їхнє застосування розширюється в різних галузях, сприяючи оптимізації та модернізації різноманітних аспектів побуту та промисловості.

1.4 Системи для організації спортивних змагань, які використовуються на території України

На території України системи для організації спортивних змагань стають все більш сучасними та технологічно вдосконаленими, відповідаючи вимогам сучасного спортивного середовища. Розглянемо деякі із застосовуваних систем, що сприяють ефективному проведенню та моніторингу спортивних подій.

1. Електронні системи хронометражу та реєстрації:

Електронні системи хронометражу та реєстрації широко використовуються на території України для точного визначення часу та

результатів учасників спортивних змагань. Ці системи забезпечують надійність та швидкість обробки даних.

2. RFID технології для ідентифікації учасників:

Використання RFID технологій для ідентифікації учасників дозволяє швидко та ефективно реєструвати спортсменів на заходах, а також ведення точного обліку їхнього часу та результатів.

3. Інтерактивні табло та екрани:

Сучасні системи використовують інтерактивні табло та екрани для відображення різноманітної інформації під час спортивних подій. Це може включати в себе результати, статистику, графіки та інші важливі дані для глядачів та учасників.

4. Онлайн-сервіси та мобільні додатки:

З впровадженням онлайн-сервісів та мобільних додатків організатори та учасники можуть легко отримувати доступ до розкладу, результатів, анонсів та іншої інформації про спортивні змагання в режимі реального часу.

5. Автоматизовані системи управління турніром:

Для організації турнірів та змагань широко використовуються автоматизовані системи управління, які допомагають в плануванні, реєстрації, розподілі груп, та веденні статистики.

6. Системи антидопінгового контролю:

Для забезпечення чесності та справедливості у спортивних змаганнях використовуються спеціалізовані системи антидопінгового контролю, що дозволяють ефективно виявляти порушення антидопінгових правил.

Загальна тенденція використання цих систем полягає в підвищенні рівня професіоналізму та прозорості у спортивних заходах. Впровадження сучасних технологій сприяє поліпшенню організаційних процесів, зростанню інтересу глядачів та забезпеченню високого стандарту проведення спортивних змагань в Україні.

На території України в сучасному спорті дедалі більше використовуються технології, що сприяють підвищенню якості та привабливості спортивних заходів. Електронні системи хронометражу та реєстрації дозволяють не лише точно визначити час учасників, але і створюють умови для швидкої та ефективної обробки результатів. Це сприяє не тільки об'єктивності визначення переможців, а й розкриттю нових можливостей для підвищення рівня конкуренції та спортивної динаміки.

Однак електронні системи - це лише частина розмаїття технологій, які знаходять застосування в організації спортивних заходів. Впровадження RFID технологій для ідентифікації учасників дозволяє зробити реєстрацію більш ефективною та унікальною для кожного учасника, а також забезпечує точність визначення їхнього місця та часу на трасі чи полі. Інтерактивні табло та екрани надають глядачам та учасникам можливість отримувати розширену інформацію, відображену в зручному форматі, що підвищує залученість та взаємодію всіх учасників в спортивному процесі.

Окрім цього, розвиток онлайн-сервісів та мобільних додатків стає невід'ємною частиною спортивного середовища, дозволяючи швидко та зручно отримувати доступ до інформації про заходи, а також сприяючи взаємодії глядачів та учасників. Автоматизовані системи управління турніром, в свою чергу, оптимізують процеси планування та розподілу, роблячи проведення спортивних змагань більш структурованим та ефективним. Загалом, впровадження сучасних технологій в організацію спортивних заходів на території України надає нові можливості для розвитку та популяризації спорту в усіх його аспектах.

1.5 Недоліки використання поточних систем

Незважаючи на те, що сучасні технології виявляють значний позитивний вплив на організацію спортивних заходів, вони також супроводжуються рядом недоліків та викликів, які потребують уважного вивчення та вирішення.

По-перше, одним із ключових недоліків є висока вартість впровадження та підтримки сучасних систем. Закупівля необхідного обладнання, програмного забезпечення та тренування персоналу може виявитися фінансово витратною операцією, особливо для менших спортивних заходів та організаторів з обмеженим бюджетом.

По-друге, проблемами може стати залежність від технологій та їхній можливий збій або відмова під час проведення заходів. Технічні проблеми, такі як відключення живлення, несправні датчики або програмні помилки, можуть призвести до перерв у роботі систем та порушити нормальний хід змагань, що має негативний вплив на враження від події.

По-третє, інтеграція різних технологічних рішень може виявитися складною задачею. Наприклад, використання різних систем для хронометражу, ідентифікації та відображення результатів може вимагати додаткового зусилля для синхронізації та взаємодії між ними, щоб уникнути конфліктів та невідповідностей у виведенні даних.

По-четверте, існує ризик приватності та безпеки даних, особливо при використанні технологій ідентифікації, таких як RFID. Збір та зберігання особистих даних може викликати суперечки із законами про конфіденційність, і в разі недостатнього захисту цієї інформації може виникнути ризик витоку даних або недобросовісного використання.

Загалом, необхідно уважно враховувати ці недоліки та розробляти стратегії для їхнього подолання при впровадженні технологічних рішень в організацію спортивних змагань. Эфективна інтеграція технологій повинна

бути супроводжена ретельним плануванням, навчанням персоналу та вдосконаленням систем безпеки та захисту даних.

2 ДОСЛІДЖЕННЯ ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Найдоцільніший формат інформаційної системи

Для застосунку фінішу можна відокремити два варіанти: WEB-сайт або десктоп-застосунок.

2.1.1 Переваги та недоліки розробки системи у вигляді десктоп застосунку

Десктоп-застосунок має ряд конкурентних переваг, а саме:

Висока продуктивність:

Десктоп-застосунки мають доступ до ресурсів локального комп'ютера, що дозволяє їм працювати швидше та ефективніше, особливо при обробці великої кількості даних або запуску ресурсоємних операцій.

Офлайн-режим:

Користувачі можуть використовувати десктоп-застосунок навіть без підключення до Інтернету. Це особливо важливо в ситуаціях, коли доступ до мережі обмежений або неможливий.

Більший доступ до функціоналу ОС:

Десктоп-застосунки можуть легко взаємодіяти з операційною системою, використовувати різноманітні API та функції, що може поліпшити їхні можливості та інтеграцію з іншими програмами.

Зручний доступ до ресурсів пристрою:

Десктоп-застосунки можуть легко взаємодіяти з різними пристроями, такими як принтери, сканери, камери, що може бути важливим для роботи з даними та обладнанням на місці.

Але є і декілька проблем:

Обмежена мобільність:

Десктоп-застосунки не є настільки мобільними, як веб-застосунки чи мобільні додатки. Вони призначені для роботи на конкретному пристрої, що може обмежити доступність для користувачів.

Складніше розгортання та оновлення:

Для користувачів важливо вчасно отримувати оновлення та нові функції. Десктоп-застосунки можуть вимагати складнішого процесу розгортання та оновлення порівняно з веб-застосунками.

Залежність від операційної системи:

Розробка десктоп-застосунку для конкретної операційної системи (наприклад, Windows, macOS або Linux) може вимагати додаткового часу та зусиль для підтримки різних платформ.

Вищі витрати на розробку та тестування:

Розробка десктоп-застосунку може бути більш витратною, оскільки потрібно враховувати різні операційні системи та забезпечити сумісність з різними версіями. Також, тестування може бути складнішим через різноманіття конфігурацій обладнання.

2.1.2 Переваги та недоліки розробки системи у вигляді веб застосунку

Розробка веб-застосункі має наступні переваги:

Універсальний доступ:

Веб-застосунки можна використовувати на різних пристроях та операційних системах, що робить їх більш універсальними та доступними для користувачів.

Легке оновлення та розгортання:

Оновлення веб-застосунків можна впроваджувати централізовано на сервері, що полегшує розгортання нових версій та забезпечує всім користувачам однаковий доступ до оновлень.

Мобільність:

Веб-застосунки можна використовувати на мобільних пристроях, що дозволяє користувачам отримувати доступ до необхідної інформації з будь-якого місця.

Зручність в розробці та тестуванні:

Веб-застосунки можуть бути розроблені та протестовані один раз для різних платформ, що полегшує роботу розробників та зменшує витрати.

Але також є і проблеми:

Залежність від Інтернету:

Веб-застосунки вимагають підключення до Інтернету, і в разі відсутності мережі користувачам буде важко отримати доступ до функцій застосунку.

Обмежені можливості доступу до ресурсів пристрою:

В порівнянні з десктоп-застосунками, веб-застосунки можуть мати обмежений доступ до ресурсів локального пристрою, таких як файлова система, принтери тощо.

Обмежені можливості офлайн-режиму:

Хоча багато сучасних веб-застосунків підтримують офлайн-режим, це не завжди є таким ефективним, як в десктоп-застосунках.

Браузерні обмеження:

Розробка для різних браузерів може створювати виклики через різні інтерпретації стандартів та багатство конфігурацій браузерів.

Було прийнято рішення використовувати саме десктоп-застосунок тому що змагання можуть проводитися у горах, де може пропадати інтернет. І тільки десктоп-застосунок надає можливість продовжувати роботу через офлайн режим.

2.2 Технології розробки настільних застосунків

Розробка десктоп-застосунків включає в себе різноманітні технології, кожна з яких має свої унікальні особливості. Ось декілька ключових технологій для розробки десктоп-застосунків:

Electron: кросплатформенний фреймворк для десктоп-застосунків, використовуючи веб-технології. Його перевагами є кросплатформенність та можливість використання веб-технологій, таких як HTML, CSS, та JavaScript.

Electron є високопопулярним кросплатформеним фреймворком для розробки десктоп-застосунків, який дозволяє використовувати веб-технології для створення програм. В основі його функціональності лежить використання Chromium та Node.js, що надає можливість створювати додатки, які працюють на різних операційних системах, включаючи Windows, macOS та Linux.

Однією з ключових переваг Electron є його кросплатформенність, що дозволяє розробникам створювати один застосунок, який працює на різних платформах, не втрачаючи при цьому функціональності та зовнішнього вигляду. Це значно спрощує розробку та підтримку десктоп-додатків, зменшуючи зусилля, які необхідно вкладати у платформозалежний код.

Ще однією важливою перевагою є можливість використовувати знайомі веб-технології, такі як HTML, CSS та JavaScript, що робить процес розробки більш доступним та прискорює вивчення фреймворку для новачків у галузі веб-розробки.

Таким чином, Electron став потужним інструментом для розробників, що відкриває нові можливості у створенні десктоп-застосунків, об'єднуючи зручність веб-технологій та універсальність кросплатформеності.

Windows Forms: використання C# та Windows Forms для розробки десктоп-застосунків, зорієнтованих на операційну систему Windows. Це забезпечує інтеграцію з екосистемою Microsoft та простоту використання.

JavaFX: використання Java та JavaFX для розробки кросплатформенних десктоп-застосунків. JavaFX надає кросплатформенність та масштабованість.

Qt (C++): використання C++ та фреймворку Qt для розробки кросплатформенних десктоп-застосунків. Qt відомий своєю високою продуктивністю та кросплатформенністю.

Gtk: використання Gtk для розробки десктоп-застосунків, де можна використовувати різні мови програмування. Gtk надає гнучкість та можливість вибору мови програмування.

WPF: використання C# та Windows Presentation Foundation (WPF) для розробки десктоп-застосунків для Windows. WPF визначається сучасним дизайном та інтеграцією зі сервісами Microsoft.

Враховуючи усі доступні варіанти біло прийнято рішення використовувати саме Electorn тому що він базується на JavaScript і таким чином ми можему писать усі частини системі на єдиній мові програмування.

2.3 Вибір архітектури програмного забезпечення

Після прийняття рішення стосовно мови програмування інформаційної системи, настає час для вибору архітектури програмного застосунку. Вибір мови в парі з архітектурою грає вирішальну роль у визначенні ефективності та розвитку десктоп-застосунків. Необхідно глибше вникнути в різні архітектурні концепції, розглядаючи їхні переваги та недоліки, щоб знайти оптимальне рішення для конкретного завдання.

Монолітна архітектура — це простий, але ефективний підхід, де весь код застосунку розташований в єдиному блоку. Це спрощує розробку та тестування, але може ускладнити масштабування та оновлення великих проектів. Віддавати свій вибір на користь монолітної архітектури варто у тому випадку, коли програмний застосунок досить невеликий та необхідна простота розробки. (див. Рисунок 6)

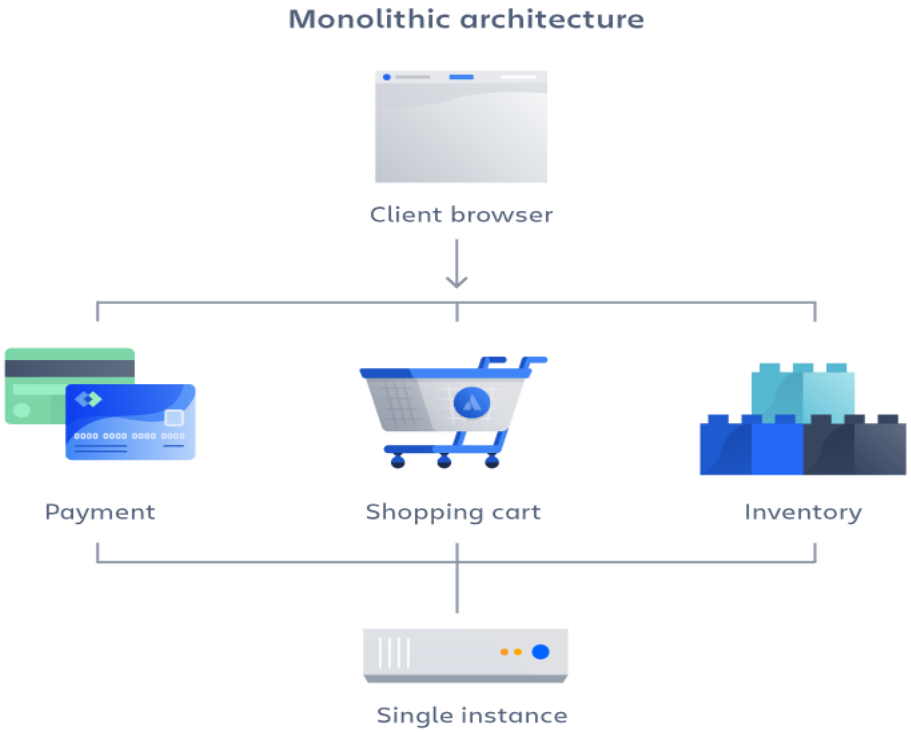

Рисунок 6 — Монолітна архітектура

Мікросервісна архітектура використовує набір невеликих, незалежних мікросервісів для реалізації функціоналу. Це забезпечує гнучкість та високу доступність, але може ускладнити управління та конфігурацію. Обирати мікросервіси необхідно у тому випадку, коли потрібна велика гнучкість та планується значний розвиток проекту. (див. Рисунок 7)

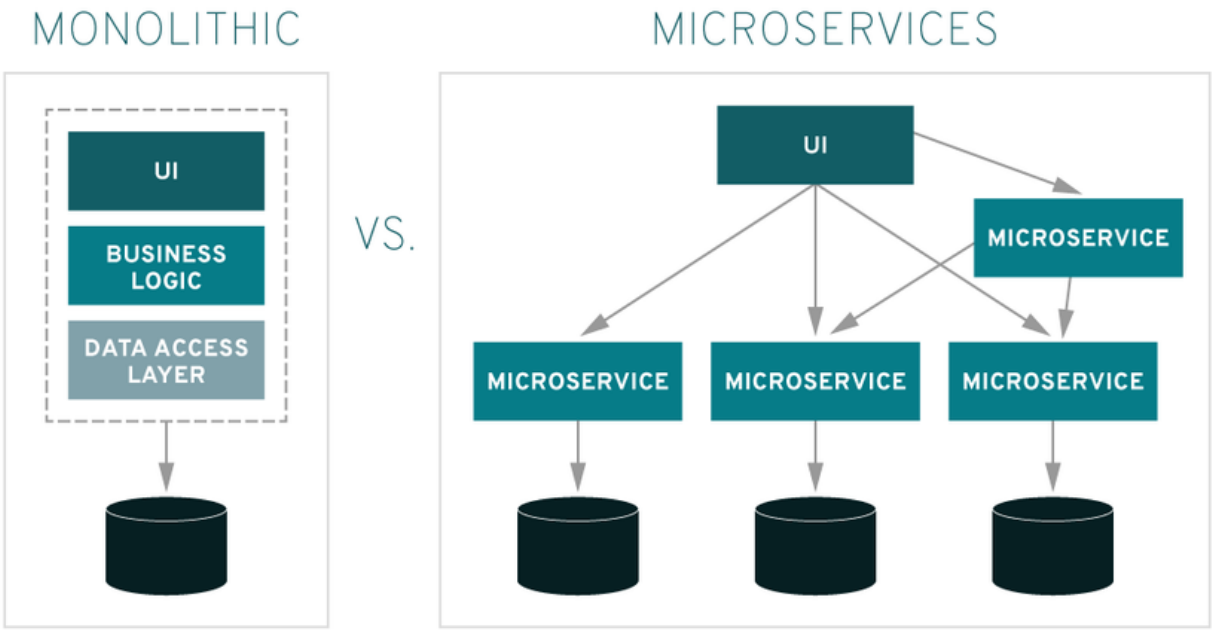

Рисунок 7 — Схема мікро-серверної архітектури

Клієнт-серверна архітектура встановлює чітке розділення між функціоналом на клієнтській та серверній сторонах. Це забезпечує легше масштабування, але може призвести до збільшеної складності в розробці. Необхідно обирати цей підхід, якщо потрібно розподілити обов'язки між клієнтом та сервером, а у майбутньому планується значне розширення інформаційної системи. (див. Рисунок 8)

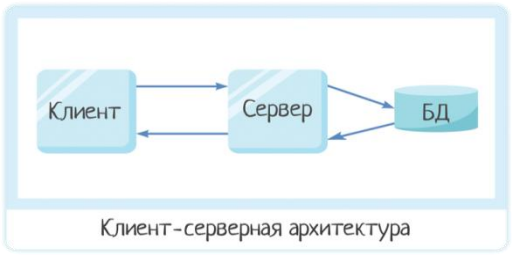

Рисунок 8 — Схема клієнт-серверної архітектури

Клієнт-серверна архітектура є однією з основних концепцій у розробці програмних систем. Вона передбачає використання двох основних компонентів: клієнта, який надсилає запити, та сервера, який відповідає на ці запити, обробляючи їх та передаючи результат клієнту.

