МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕНН

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Автоматизована система моніторингу і керування датчиками в комп'ютерній системі 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)
Р.О. Лазарєв
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н. І.А.Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ Дісітел
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

___Лазарєву Роману Олександровичу_____
(прізвище, ім'я, по батькові)

1.Тема роботи __Автоматизована система моніторингу і керування датчиками в комп'ютерній системі організації спортивних змагань__

керівник роботи ___Скрипник Ірина Анатоліївна, доцент, к.ф.-м.н.
(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 засоби проектування системи старту спортивного змагання;
- 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)
16 слайдів презентації

6. Консультанти розділів магістерської роботи

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

7. Дата видачі завдання 01.09.2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області шляхом комунікація із атлетами для виявлення методів проведення спортивних змагань	02.09-12.09.23	виконано
2	Формулювання та узгодження із керівником тематики роботи	13.09-14.09.23	виконано
3	Вивчення аналогів, існуючих на ринку рішень	15.09-21.09.23	виконано
4	Визначення вимог до майбутньої інформаційної системи	22.09-25.09.23	виконано
5	Проектування апаратної частини системи старту	26.09-29.09.23	виконано
6	Проектування інтерфейсу програмного застосунку	30.09-05.09.23	виконано
7	Створення початкового застосунку та його зв’язування із сервером	06.10-10.10.23	виконано
8	Реалізація спроектованого інтерфейсу	11.10-21.10.23	виконано
9	Проектування API для роботи з апаратною частиною	22.10-27.10.23	виконано
10	Розробка спроектованих алгоритмів	28.10-10.11.23	виконано
11	Тестування апаратної частини та її оптимізація	11.11-18.11.23	виконано
12	Тестування інформаційної системи та виправлення недоліків	19.11-21.11.23	виконано
13	Оформлення звіту	22.11-27.11. 23	виконано

Студент _____ Р.О. Лазарєв
(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: 83

Рисунків: 19

Джерел: 24

Лазарєв Р. О. Автоматизована система моніторингу і керування датчиками в комп’ютерній системі організації спортивних змагань : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І. А. Скрипник. Запоріжжя : ЗНУ, 2023. 83 с.

Мета роботи полягає у вивченні програмних та апаратних рішень, які використовують для проведення старту спортивних змагань, їхніх альтернатив та реалізації інформаційної системи для автоматизованого проведення