Однією з ключових переваг цього підходу є чітке розділення обов'язків між клієнтською та серверною стороною. Це розподілення відповідальностей дозволяє зберігати бізнес-логіку та дані в централізованому серверному середовищі, що полегшує управління та забезпечує єдність даних.

Однак такий підхід може призвести до збільшеної складності у розробці, особливо при великому обсязі функціональності. Необхідно враховувати велику кількість взаємодій між клієнтом та сервером, а також вирішувати питання синхронізації та безпеки даних.

Вибір клієнт-серверної архітектури варто розглядати тоді, коли планується розподілити обов'язки між різними компонентами системи та важливо забезпечити легке масштабування. Це особливо актуально в тих випадках, коли передбачається значне розширення функціональності чи інформаційного обсягу системи у майбутньому.

2.4 Вибір технологій для розробки користувальницького інтерфейсу

Вибір технологій для розробки десктоп-застосунку — це стратегічне рішення, яке визначається конкретними вимогами проекту та власними уподобаннями розробників. Обгрунтуємо вибір React та Tailwind CSS для розробки десктоп-застосунку:

React:

1. Компонентна архітектура:

React пропонує компонентний підхід до розробки, що дозволяє розбити інтерфейс на невеликі, самодостатні компоненти. Це полегшує управління станом, підтримку коду та взаємодію між різними частинами застосунку.

2. Висока продуктивність:

React використовує віртуальний DOM та механізми оптимізації, що забезпечує високу продуктивність, швидкий рендерінг та ефективне використання ресурсів пристрою.

3. Широка спільнота та підтримка:

React користується великою та активною спільнотою розробників. Це забезпечує доступ до численних бібліотек, інструментів та ресурсів для вирішення різних завдань та проблем.

4. Кросплатформенність:

React може використовуватися для розробки не лише веб-застосунків, але й десктопних застосунків за допомогою фреймворку Electron. React працює о даній схемі (див. Рисунок 9):

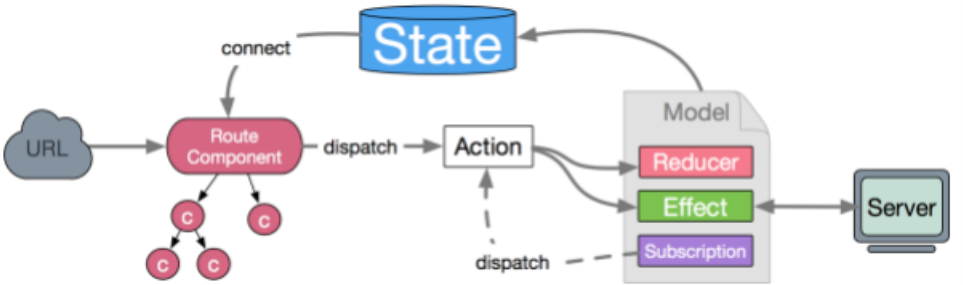

Рисунок 9 — Схема роботи React

Tailwind CSS має наступні переваги:

Рисунок 10 – Tailwind логотип

1. Спрощена робота зі стилями:

Tailwind CSS пропонує "utility-first" підхід, що дозволяє застосовувати стилі безпосередньо в HTML-коді. Це зроблює роботу зі стилями більш простою та швидкою, зменшуючи кількість CSS-коду.

2. Конфігураційна гнучкість:

Tailwind CSS надає можливість налаштовувати стилі відповідно до потреб проекту. Замість написання власного CSS, розробники можуть використовувати вже існуючі класи Tailwind або легко налаштовувати свої.

3. Масштабованість проектів:

Завдяки концепції "utility-first", Tailwind CSS підходить як для невеликих проектів, так і для великих, де простота використання та швидкість розробки є ключовими факторами.

4. Активна спільнота:

Tailwind CSS також має велику та активну спільноту розробників, що забезпечує підтримку, регулярні оновлення та надає доступ до різних розширень та інструментів.

Обрані технології роблять фокус на розробці зручної та швидкої інтерфейсної частини застосунку, забезпечуючи при цьому високий рівень продуктивності та гнучкість у налаштуванні.

Головний аспект при вибору технологій була можливість перевикористання компонентів у різних частинах системи.

2.5 Вибір фронтенд сховища данних

При виборі механізму для управління станом даних у десктоп-застосунку, було враховано кілька альтернативних підходів, таких як локальний стан React, використання Context API, MobX та Redux.

Використання локального стану React та Context API (див. Рисунок 11) було розглянуто з точки зору простоти та легкості інтеграції з React-компонентами. Однак, ці підходи можуть стати менш зручними для складних операцій зі станом та великих дерев домовленостей.

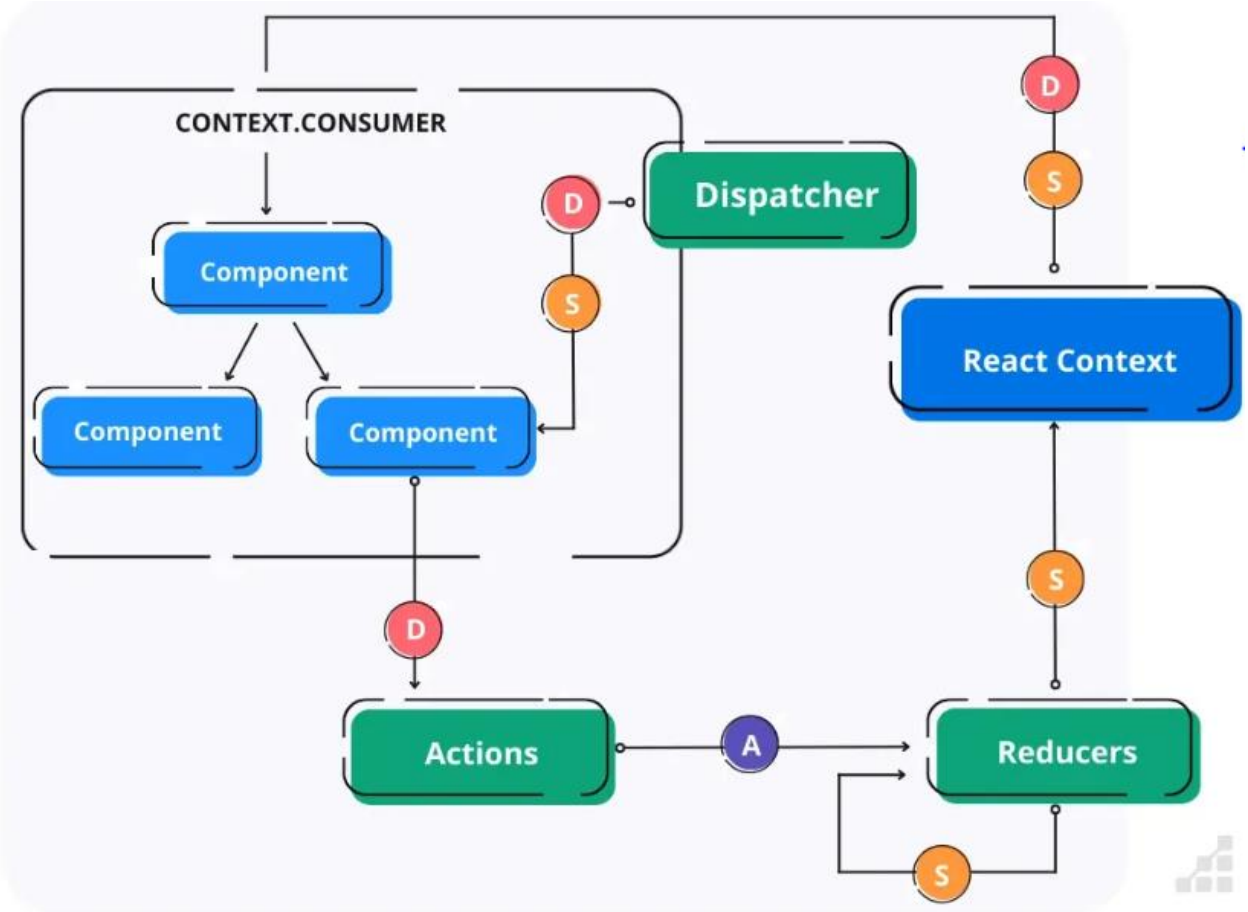

Рисунок 11 — Схема роботи React Context API

MobX було враховано через його простоту використання та акцент на реактивному програмуванні. Проте, для деяких сценаріїв він може виглядати менш декларативним порівняно з іншими рішеннями. Схема роботи MobX виглядаю наступним чином: (див. Рисунок 12)

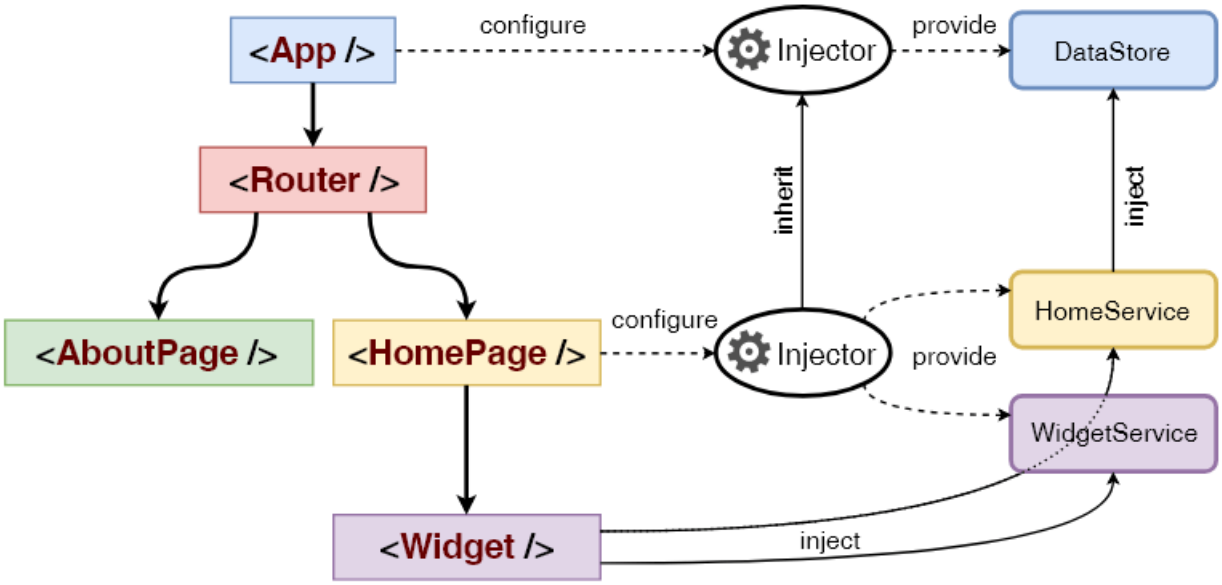

Рисунок 12 — MobX схема роботи

Redux є потужнім інструментом для управління станом додатків у великих та складних веб-проектах. Це становить частину концепції управління станом в React-додатках, але може бути використане і в інших JavaScript-фреймворках.

Його основна ідея полягає в тому, щоб зберігати весь стан додатка у єдиному об'єкті, відомому як "store". Зміни стану відбуваються лише через спеціальні об'єкти, відомі як "actions", які відправляються до "reducers" для обробки. "Reducers" - це чисті функції, які змінюють стан додатка відповідно до отриманих "actions".

Однією з головних переваг Redux є його простота та передбачуваність. Зберігання всього стану у єдиному об'єкті полегшує відладку та визначення причин змін стану. Крім того, це робить стан додатка більш прозорим та легко управляється.

Однак важливо враховувати, що використання Redux може призвести до зайвого бойлерплейту в простих додатках, тому вибір його використання повинен залежати від конкретних потреб проекту та його масштабів. Схема роботи Redux виглядає наступним чином: (див. Рисунок 13)

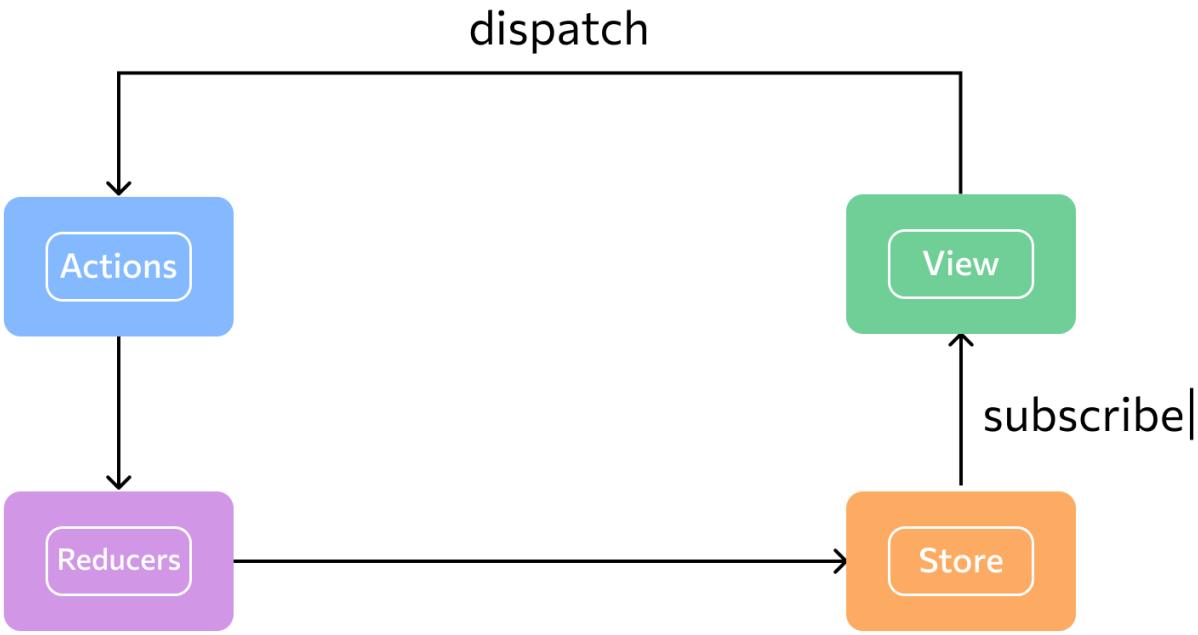

Рисунок 13 — схема роботи Redux сховища

У кінці обрано Redux через його прогнозованість та прозорість змін стану. Унідірекційний потік даних та активна спільнота роблять його ефективним інструментом для управління станом великих та складних десктоп-застосунків. Також, інструменти розробника, такі як Redux DevTools, сприяють легкості вивчення та відлагодженню коду.

2.6 З’єднання застосунку з сервером

Для забезпечення з'єднання нашого десктоп-застосунку з сервером використовується технологія REST API, яка надає зручний та стандартизований спосіб обміну даними між клієнтом та сервером. Реалізація цього підходу дозволяє нам забезпечити ефективну синхронізацію даних та взаємодію з сервером для отримання або оновлення інформації.

Однак, враховуючи специфіку реальних спортивних подій, особливо в гірських районах, де доступ до Інтернету може бути непередбачуваним, ми використовуємо бібліотеку Redux Persist. Ця бібліотека дозволяє зберігати стан додатку локально навіть у випадку відсутності Інтернет-з'єднання.

Використання Redux Persist надає перевагу у випадках, коли спортивні мероприяття можуть відбуватися в умовах обмеженого чи відсутнього Інтернет-з'єднання. Завдяки цьому механізму, наш десктоп-застосунок може продовжувати працювати, зберігаючи та оновлюючи дані локально, і автоматично синхронізуватися з сервером, якщо з'єднання буде відновлено.

Такий підхід дозволяє забезпечити надійність та продуктивність застосунку, навіть у сценаріях з обмеженим доступом до мережі, що важливо для успішної реалізації в специфічних умовах спортивних змагань, особливо в гірських регіонах.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЕСКТОП-ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ СПОРТИВНИХ ЗМАГАНЬ

3.1 Бізнес обґрунтування та вимоги інформаційної системи

У сучасному спорті, де точність та ефективність стають ключовими факторами, наш десктоп-застосунок націлений на оптимізацію та автоматизацію процесів під час організації та проведення спортивних змагань. Це визначається потребою в точності реєстрації атлетів, керування їхніми даними та відслідковуванням їхніх фінішних результатів.

3.2 Вимоги до програмного продукту

Перед початком проектування та розробки інформаційної системи було визначено функціональні та нефункціональні вимоги, яким вона повинна задовольняти.

3.2.1 Функціональні вимоги до програмного продукту

Для початку розробки та повноцінного впровадження інформаційної системи було необхідно додати такі можливості:

- логінація;
- регістрація атлетів ;
- програмування RFID номерків;
- перегляд атлетів, івентів, категорій, етапів;
- оновлення атлетів, івентів, категорій, етапів;

3.2.2 Нефункціональні вимоги до програмного продукту

Ефективність:

- Програмний продукт повинен забезпечувати швидкий доступ до даних.
- Час обробки запиту до серверу, коли атлет перетинає фініш має бути до секунди.
- Програмний продукт повинен бути готовий до масштабування, здатний ефективно працювати при збільшенні обсягу даних та істотному прирості навантаження.

Надійність:

- Програмний продукт повинен забезпечувати стабільну роботу без збоїв та критичних помилок.
- В разі виникнення помилок або відмов системи, програмний продукт повинен забезпечувати коректне відновлення роботи та збереження даних, які в ньому зберігаються.

Легкість використання:

- Інтерфейс користувача повинен бути простим та інтуїтивно зрозумілим для максимального комфорту абсолютної більшості користувачів, які будуть використовувати інформаційну систему.
- Програмний продукт має надавати зручні інструменти для взаємодії з системою та проведення необхідних операцій, які визначені функціональними та нефункціональними вимогами до нього.
- Забезпечення можливості персоналізації інтерфейсу для врахування індивідуальних потреб користувачів.

Безпека:

- Доступ до інформації повинен бути обмеженим та контрольованим.
- Система повинна забезпечувати автентифікацію користувачів**.**

3.3 Розробка інформаційної системи за допомогою Visual Studio Code

Visual Studio Code є відкритим та легким текстовим редактором, розробленим Microsoft. Він став одним з найпопулярніших інструментів для розробників завдяки своїй потужності та розширюваності. Незважаючи на те, що VS Code не є інтегрованою розробкою середовища, він надає велику кількість функцій, які спрощують роботу з кодом.

VS Code підтримує різні мови програмування та фреймворки, забезпечуючи розширену підтримку для HTML, CSS, JavaScript, а також для популярних мов, таких як Python, Java, та інших. Редактор має інтегровану систему контролю версій, можливість використання розширень для покращення функціоналу, а також вбудовані інструменти для відлагодження коду.

Великою перевагою VS Code є його швидкість та легкість використання. Він швидко запускається та взаємодіє з різними типами проектів. Деякі інші особливості включають розширену автодоповнення коду, підтримку розширень для різноманітних завдань, відлагодження в реальному часі та інші.

Загалом, VS Code є відмінним вибором для розробників, які цінують продуктивність, розширюваність та зручний інтерфейс роботи з кодом.
Розробка програмного продукту була розподілена на 3 частини:

- проектування інформаційної системи;
- реалізація програмної частини;
- розробка інтерфейсу застосунку.

3.3.1 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. Рисунок 14).

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.

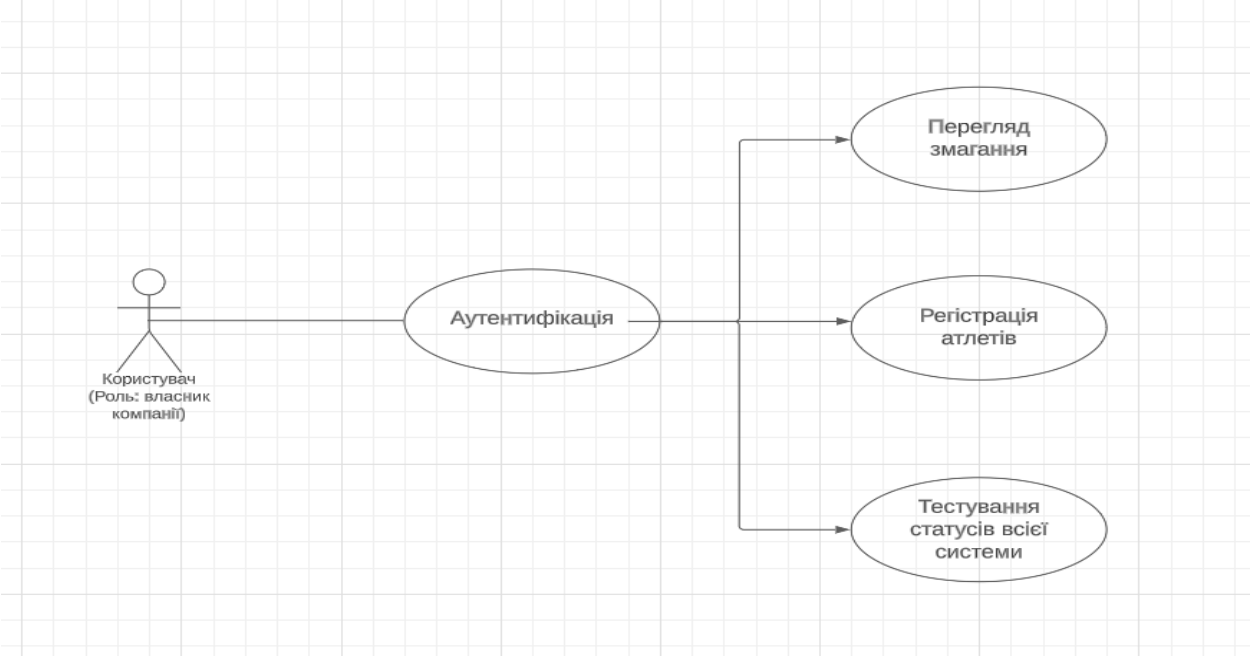

Рисунок 14 — Use-case діаграма

Для того, щоб було зрозуміло, як саме ці можливості користувача були втілені у інформаційну систему, наведена діаграма класів (див. Рисунок 15).

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.

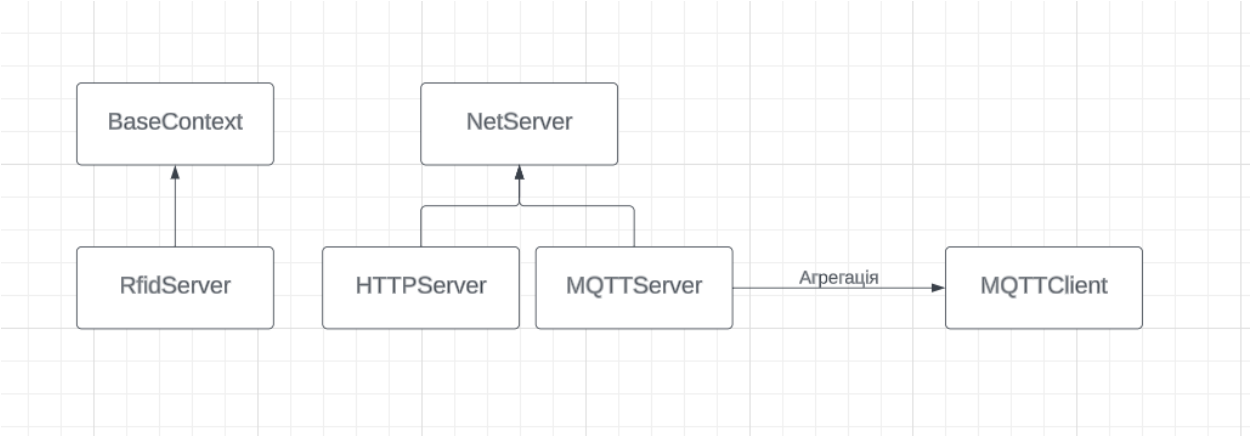

Рисунок 15 — Діаграма класів

3.3.2 Реалізація програмної частини

1. Клас BaseContext.ts головний клас від якого успадковуються інші класи, які регіструються у dependency injection контейнері.
2. Клас RfidServer.ts створено для клієнтської взаємодії з RFID приладом.
3. Клас NetServer.ts абстрактний клас від якого успадковуються інші RFID класи.
4. Клас HTTPServer.cs служить для серверної взаємодії з RFID приладом
5. Клас MQTTServer.ts взаємодіє з імпінч приладом.

Лістинг 1 *Функціонал відправки команди*

```
    private send(commandText: string, saveToRedux = true,
response: string, isSuccess: boolean) {
        const { redux: { dispatch, state } } = this.di;

        const command: ICommandType = {
            command: commandText,
            response: response,
            time: Date.now(),
            isSuccess: isSuccess,
        };

        for (const error of Object.keys(this.errors)) {
            if (!response.includes(error)) {
```

```
                continue;
            }

            if (error !== 'Error 1: Command not
understood') {
                dispatch(setTagStatus(this.errors[error]));
            }
            throw error;
        }

        if (saveToRedux || !isSuccess) {
            const commands = state().rfid.commands;
            const lastCommand = commands[commands.length -
1];
            if (lastCommand?.command.includes(commandText))
{
                dispatch(popCommand());
            }
            dispatch(addRfidCommand(command));
        }
        return command.response;
    }
```

Функція send призначена для обробки та відсилки команд RFID (ідентифікація за допомогою радіочастот) в середовищі Redux. Вона приймає параметри, такі як текст команди, булевий прапор, що вказує, чи потрібно зберегти команду в Redux, рядок відповіді та прапор успішності. Функція перевіряє передбачені помилки у відповіді, відправляє відповідні дії в разі помилок та вирішує зберігання команд RFID в стані Redux. Крім того, вона переконується, що надлишкові команди не зберігаються послідовно в стані. Функція повертає рядок відповіді для подальшого використання.

Лістинг 2 *Функціонал прийняття даних*

```
protected handleConnection(conn: net.Socket) {
        // console.log('NetServer: Connection is oppened.
RFID is listening')
        conn.setEncoding('utf8');
        conn.on('data', (rawReaderData: string) => {
            if (this.socket) {
                this.socket.destroy();
            }
            this.socket = conn;
            const readerDataArray = rawReaderData
```

```
                .split(/[\0\r\n]/)
                .filter((str) => str !== '');

            readerDataArray.forEach((row: string) => {
                const subArray = row.split(/,/);
                console.log('DATA', subArray);
                if (subArray.length >= 4) {
                    switch (subArray[0]) {
                    case 'TAG':
                        if
(subArray[1].match(/^([\w\d]{4}\s?){1,6}$/)) {

this.mainWindow?.webContents.send('rfidTag', subArray);
                        }
                        break;
                    default:
                        break;
                    }
                }
            });
        });
    }
```

Функція handleConnection призначена для обробки з'єднань в мережевому сервері. Функція приймає об'єкт сокету (socket) типу net.Socket. При кожному отриманні даних від з'єднаного пристрою, функція встановлює кодування з'єднання, а потім обробляє отримані рядки.

Під час обробки даних, функція перевіряє існування попереднього з'єднання і в разі його наявності, руйнує його. Потім встановлює нове з'єднання. Отримані дані розділяються на рядки та обробляються окремо.

Кожен рядок поділяється на підмасиви, розділені комами, і виводиться в консоль для налагодження. У разі, якщо рядок містить принаймні чотири елементи, виконується перевірка типу даних. Якщо тип дорівнює 'TAG', і другий елемент відповідає визначеному шаблону, то дані відправляються в головне вікно за допомогою відповідного ім'я події ('rfidTag').

В інших випадках функція не виконує додаткових дій.

Лістинг 3 *Функціонал отримання даних*

```
protected handleMessage(topic:any, message:any) {
        try {
```

```
            const json = JSON.parse(message?.toString());
            if (json['tagInventoryEvent']['epc']){
                const decodedData =
Buffer.from(base64.toByteArray(json['tagInventoryEvent']['e
pc'])).toString('hex');
                this.sendRfidTag(['check', decodedData,
moment(json['timestamp']).valueOf()]);
            }
        } catch (error) {
            console.log('Can not decoded data from rfid',
message);
        }
    }
```

Функція handleMessage відповідає за обробку повідомлень, які надходять по певному темі в контексті комунікації. Функція отримує два параметри: topic (тема повідомлення) та message (саме повідомлення).

У спробі виконати обробку повідомлення, функція спробує розпізнати його у форматі JSON за допомогою JSON.parse(). Якщо це вдається, функція перевіряє наявність елементу epc в ключі tagInventoryEvent об'єкта. Якщо цей елемент існує, то дані розкодовуються з формату Base64, конвертуються у шістнадцятковий формат та відправляються за допомогою функції sendRfidTag() разом із зазначенням типу (['check']), розкодованими даними та міткою часу в момент події.

У випадку виникнення помилки під час розкодування, функція виводить повідомлення про невдалу спробу у консоль, передаючи при цьому саме повідомлення, яке не вдалося розкодувати.

Лістинг 4 *Функціонал відправки декількох команд*

```
private async runCommands(commands: string[] | Function,
IPs : string[] = []) {
        const { alien : { port, username, password, host }
} = config;
        const hosts = Array.isArray(IPs) && IPs.length > 0?
IPs : host;

        await this.start();
        for (let i = 0; i < hosts.length; i++) {
            const host = hosts[i];
```

```
            console.log('runCommands, host=',host);
            try {
                if (this.server) {
                    await this.server.connect({
                        host,
                        port,
                        negotiationMandatory: false,
                        loginPrompt: /Username(>?)/,
                        passwordPrompt: /Password(>?)/,
                        username,
                        password,
                    });
                    await this.sendCommand(username,
false);
                    await this.sendCommand(password,
false);
                } else {
                    throw new Error('The server was
disconnected!');
                }

            } catch (err: any) {
                console.log('Cannot connect:: ' +
err.toString());
                throw new Error('The server was
disconnected!');
            }

            if (Array.isArray(commands)) {
                for (const command of commands) {
                    if (this.server) {
                        await this.sendCommand(command);
                    } else {
                        break;
                    }
                }
            } else if (isFunction(commands)) {
                await commands();
            }

            await this.sendCommand('Save', false);
            await this.sendCommand('exit', false);
        }

        this.stop();
    }
```

Функція runCommands відповідає за виконання команд на віддаленому сервері чи серверах. Функція приймає два параметри: commands (список команд або функція) та необов'язковий масив IPs, що містить IP-адреси серверів. Значення параметрів за замовчуванням визначаються в конфігураційному файлі.

Спочатку функція викликає метод start() для підготовки до виконання команд. Далі, для кожного серверу, вказаного в масиві hosts, функція намагається підключитися до серверу, використовуючи дані авторизації з конфігурації. Якщо з'єднання успішне, вона відправляє ім'я користувача та пароль.

Лістинг 5 *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;
}
```

Функція **entities** визначає редуктор (reducer) для обробки змін у стані додатка, які стосуються сутностей (entities). Вона використовується в контексті управління станом додатка, зокрема в Redux або інших бібліотеках управління станом.

Функція приймає поточний стан (state) та дію (action), і в залежності від визначених умов здійснює відповідні зміни у стані.

Лістинг 6 *Action декоратор*

```
const action = () => {
    return (target: any, propertyKey: string) => {
        const entityName = target.constructor.name;
```

```
        const entityItem = entityName in Entity.mActions ?
Entity.mActions[entityName] : {};
        if (!(propertyKey in entityItem)) {
        entityItem[propertyKey] = {
            watcher: propertyKey,
            actionFunc: (data) =>
actions.action(propertyKey, data),
            isAdded: false,
        };
        }
        Entity.mActions[entityName] = entityItem;
    };
};
```

Функція **action** декоратором, який використовується для додавання додаткової логіки до методів класу. Декоратор призначений для визначення дій (actions) в контексті управління станом за допомогою бібліотеки, такої як Redux або іншої системи управління станом.

Лістинг 7 *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 = [];
            }
        }
    };
```

Функція **saga** є генератором саги (saga generator) для бібліотек, таких як Redux-Saga, яка дозволяє визначати та взаємодіяти із сагами (асинхронними обробниками) на основі методів класів.

Лістинг 8 *Черги атлетів*

```
import { fromJS, List } from "immutable";
import moment from "moment";
import      {      AthleteStageStatus,      AthleteStatus,
ATHLETE_ORDER_MULT,    ENTITY,    IOperativeItem    }    from
"../constants";
import { IContextContainer } from "../container";
```

```
import { IEventCategoryType, IEventType, IStageType,
EntityList, ITopEventResultType } from
"../models/EntityTypes";

const athleteQueue = (ctx: IContextContainer) => {
    return (stageId: string): EntityList<IOperativeItem> =>
{
        const { entities, identity } = ctx.redux.state();

        const currentEvent = entities
            .get(ENTITY.EVENT)
            ?.find(
                (e: IEventType) => e.get('slug') ===
identity.user.current.eventSlug
            );

        const topEventResult = entities
            ?.get(ENTITY.TOP_EVENT_RESULT)
            ?.find(
                (result: ITopEventResultType) =>
result.get('event') === currentEvent?.get('id')
            );
        const categories = entities.get(ENTITY.CATEGORY);
        const users = entities.get(ENTITY.USER);
        const athletes = entities.get(ENTITY.ATHLETE);
        const userEventResults =
entities.get(ENTITY.USER_EVENT_RESULT);

        const currentStage = currentEvent
            ?.get('stages')
            ?.find((stage:IStageType) => stage.get('id') ===
stageId);
```

```
        const stagePause = entities.get(ENTITY.STAGE_PAUSE);
        const    stPause    =    stagePause?.find(pause    =>
pause?.get('stageId') === stageId);

        const                eventCategories                =
currentEvent?.get('categories').sort(
            (x:  IEventCategoryType,  y:  IEventCategoryType)
=> x?.get('order') - y?.get('order')
        );

        const reduceAthletes = (items, athlete) => {
            const  user  =  users?.find(u  =>  u?.get('id')  ===
athlete?.get('user'));
            const cat = categories?.find((c) => c?.get('id')
=== athlete?.get('category'));
            const  evCat  =  eventCategories?.find((ec)  =>
ec?.get('categoryId') == athlete?.get('category'));
            const results = userEventResults?.find(
                                uer  =>  uer?.get('userId')
=== user?.get('id') &&
                                uer?.get('stageId')      ===
currentStage?.get('id')
            );
            const aStage = athlete?.get('stages')?.find(s =>
s.get('stageId') === currentStage?.get('id'));
            const                  inOrder                  =
Number.parseInt((aStage?.get('caOrder')                    /
ATHLETE_ORDER_MULT).toFixed())
            const  inCatID  =  eventCategories?.find((ec)  =>
ec?.get('order') == inOrder)?.get('categoryId');
            const    inCat    =    categories?.find((c)    =>
c?.get('id') === inCatID);
```

```
            const item = fromJS({
                userId: user?.get('id'),
                athleteId: athlete?.get('id'),
                stageId: currentStage?.get('id'),
                stageStatus: currentStage?.get('status'),
                firstName: user?.get('firstName'),
                lastName: user?.get('lastName'),
                number: athlete?.get('number'),
                aOrder: athlete?.get('order'),
                rfid: athlete?.get('rfid'),
                cOrder: evCat?.get('order'),
                place: aStage?.get('place'),
                caOrder: aStage?.get('caOrder'),
                falseStart: aStage?.get('falseStart'),
                categoryId: cat?.get('id'),
                categoryName: cat?.get('categoryName'),
                color: cat?.get('colorIndex'),
                status: aStage?.get('status'),
                inCat: {
                    id: inCatID,
                    categoryName:
inCat?.get('categoryName'),
                    color: inCat?.get('colorIndex')
                },
                results: {
                    uuid: results?.get('uuid'),
                    startTime: results?.get('startTime'),
                    endTime: results?.get('endTime'),
                    duration: results?.get('duration') || 0,
                    difference: 0,
                },
                planStartTime: 0
            }) as IOperativeItem;
```

```
            console.log(`----------  results  startTime  for
${user?.get('firstName')}          ${user?.get('lastName')}:
${results?.get('startTime')}`);

            return items.push(item);
        };

        let startTime = currentStage?.get('startTime');

        const reduceCategories = (items, cats, key: string)
=> {
            const cOrder = Number.parseInt(key);
            for (let i = 0; i < cats.size && cOrder > 0; i++)
{
                let a = cats?.get(i);
                const  falseStart  =  a?.get('falseStart')  *
60000;
                if            (a?.get('status')            ==
AthleteStageStatus.FIRST_STAGE) {
                    startTime                                =
currentStage?.get('startTime');
                }    else    if    (a?.get('status')    ==
AthleteStageStatus.FIRST_CATEGORY) {
                    const              cDelay               =
currentStage?.get('categoryDelay') * 60000;
                    startTime += falseStart + cDelay;
                } else {
                    const              aDelay               =
currentStage?.get('delay') * 60000;
                    startTime += falseStart + aDelay;
                }
                if (a?.has('results')) {
                    const top = topEventResult
```

```
                        ?.get('stages')
                        ?.find(topStage                    =>
topStage?.get('stage') === currentStage?.get('id'))
                        ?.get('results')
                        ?.find((stageResult:     any)     =>
stageResult?.get('categoryId') === a?.get('categoryId'));
                    const               diff               =
a?.getIn(['results','duration']) - top?.get('time');
                    a  =  a?.setIn(['results',  'difference'],
diff);
                }
                const     tempStPause     =     stPause     ?
stPause.get('pause') : 0;

                const aa = a?.set('planStartTime', startTime
+ tempStPause);
                items = items.push(aa);

            }
            return cOrder === 0 ? items.concat(cats): items;
        };
        return athletes
            ?.filter(athlete =>
                athlete?.get('event')                      ===
currentEvent?.get('id') &&
                athlete?.get('status')                     ===
AthleteStatus.REGISTERED
            )
            ?.reduce(reduceAthletes, List<IOperativeItem>())
            ?.sort((x,    y)    =>    x?.get('caOrder')    -
y?.get('caOrder'))
            ?.filter(a => a?.get('number'))
```

```
            ?.groupBy(a      =>      (a?.get('caOrder')      /
ATHLETE_ORDER_MULT).toFixed())
            ?.reduce(reduceCategories,
List<IOperativeItem>());
    };
};

export default athleteQueue;
```

Функція **athleteQueue** призначена для формування списку оперативних об'єктів, що представляють атлетів в черзі на конкретному етапі події. Вона отримує дані з Redux та контексту, визначає поточну подію та етап, фільтрує та обробляє дані атлетів, групує їх за категоріями, визначає планований час старту та повертає оновлений список атлетів.

Функція використовує Immutable.js для ефективного оброблення та маніпулювання структурами даних. Кожен оперативний об'єкт містить інформацію про атлета, його статус на етапі, категорію, номер, RFID і багато іншого.

Всі дії функції спрямовані на те, щоб забезпечити коректне та структуроване представлення даних про атлетів у контексті управління станом додатка для спортивних подій.

Лістинг 8 *Клас Guard для реалізації ролей у проекті*

```
import Acl from './Acl';
import { ROLE, IRoles, IRules, IIdentity, ISecretRole } from
'./types';

class Guard {

    get isCheckAuth() { return this.mIsCheckAuth; }
    set isCheckAuth(value) { this.mIsCheckAuth = value; }
```

```
    private mAcl: Acl;
    private mRoles: IRoles;
    private mRules: IRules;
    private mCurrent: ISecretRole;

    private mIsCheckAuth: boolean;

    constructor(roles: IRoles, rules: IRules) {
        this.mRoles = roles;
        this.mRules = rules;
    }

    public get acl(): Acl { return this.mAcl; }
    public get roles(): IRoles { return this.mRoles; }
    public get rules(): IRules { return this.mRules; }

    private isRouteMatch(path: string): string {

        // allow to use '/' in end of path
        if (path.length > 1 && path.substr(path.length - 1)
=== '/') {
            path = path.substr(0, path.length - 1);
        }
        const  pParts = path && path.split('/');

        for (const resource in this.mRules) {
            if (this.mRules.hasOwnProperty(resource)) {
                const rParts = resource.split('/');
                if (rParts.length >= pParts.length) {
                    let result = null;
                    for (let i = 0; i < rParts.length; i++)
{
```

```
                        if (!(rParts[i] === pParts[i] ||
rParts[i] === '*')) {
                            result = null;
                            break;
                        }
                        result = resource;
                    }
                    if (result) {
                        return result;
                    }
                }
            }
        }
        return null;
    }

    public inRouter(resource: string){
        let isRouter = false;
        try {
            if (this.mRules.hasOwnProperty(resource)) {
                isRouter = true;
            } else {
                const match = this.isRouteMatch(resource);
                if (match) {
                    if (this.mRules.hasOwnProperty(match)) {
                        isRouter = true;
                    }
                }
            }
        } catch (e) {
            isRouter = false;
        }
        return isRouter;
```

```
    }

    public allow(resource: string, grant: string, secret: string = null, role: string = null): boolean {
        let s = secret === null ? this.mCurrent.secret : secret;
        s = s ? s + ':' : '';
        if (!role) {
            role = this.mCurrent.role;
        }
        let isAllowed = false;
        try {
            if (this.mRules.hasOwnProperty(resource)) {
                isAllowed = this.mAcl.isAllowed(
                    s + role,
                    s + resource,
                    grant,
                );
            } else {
                const match = this.isRouteMatch(resource);
                if (match) {
                    if (this.mRules.hasOwnProperty(match)) {
                        isAllowed = this.mAcl.isAllowed(
                            s + role,
                            s + match,
                            grant,
                        );
                    }
                }
            }
        } catch (e) {
            isAllowed = false;
        }
```

```
        return isAllowed;
    }

    public build(identity: IIdentity) {
        this.mCurrent = identity.current;
        const acl = new Acl();
        this.init(acl);
        identity.secrets.forEach((item) => {
            this.init(acl, item);
        });

        this.mAcl = acl;
        return this;
    }

    private init(acl: Acl, secretRole: ISecretRole = null) {
        const s = secretRole !== null ? secretRole.secret +
':' : '';
        for (const role in this.mRoles) {
            if (this.mRoles.hasOwnProperty(role)) {
                const item = this.mRoles[role];
                const             parentRoleId             =
item.hasOwnProperty('parent') ? item.parent : null;
                let parentRole: string[] = null;
                if (parentRoleId !== null) {
                    parentRole = [];
                    parentRoleId.forEach((pRole)          =>
parentRole.push(s + pRole));
                }
                const pr: string[] | string  =  secretRole
!== null && role === ROLE.GUEST ? ROLE.GUEST : parentRole;
                acl.addRole(s + role, pr);
            }
```

```
        }

        for (const resource in this.mRules) {
            if (this.mRules.hasOwnProperty(resource)) {
                if (secretRole === null) {
                    acl.addResource(resource);
                } else {
                    acl.addResource(s + resource, resource);
                }
            }
        }

        for (const resource in this.mRules) {
            if (this.mRules.hasOwnProperty(resource)) {
                const grant = this.mRules[resource];
                const res = s + resource;
                if (grant.hasOwnProperty('allow')) {
                    for (const role in grant.allow) {
                        if
(grant.allow.hasOwnProperty(role)) {
                            const          grants          =
grant.allow[role];
                            for   (let   i   =   0,   size   =
grants.length; i < size; i++) {
                                acl.allow(s  +  role,  res  ,
grants[i]);
                            }
                        }
                    }
                if (grant.hasOwnProperty('deny')) {
                    for (const role in grant.deny) {
```

```
                        if (grant.deny.hasOwnProperty(role))
{
                            const grants = grant.deny[role];
                            for (let i = 0, size =
grants.length; i < size; i++) {
                                acl.deny(s + role, res ,
grants[i]);
                            }
                        }
                    }
                }
            }
        }
    }
}
export default Guard;
```

Клас **Guard** визначає систему контролю доступу (ACL) для авторизації в додатку. Він використовує об'єкт Acl для визначення правил доступу до ресурсів на основі ролей користувачів та інших параметрів авторизації. Клас забезпечує методи для перевірки дозволу на виконання дій, визначення належності маршрутів до правил та ініціалізації системи контролю доступу.

Основні функції включають в себе визначення властивостей та конструктор для ініціалізації об'єкта, методи для визначення відповідності маршрутів правилам, перевірки дозволу для ресурсів та ініціалізації системи контролю доступу на основі ідентифікаційного об'єкта користувача.

Усе це спрощує реалізацію авторизації в додатку та забезпечує зручний інтерфейс для управління правами доступу користувачів.

Лістинг 10 *TypeScript типи*

```
export enum ROLE {
    GUEST = 'guest',
```

```
    ATHLETE = 'athlete',
    OPER = 'oper',
    OWNER = 'owner',
    ADMIN = 'admin',
}

export interface CategoryAthletes {
    [category: string]: {
        id?: string;
        athletes: any[],
        description: string,
        color: number,
        name: string,
    };
}

export enum GRANT {
    // for business logic
    READ = 'read',
    WRITE = 'write',
    EXECUTE = 'execute',

    // for http requests
    GET = 'GET',
    POST = 'POST',
    PUT = 'PUT',
    DELETE = 'DELETE',
}

export interface ISecretRole {
    role: ROLE;
    companyId: any;
    companySlug: string;
```

```
    eventId: any,
    eventSlug: string;
    secret: string;
    url: string;
}

export interface IIdentity {
    userId: any;
    firstName: string;
    lastName: string;
    slug: string;
    userEmail: string;
    token?: string;
    guard?: Guard;
    current: ISecretRole;
    secrets: ISecretRole[];
    locale: string;
    timezone: string;
    avatar?: string;
}

export interface IRoleData {
    display: string;
    url: string;
    parent?: ROLE[];
    private?: boolean;
}

export interface IRoles {
    [key: string]: IRoleData;
}

export interface IGrants {
```

```
    [key: string]: string[];
}

export interface IAllowDeny {
    allow: IGrants;
    deny?: IGrants;
}

export interface IRules {
    [key: string]: IAllowDeny;
}

export interface IIdentityACL {
    user: any;
    roles: IRoles;
    rules: IRules;
}
```

У даному файлі визначено перелік ролей користувачів у системі (enum ROLE), структури для представлення інформації про категорії атлетів, ролі користувача з урахуванням конкретної компанії та події, а також інтерфейси для системи контролю доступу, правил та різноманітних дозволів. Ці конструкції надають структурований та типізований підхід до визначення ролей, правил та інших параметрів для подальшого використання у TypeScript-проектах.

Лістинг 11 *Створення нового Electron вікна*

```
import path from 'path';
import shortid from "shortid";
import {app, BrowserWindow } from 'electron';
import { IContextContainer } from "../container";
import { resolveHtmlPath } from '../main/util';
```

```
const RESOURCES_PATH = app.isPackaged
? path.join(process.resourcesPath, 'assets')
: path.join(__dirname, '../../assets');


const getAssetPath = (...paths: string[]): string => {
    return path.join(RESOURCES_PATH, ...paths);
};

const newWindow = (ctx: IContextContainer) => {
    return (url: string, title: string, height: number,
width: number) => {
        const mainWindow = new BrowserWindow({
            show: false,
            width,
            height,
            icon: getAssetPath('icon.png'),
            autoHideMenuBar: true,
            webPreferences: {
                webSecurity: false,
                preload: app.isPackaged
                ? path.join(__dirname, 'preload.js')
                :                       path.join(__dirname,
'../../.erb/dll/preload.js'),
            },
        });

        mainWindow
            .loadURL(resolveHtmlPath('index.html',
shortid.generate(), url))
            .then(() => mainWindow.setTitle(title));
```

```
    }
};

export default newWindow;
```

Функція **newWindow** є фабричною функцією, яка приймає контекст (ctx) та повертає функцію для створення нового вікна. При створенні вікна використовуються параметри, такі як URL, заголовок, висота та ширина. Вікно створюється з прихованим станом, встановлюються його характеристики (зокрема, іконка та безпека веб-застосунку), та завантажується вказаний URL.

Лістинг 15 *Job клас для роботи з кронами*

```
import BaseContext from './BaseContext';
import {action} from './store/actions';

export default class JobHandler extends BaseContext {
    private _interval: any;
    private _0_3_sec: number;
    private _5_sec: number;
    private _90_sec: number;

    constructor(opts: any) {
        super(opts);
        this._0_3_sec = 0;
        this._5_sec = 0;
        this._90_sec = 0;
    }

    public start = () => {
        this._interval = setInterval(this.tick, 100);
    };
```

```
public stop = () => {
    clearInterval(this._interval);
};

private tick = () => {
    // 0.33 second interval
    this._0_3_sec += 1;
    if (this._0_3_sec === 3) {
        this.task_0_3_sec();
        this._0_3_sec = 0;
    }

    // 1 second interval
    this._5_sec += 1;
    if (this._5_sec === 50) {
        this.task_5_sec();
        this._5_sec = 0;
    }

    // 90 seconds interval
    this._90_sec += 1;
    if (this._90_sec === 900) {
        this.task_90_sec();
        this._90_sec = 0;
    }
};

// the task list that should handle every 0.3 seconds
private task_0_3_sec() {
}

// the task list that should handle every 5 seconds
private task_5_sec() {
```

```
        this.synchronization();
    }

    // the task list that should handle every 90 seconds
    private task_90_sec() {
    }

    // ------------------ tasks implementations ------------
---------------

    private synchronization() {
        const {
            redux: {state, dispatch},
        } = this.di;
        const {changes} = state();
        if (changes?.size > 0) {
            changes.mapKeys((k: string) => {
                if (changes.get(k).size > 0) {
                    const type = 'sync' + k[0].toUpperCase()
+ k.slice(1);
                    dispatch(action(type, {}));
                }
            });
        }
    }
}
```

Клас **Job** розширює базовий клас BaseContext. Цей клас служить для обробки різноманітних завдань (tasks) з різними інтервалами виклику. Кожне завдання викликається з певною періодичністю, яку визначено в межах методу tick.

Клас має внутрішні змінні для відстеження часу для різних інтервалів (0.33 секунди, 5 секунд, 90 секунд). У методі tick ці значення збільшуються, і якщо досягається встановлене значення інтервалу, викликаються відповідні методи завдань (task_0_3_sec, task_5_sec, task_90_sec).

Також у класі є конструктор, який встановлює початкові значення для інтервалів, та методи start та stop для запуску та зупинки циклу обробки завдань.

Метод synchronization використовує об'єкт redux для отримання стану та диспетчера, щоб синхронізувати зміни за допомогою дій, визначених у функції action. Це відбувається відповідно до розкладу обробки завдань (5-секундний інтервал у даному випадку)

.

3.3.3 Розробка інтерфейсу застосунку

Для розробки клієнтської частини використовувався React та Tailwind. Розроблений інтерфейс є зрозумілим та простим у використанні.

Під час активного процесу програмування, стан кнопки миттєво змінюється на відображення лоадера, що передає користувачеві інформацію про те, що система знаходиться в режимі виконання операції. Цей індикатор надає важливого візуального підтвердження активності та виконання програмування RFID номерків. Модальне вікно виклядає наступним чином: (див. Рисунок 16)

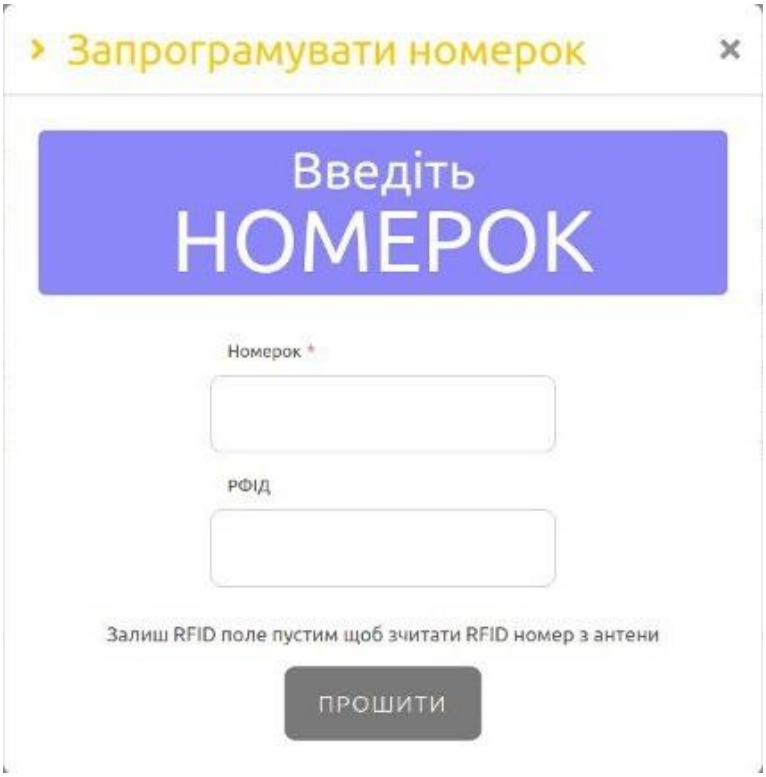

Рисунок 16 — Вікно з програмуванням RFID стрічки

В системі передбачено використання модального вікна з трьома вкладками, яке спрощує процес діагностики та контролю роботи RFID-системи.

Перша вкладка: Системна перевірка

Перша вкладка призначена для комплексної перевірки роботоспроможності системи. Тут оператор може визначити, чи працює система коректно. Перевіряється з'єднання з сервером, стан RFID-антенн, а також встановлюється лінк із станціями на кожному з етапів спортивного заходу. Інформація про здоров'я системи надається оператору у зручному форматі.

Друга вкладка: Перевірка антенн RFID

Друга вкладка служить для відокремленої перевірки роботи RFID-антенн. На цьому етапі RFID-ридер розпочинає зчитування всіх номерів, що знаходяться в межах дії антенн, та відображення їх на екрані. Це дозволяє

оператору перевірити ефективність кожної антени окремо, забезпечуючи високу точність роботи.

Третя вкладка: Історія команд

У третій вкладці доступна історія всіх команд, що були відправлені на RFID-ридер. Команди представлені в хронологічному порядку, починаючи з найновіших і закінчуючи найстарішими. Це надає оператору можливість відстежувати та аналізувати всі взаємодії з рідером впродовж часу.

Такий організований модальний інтерфейс дозволяє операторам здійснювати швидку та ефективну діагностику, забезпечуючи надійний контроль за роботою RFID-системи на кожному етапі її функціонування. Модальне вікно тестування системи виклялає наступним чином (див. Рисунок 17)

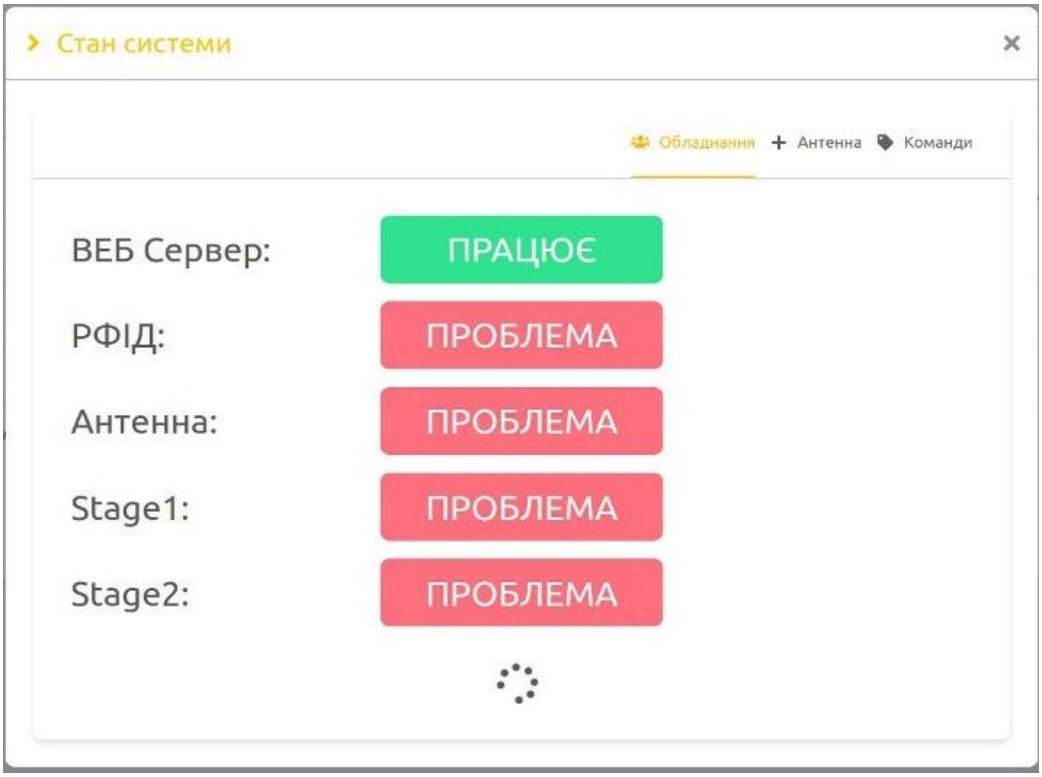

Рисунок 17 — Вікно з програмуванням тестуванням системи

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. Рисунок 18)

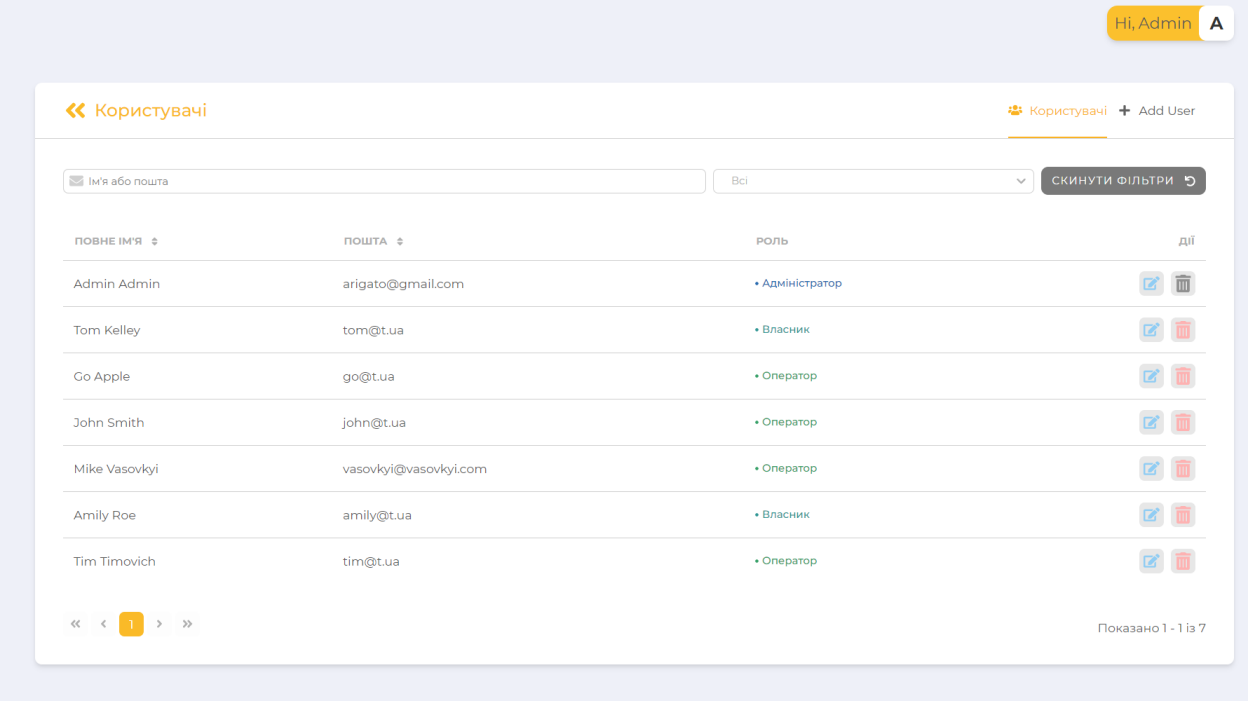

Рисунок 18 — приклад сторінки менеджменту користувачів з адмінської сторони

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. Рисунок 19).

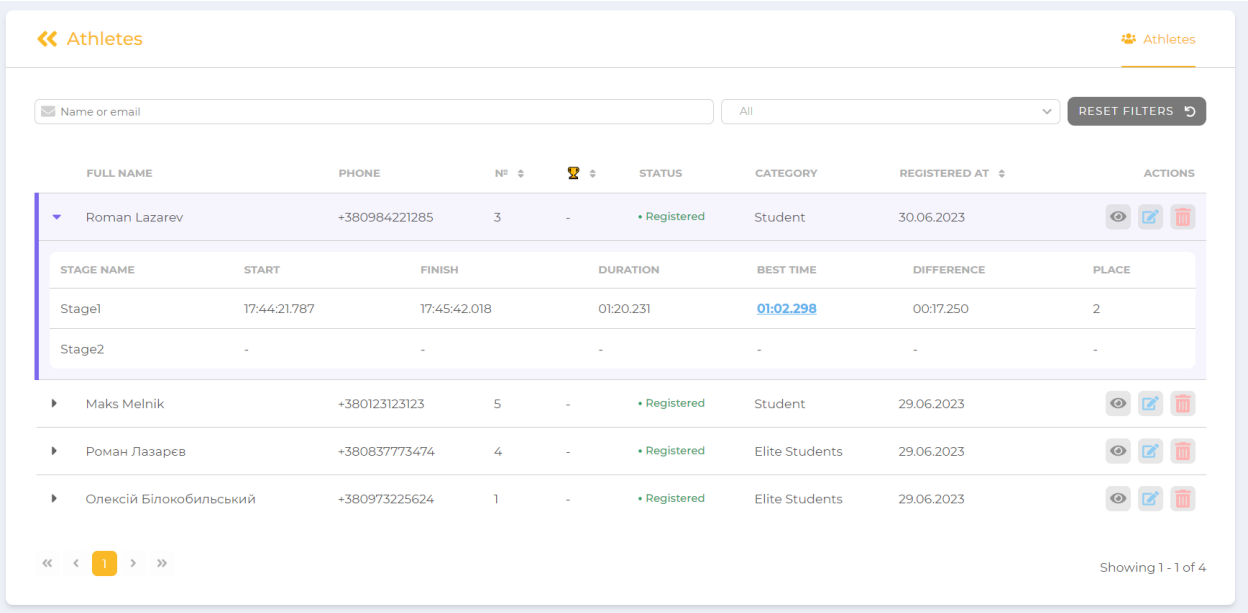

Рисунок 19 — Приклад вигляду сторінки спортсменів

На сторінці **Categories** можна побачити списки категорій подій (див. Рисунок 20). 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.

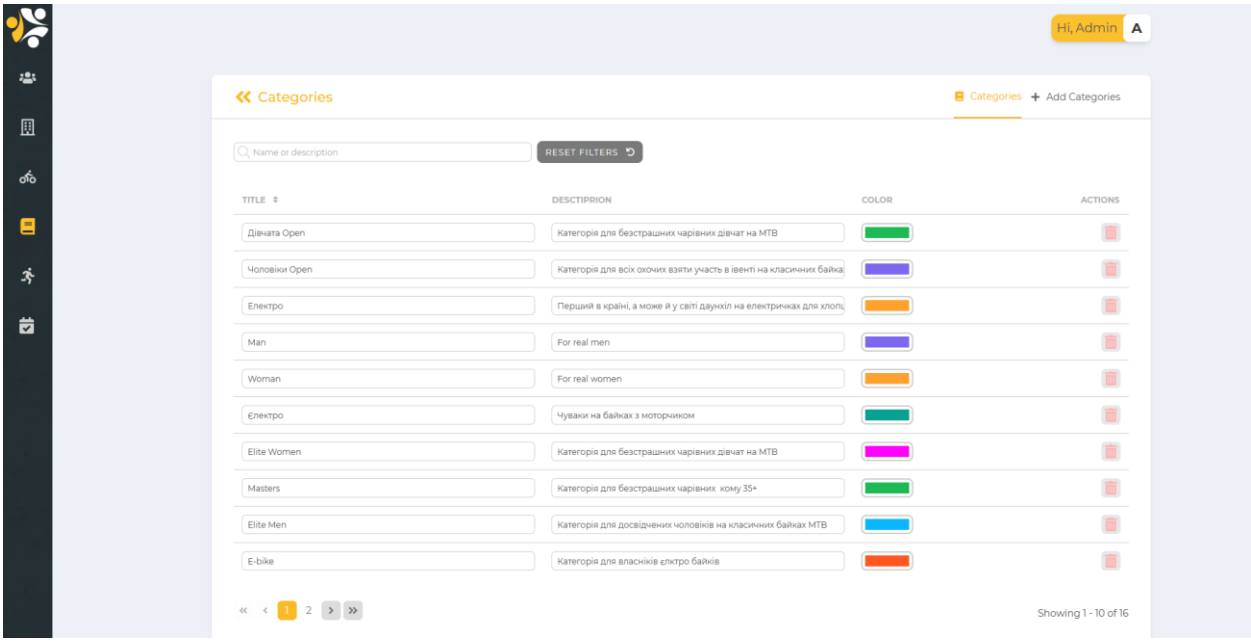

Рисунок 20 — Приклад вигляду сторінки **Categories**

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. Рисунок 21).

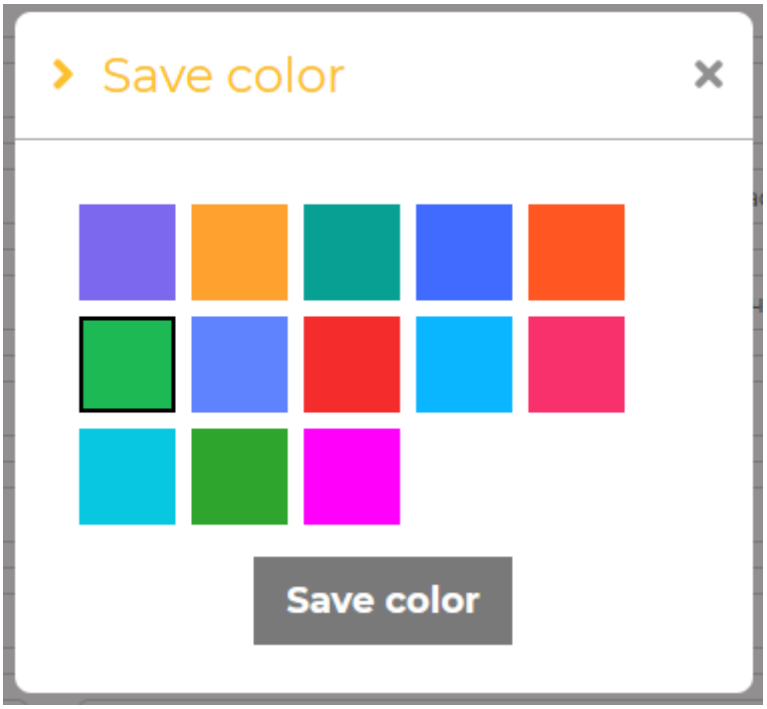

Рисунок 21 — Приклад вигляду модального вікна **Save Color**

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. Рисунок 22).

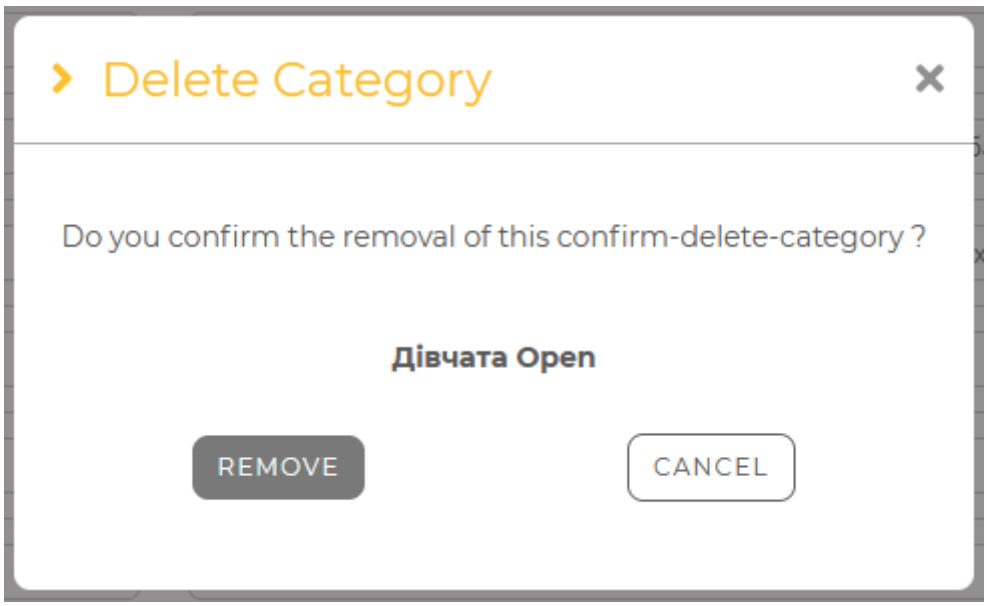

Рисунок 22 — Приклад вигляду модального вікна **Delete Category**

Для того щоб редагувати дані компаній, потрібно натиснути на кнопку напроти вибраної компанії, після цього відбудеться перехід на сторінку **Edit Company**. 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. Рисунок 23).

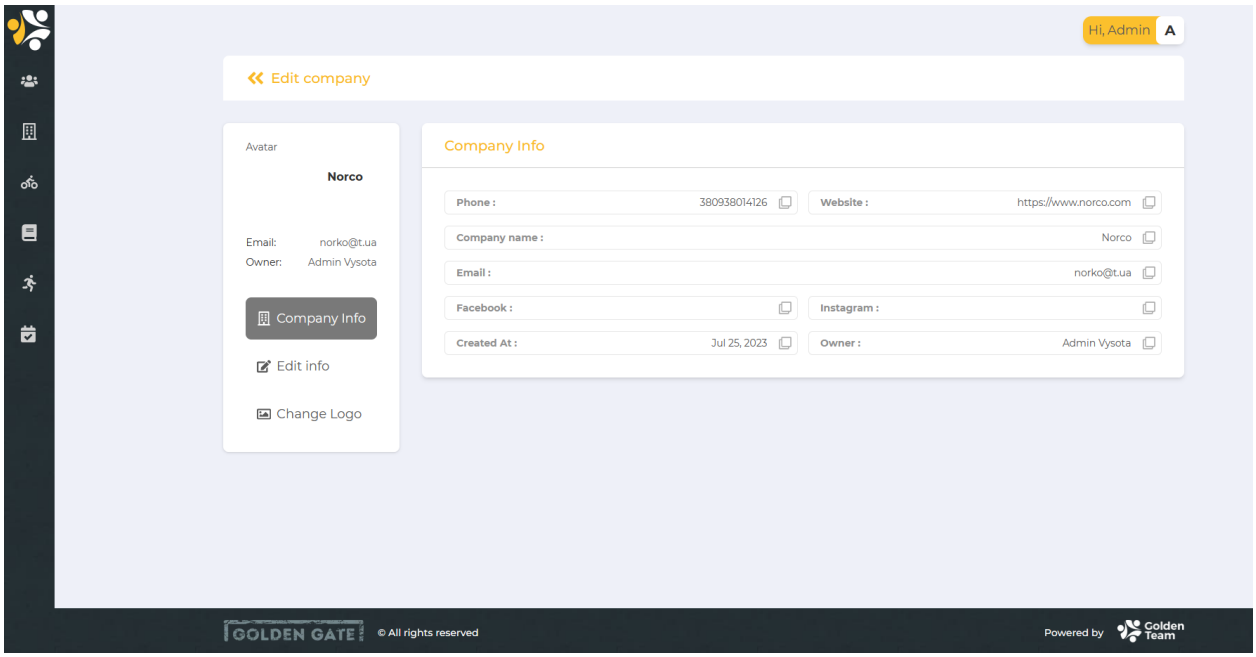

Рисунок 23 — Приклад вигляду сторінки **Edit Company**

На сторінці **Events** можливо подивитись список подій, в якому є дані: назва події, дата та час початка події, статус та кількість етапів (див. Рисунок 24).

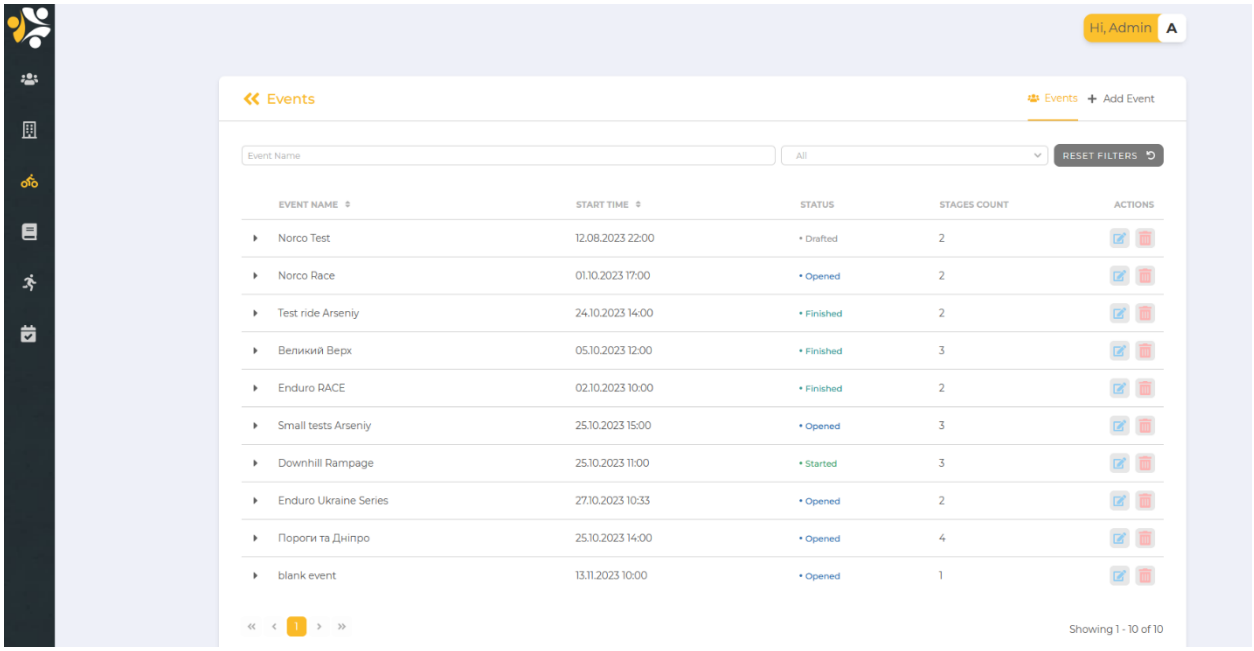

Рисунок 24 — Приклад вигляду сторінки **Events**

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. Рисунок 25).

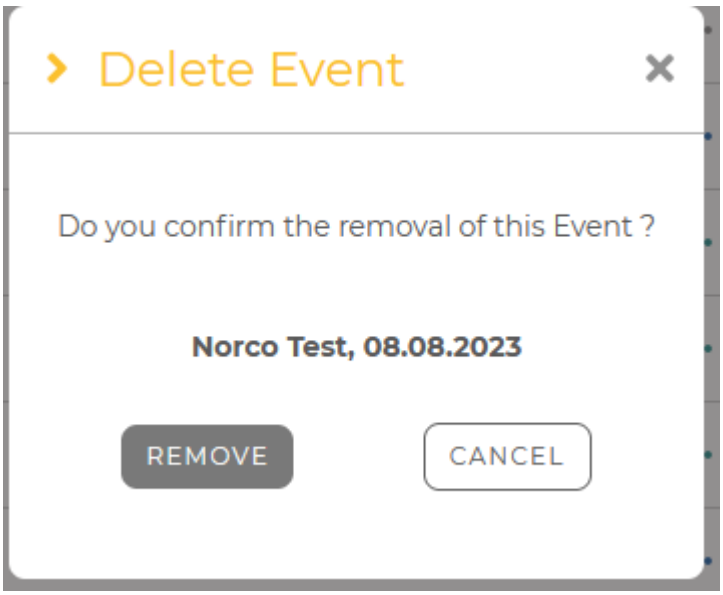

Рисунок 25 — Приклад вигляду модального вікна **Delete Event**

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).

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 **Payment info** дозволяє заповнити платіжну інформацію. **Cover image** та **Change logo** дозволяє змінити дозволяє змінити банер події та логотип події відповідно. **Change categories** дозволяє додати категорію за допомогою кнопки **Add** до обраної події. **Change stages** дозволяє додати стадію. За допомогою кнопки **Add Stage** відкривається модальне вікно для заповнення даних стадії. Після введення всіх даних у модальному вікні зміни можливо зберегти за допомогою кнопки **Save**.

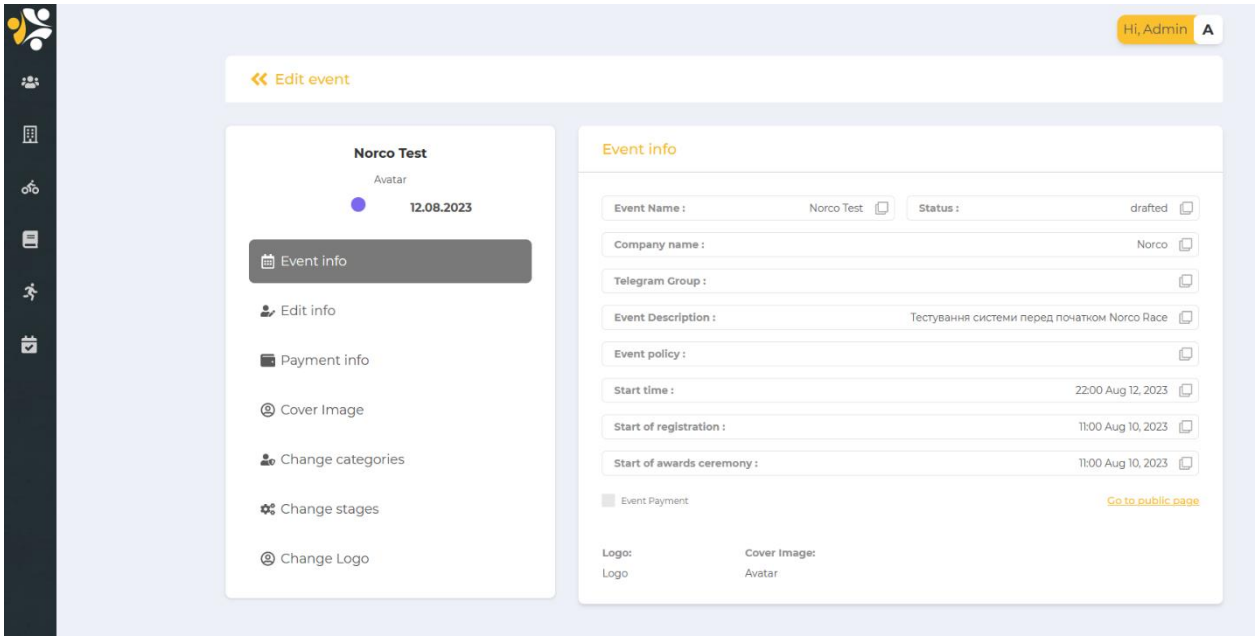

Рисунок 26 — Приклад вигляду сторінки **Edit Event**

Модальне вікно редагування інформації про івент (див. Рисунок 27) спрямоване на оптимізацію управління подіями в системі. Оператор має можливість легко змінювати ім'я івенту, щоб відобразити будь-які оновлення або корекції. Додатково, вікно дозволяє встановлювати або змінювати пов'язану з івентом телеграм-групу, полегшуючи комунікацію та обмін інформацією. Крім того, оператор може редагувати час початку івенту, надаючи гнучкість у визначенні та коригуванні параметрів. Це вікно створено для забезпечення зручності та ефективності в управлінні даними про івенти в системі.

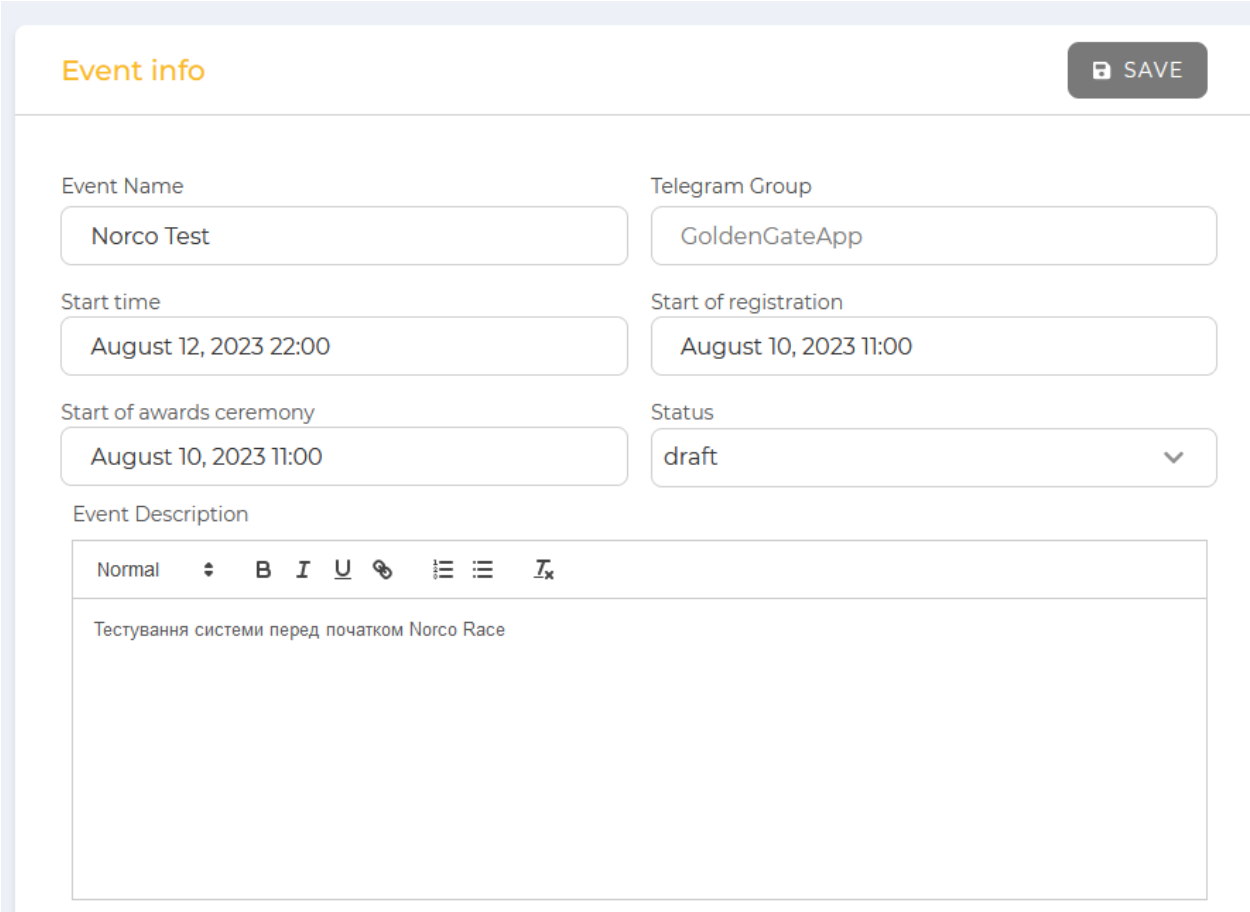

Рисунок 27 — Приклад вигляду сторінки **Edit info**

Модальне вікно (див. Рисунок 28), використовуючи плагін Dropzone, впроваджено з метою зручного додавання банера до івенту в системі. Завдяки використанню плагіна Dropzone, оператори можуть швидко та легко завантажувати зображення, перетягуючи їх у відповідну зону в модальному вікні. Цей інтуїтивний інтерфейс спрощує процес додавання банера до конкретного івенту.

Можливість додавання або зміни банера шляхом вибору відповідної зони в модальному вікні дає операторам широкі можливості оновлення візуальних елементів івентів. Це не тільки прикрашає представлення конкретного заходу, але й покращує його візуальну привабливість для користувачів системи.

Важливою функцією є підтримка візуалізації івенту за допомогою завантаженого банера. Це не лише додає естетику, але і робить інформацію

про івент більш доступною та легко сприйманою для всіх користувачів системи.

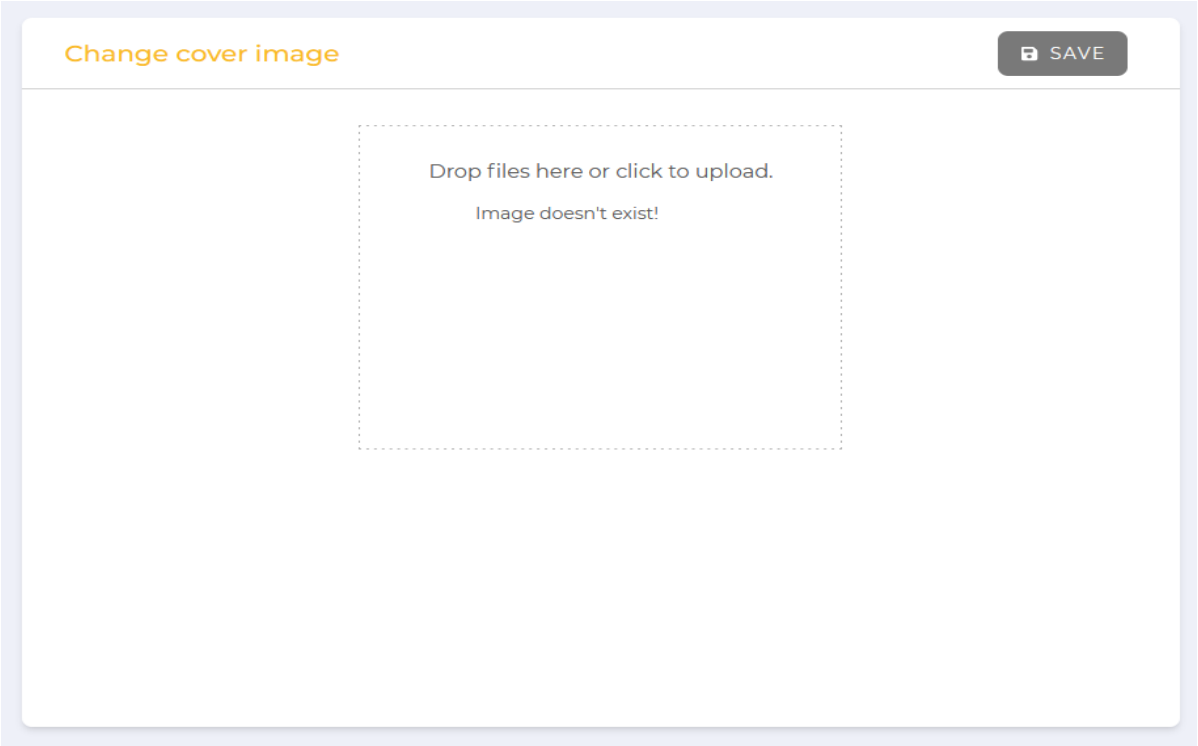

Рисунок 28 — Приклад вигляду сторінки **Cover image** та **Change logo**

Модальне вікно для редагування категорій в івенті (див. Рисунок 28) забезпечує операторам зручний інструмент для налаштування та оптимізації структури спортивного заходу. Опція додавання нових категорій дозволяє адаптувати івент до різноманітності учасників та їх вмінь, забезпечуючи широкі можливості персоналізації. Видалення категорій стає ефективним способом управління списком та підтримки оновлення структури івенту.

Окрім того, можливість зміни послідовності категорій дозволяє операторам легко адаптувати відображення івенту для забезпечення зручності та логічного порядку в управлінні категоріями. Це відкриває широкі перспективи для індивідуалізації та оптимізації відображення даних про івент у системі.

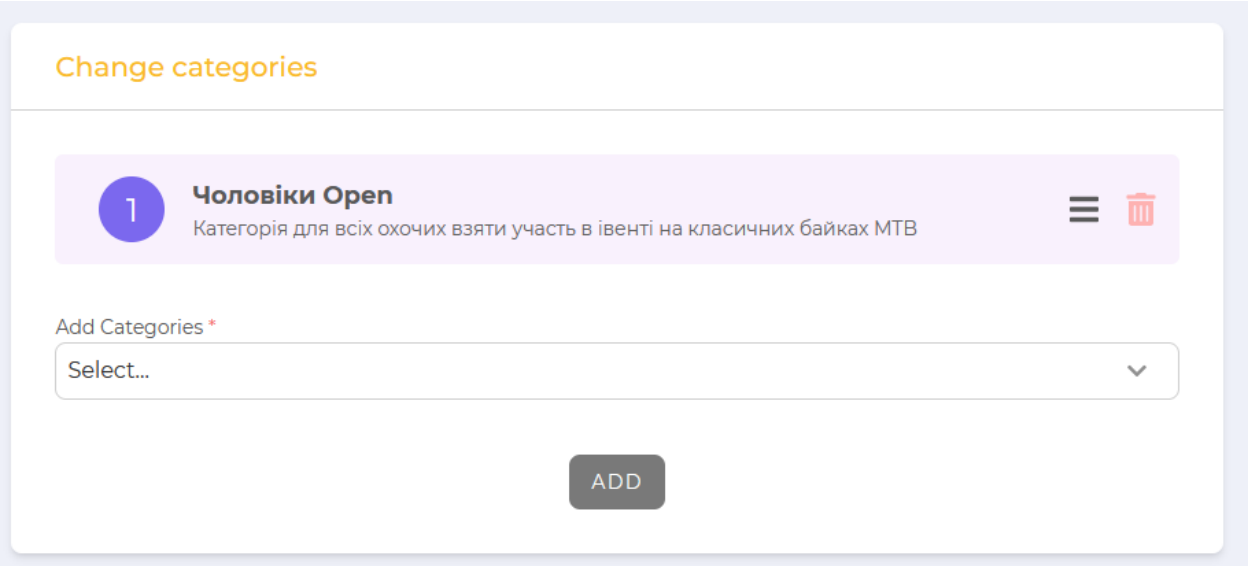

Рисунок 29 — Приклад вигляду сторінки **Change categories**

Сторінка перегляду стейджів (див. Рисунок 30) в івенті призначена для надання операторам повної інформації та зручних інструментів управління процесом старту для кожного стейджу. На цій сторінці доступні такі елементи:

Для кожного стейджу відображаються його ім'я, час старту, поточний статус (наприклад, "Активний" або "Завершено"), а також інформація про затримку між атлетами та категоріями.

Кожен стейдж супроводжується кнопками "Редагувати" і "Видалити", що надають операторам зручний інструмент для внесення змін або видалення стейджу за потреби.

Ця сторінка має на меті спростити та оптимізувати контроль над стартом івенту для ефективного ведення спортивних змагань.

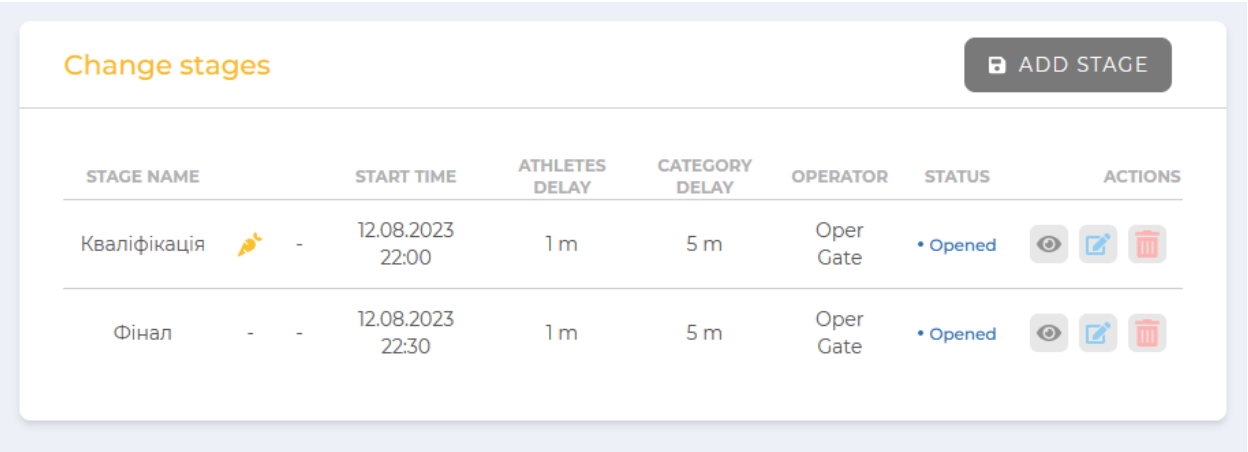

Рисунок 30 — Приклад вигляду сторінки **Change stages**

Модальне вікно для додавання стейджа (див. Рисунок 31) розроблено з метою забезпечення операторам зручного інтерфейсу для налаштування нового етапу івенту. У цьому вікні доступні наступні опції:

Оператор може вказати ім'я для нового стейджу та додати опис, що допоможе ідентифікувати та пояснити призначення етапу.

Забезпечена можливість завантажити зображення, яке візуально представлятиме стейдж. Дропзона для картинки дозволяє зручно додавати та оновлювати візуальні елементи.

Оператор може встановити час початку стейджу та задати часову затримку між атлетами та категоріями для ефективного контролю за ходом івенту.

Оператор отримує можливість вказати додаткові параметри, такі як часові задачі, які пов'язані з кожним стейджем.

Це модальне вікно забезпечує інтуїтивно зрозумілий та зручний інтерфейс для додавання нового етапу до івенту.

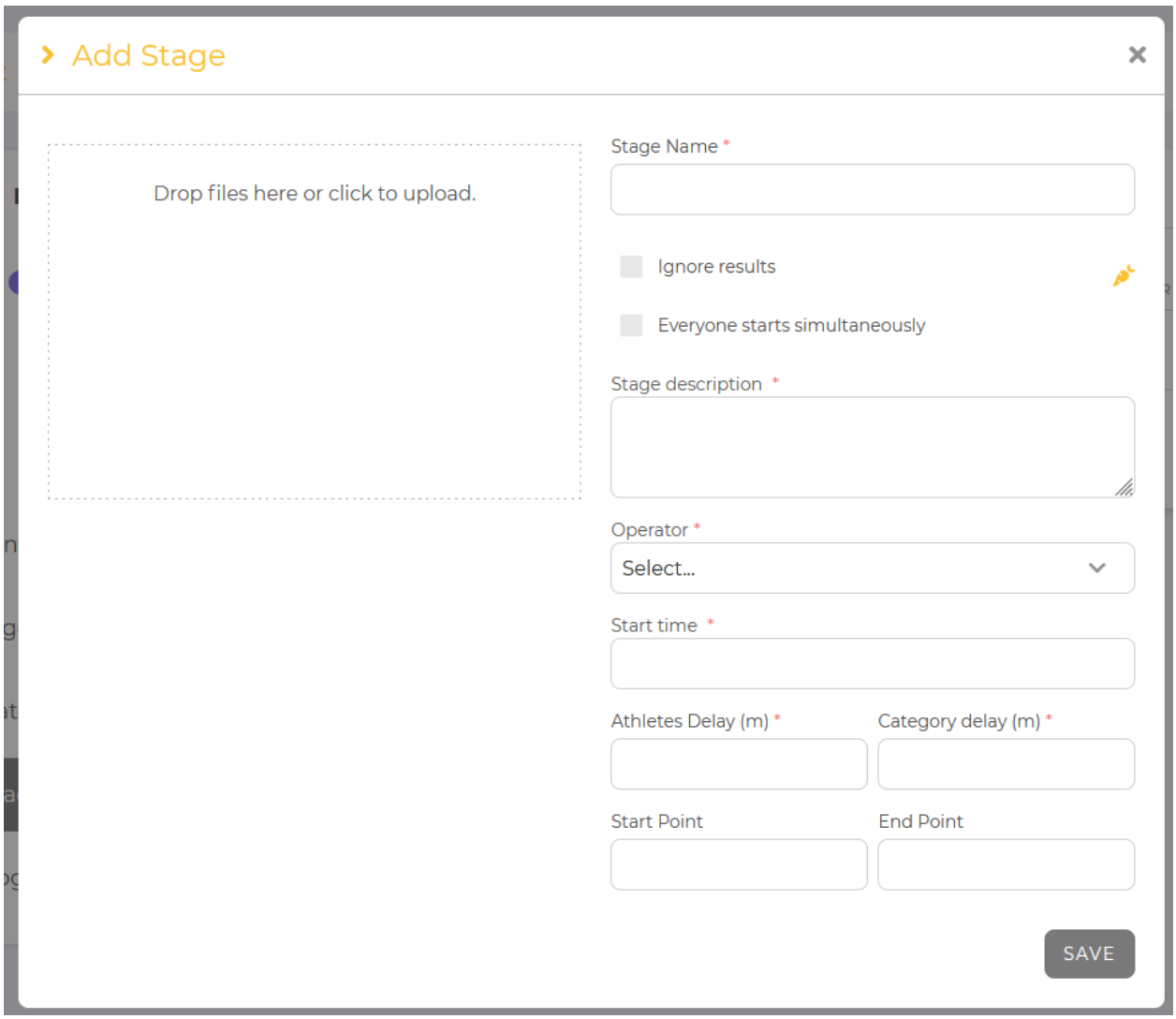

Рисунок 31 — Приклад вигляду модального вікна **Add Stage**

4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ ДЕСКТОП ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ СПОРТИВНИХ ЗМАГАНЬ

4.1 Підвищення надійності відлову атлетів на фініші

Дослідження результатів роботи десктоп-застосунку, спрямованого на організацію та ведення спортивних змагань, виявляється ключовим етапом для оцінки його ефективності та визначення шляхів подальшого вдосконалення. Однією з важливих аспектів розгляду є підвищення надійності відлову атлетів на фініші, що має прямий вплив на точність фіксації результатів та задоволення учасників.

Перш за все, проведене дослідження виявило, що вдосконалення алгоритмів відлову атлетів на фініші привело до значного зростання точності та швидкості реєстрації. Застосунок успішно використовує RFID-технології для ідентифікації учасників та автоматичного реєстрування їх фінішних часів, забезпечуючи мінімальну ймовірність помилок.

Додатково, аналіз результатів показав, що впровадження механізму персистентності для локального зберігання даних дозволяє зберігати інформацію про результати у випадках тимчасового відсутності з'єднання з сервером. Це робить систему більш надійною та готовою до роботи в умовах обмеженого Інтернет-з'єднання, що може бути критичним для подій у віддалених або гірських областях.

Узагальнюючи результати, можна стверджувати, що досліджуваний десктоп-застосунок ефективно впоровується з викликами, пов'язаними з відловом атлетів на фініші, забезпечуючи високий рівень точності та надійності в реєстрації результатів спортивних змагань.

4.2 Найдений оптимальний технічний набір та структура для організації ворот фінішу

Була наступна задача: спроектувати інтегровану систему відлову атлетів, здатну ефективно фіксувати їхні результати, є завданням, що вимагає постійного аналізу та оптимізації. У процесі розробки десктоп-застосунку для організації спортивних змагань виникли технічні виклики, які вимагали творчого та інноваційного підходу до їх вирішення.

Недоліки та шляхи їх вирішення:

Проблема з антенами:

Спочатку система стикалася з недостатністю антен, що впливало на здатність фіксації великих швидкостей атлетів. Зміцнення системи було досягнуто шляхом додавання додаткових антен, забезпечивши більше широкий охоплюючий зону відлову та підвищивши точність ідентифікації.

Спочатку ворота виклядали наступним чином: (див. Рисунок 31)

Рисунок 31 — Перша версія ворот фінішу

Розширення розмірів воріт фінішу:

Необхідність у вдосконаленні безпеки та уникненні можливих травм атлетів виявилася через замалі розміри воріт фінішу. Відповідь на це виклик було знайдено у збільшенні розмірів воріт, застосовуючи ергономічний підхід до їх конструкції. Потім поступово дойшли до такой робочої версії ворот: (див. Рисунок 32)

Рисунок 32 — Остання версія ворот фінішу

Використання 3D-друку для вирішення деталей:

Деталі системи потребували точності та унікальності, щоб забезпечити ефективну роботу. Використання технології 3D-друку стало вирішенням для створення необхідних комплектуючих, забезпечивши їхню точність та ергономічність.

Роздрукували на 3D принтері наступну схему: (див. Рисунок 33)

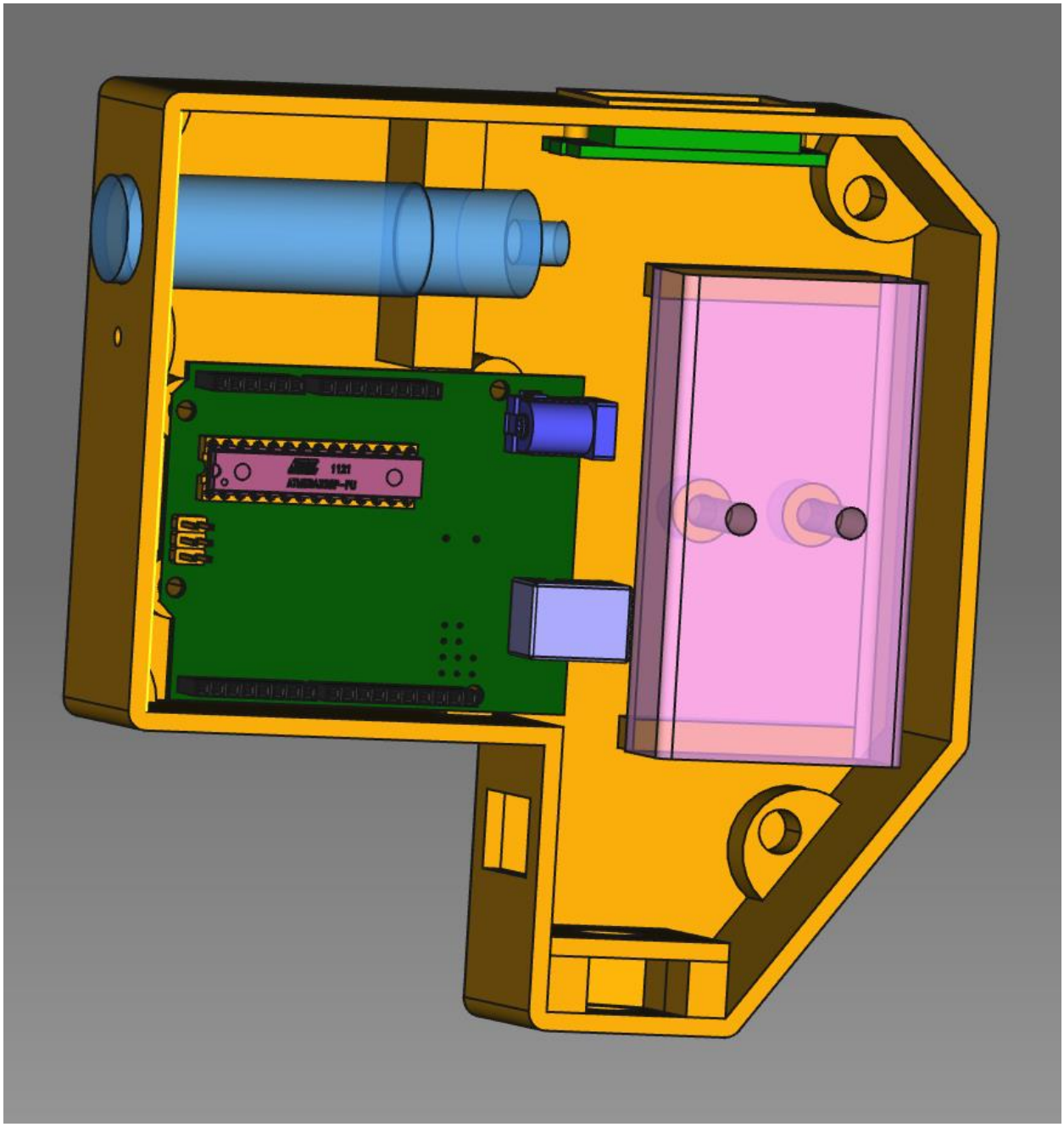

Рисунок 33 — Схема деталі для 3D принтеру

У результаті отримали наступну деталь: (див. Рисунок 34)

Рисунок 34 – Кінцевий вигляд деталі розруковані на 3D принтері

Було розруковано ще пару деталей на принтері: (див. Рисунки 35 та 36)

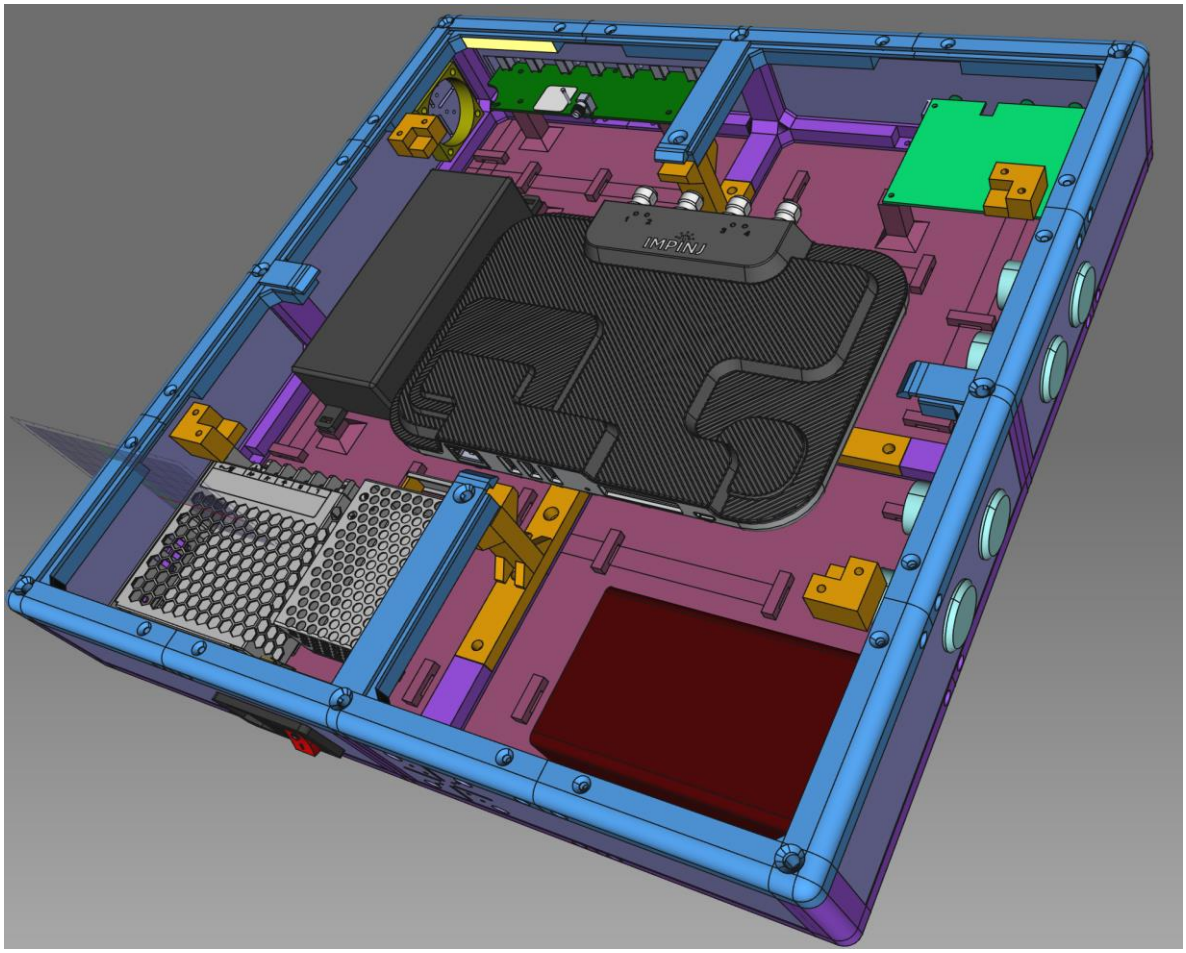

Рисунок 35 – Схема деталі

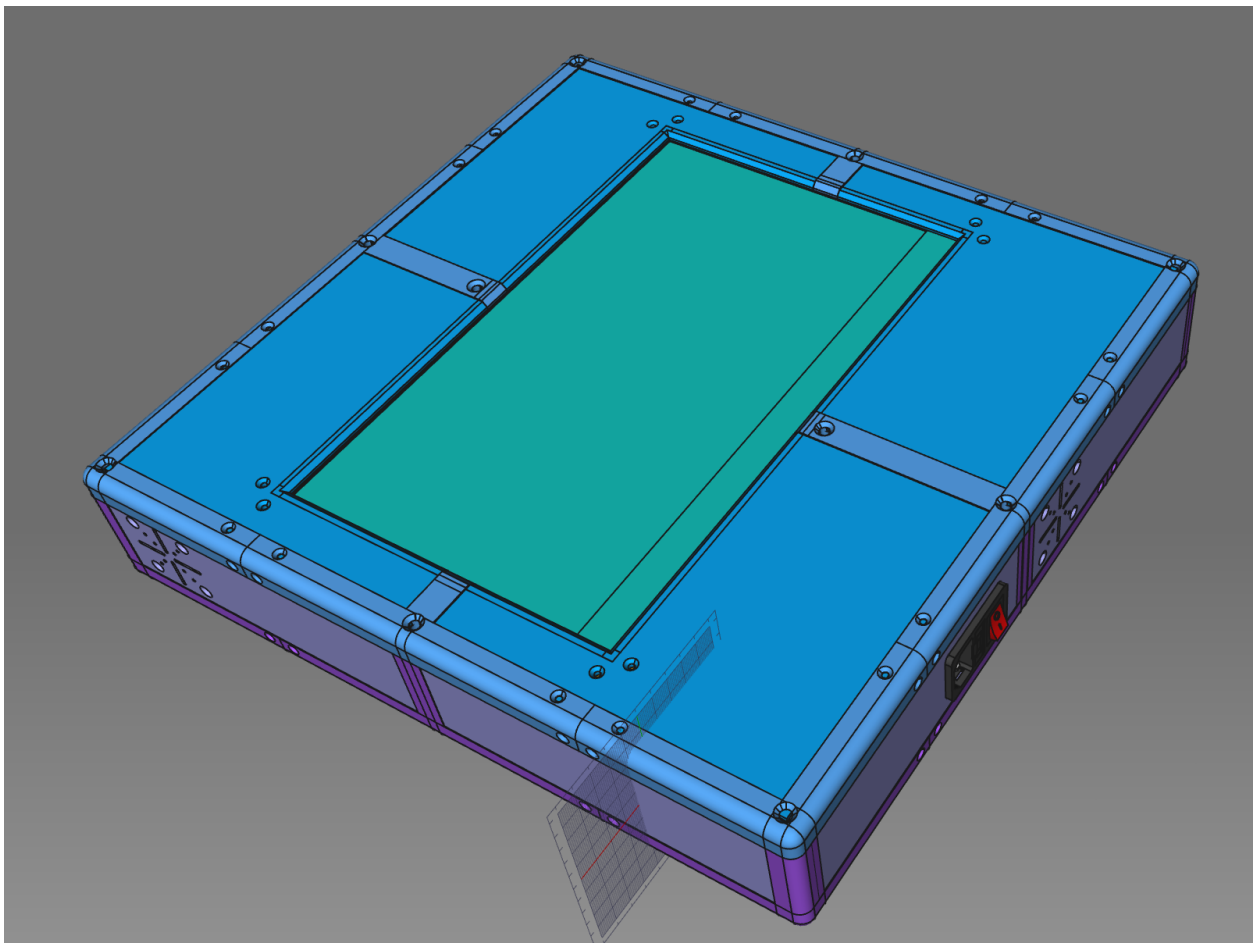

Рисунок 36 – Схема деталі

Еволюція контрольної системи:

Перші етапи використовували звичайний ноутбук, але для поліпшення надійності та мобільності перейшли до Raspberry Pi з екраном. Це дозволило системі працювати незалежно від зовнішнього обладнання та забезпечило надійний моніторинг результатів.

Зазначені технічні вдосконалення вказують на важливість постійного вдосконалення та адаптації технічних рішень у сфері спортивних змагань. Підвищення надійності відлову атлетів на фініші виявилося ключовим удосконаленням, що забезпечує точність та спрощує організацію подій.

Тепер комп’ютер на фініші виглядає таким чином: (див. Рисунок 37)

Рисунок 37 – Raspberry PC на фініші

4.3 Використання оптимальної схеми взаємодії Electron та RFID обладнання

У вас виникла складна ситуація з взаємодією Electron і Telnet для роботи з RFID. Початково ви використовували старі версії, які дозволяли об'єднувати веб-та Node.js-середовища без проблем. Однак після оновлення виявилось, що ця можливість більше не підтримується, і вам довелося знаходити новий спосіб організації спільної роботи.

Мінуси старого підходу:

1. Несумісність версій: Старі версії не забезпечували сумісності між веб-та Node.js-середовищами, що призводило до труднощів у взаємодії і обміні даними між ними.
2. Обмежена гнучкість: Неможливість об'єднати середовища обмежувала гнучкість та швидкість реагування системи на зміни та події.

Плюси нового підходу:

1. Взаємодія через Event Delivery System: Вирішення проблеми полягало в застосуванні Event Delivery System, яка дозволяє зручно та ефективно взаємодіяти між веб-інтерфейсом та Node.js-середовищем. Це рішення дозволяє передавати події та дані між окремими частинами додатку.
2. Покращена гнучкість: Новий підхід забезпечує більшу гнучкість в розробці та підтримці, оскільки дозволяє ефективно об'єднувати різні частини системи, зберігаючи при цьому їхню незалежність.
3. Оптимізована взаємодія: Event Delivery System забезпечує оптимізований обмін даними, що поліпшує ефективність та продуктивність системи.

Event delivery система працею по даній схемі: (див. Рисунок 38)

Рисунок 38 – Event Delivery ситема схема роботи

Event Delivery System (Система доставки подій) є ключовим компонентом у роботі RFID-системи, забезпечуючи ефективне та точне виведення подій для подальшої обробки. Ця система відповідає за передачу та збір інформації про події, що виникають під час використання RFID-ридера, забезпечуючи зв'язок між різними компонентами системи.

Однією з ключових функцій Event Delivery System є забезпечення синхронізації подій між антеннами, сервером та RFID-ридером. Це гарантує, що інформація про проходження кожного номерка фіксується з точністю та вчасно, надаючи повну картину подій під час спортивного заходу.

Крім того, система дозволяє налагоджувати різноманітні фільтри та умови, які визначають, які події повинні бути передані для подальшої обробки. Це дозволяє оптимізувати обробку інформації та забезпечує гнучкість у визначенні та обробці конкретних сценаріїв використання системи.

Важливим елементом Event Delivery System є його висока надійність та можливість масштабування, що робить його ідеальним рішенням для спортивних заходів різного масштабу, де вимоги до точності та ефективності великі.

В результаті змінений підхід до взаємодії з RFID дозволив уникнути обмежень старих версій і створити більш ефективну та гнучку систему.

4.4 Вирішення real-life проблем при розробці десктоп застосунку для фінішу

У процесі розробки десктоп-застосунку для фінішу спортивних змагань, виникла серйозна проблема, яка вимагала творчого та інноваційного підходу для ефективного вирішення.

Проблема раннього виявлення атлетів на фініші:

Однією з ключових проблем було раннє знаходження атлетів в зоні фінішу перед визначеним часом. Це створювало ризик неправильного визначення результатів та порушення логістики подій, оскільки система не повинна реагувати на атлетів, які ще не мають виступати.

Розробка алгоритму для визначення потреби в відлові атлета:

Для вирішення цієї проблеми був розроблений спеціальний алгоритм, який визначає, чи потрібно в даний момент виявляти атлета на фініші. Цей алгоритм враховує розклад подій та дозволяє системі інтелігентно визначати, коли атлет готовий до фінішу, зменшуючи ризик помилкового виявлення.

Оптимізація процесу прошивки та закріплення номерків:

Ще однією важливою проблемою було тривале прошивання номерків безпосередньо на соревнованіях, що призводило до затримок та невдоволення учасників. Розроблено стратегію розділення процесу, дозволяючи прошивати номерки заздалегідь в офісі, а закріплювати їх за атлетом швидко і ефективно перед стартом.

Вирішення цих проблем вимагало комплексного підходу та використання інтелектуальних алгоритмів, що дозволило оптимізувати роботу системи та забезпечити точне та ефективне виявлення атлетів на фініші, забезпечуючи плавний хід спортивних подій.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено глибоке вивчення проблематики організації спортивних змагань та визначено актуальність створення ефективної та доступної системи для їх автоматизації на рівні десктоп-додатку.

Аналізуючи потреби атлетів та організаторів змагань, виявлено, що існуючі рішення часто недоступні через високі витрати або не повністю відповідають їхнім потребам. Таке визначення проблеми стало ключовим моментом у формуванні мети та завдань розробки десктоп-додатку для організації спортивних змагань.

У процесі вибору технологій та платформ для розробки додатку віддано перевагу мові програмування JavaScript, а також використанню інструментів, таких як Electron, React та RFID-технології, що дозволяють створити кросплатформенний та ефективний інструмент для автоматизації спортивних заходів.

З розробленою системою досягнуто значного прогресу в розв'язанні проблеми ефективності та доступності організації спортивних змагань. Створений десктоп-додаток дозволяє проводити змагання ефективно та без зайвих витрат, що робить його важливим інструментом для спортивної спільноти.

Важливим аспектом розробки десктоп-додатку для організації спортивних змагань є його висока адаптивність та користувацька зручність. Враховуючи потреби різних користувачів, від атлетів до організаторів, система надає інтуїтивний інтерфейс, який спрощує взаємодію з програмою та забезпечує ефективну роботу у різних умовах.

Новизною розробленого додатку є його здатність працювати в оффлайн-режимі, що особливо важливо для спортивних заходів, які можуть відбуватися в умовах обмеженої інтернет-доступності. Це забезпечує стабільну та надійну

роботу системи в будь-яких обставинах, що важливо для успішної організації та ведення змагань.

Висновком роботи є те, що розроблений десктоп-додаток відкриває нові можливості для удосконалення процесів організації та ведення спортивних заходів. Забезпечуючи зручність використання та широкий функціонал, система стає необхідним інструментом для всіх учасників спортивної індустрії, сприяючи підвищенню якості та доступності спортивних подій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. About Product. raceresult.com : веб-сайт. URL: https://www.raceresult.com/en-us/contact/_index.php (дата звернення: 07.12.2022).
2. About Challonge. challonge.com: веб-сайт. URL: https://challonge.com/about (дата звернення: 07.12.2022).
3. About IT’s Your Race: веб-сайт. URL: https://www.itsyourrace.com/AboutUs.aspx (дата звернення: 07.12.2022)
4. Getting Started with Redux redux.js.org веб-сайт. URL: https://redux.js.org/introduction/getting-started (дата звернення: 07.12.2022).
5. What Socket.io is. socket.io веб-сайт. URL: https://socket.io/docs/v4/ (дата звернення: 07.12.2022).
6. SockJS.ably.com веб-сайт. URL: https://ably.com/periodic-table-of-realtime/sockjs (дата звернення: 07.12.2022).
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: 07.12.2022).
8. MongoDB documentation веб-сайт. URL: https://www.mongodb.com/docs/atlas (дата звернення: 07.12.2022)
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: 07.12.2022)
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: 07.12.2022)
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: 07.12.2022).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Мельник Максим Олександрович, студент 2 курсу, денної форми навчання,

Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz118bd-11@stu.zsea.edu.ua,

- підтверджую, що написана мною кваліфікаційна робота на тему: «Особливості побудови десктоп-додатку для організації 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 _____ Мельник Максим Олександрович
(студент)

Дата _____ Підпис _____Скрипник Ірина Анатоліївна
(науковий керівник)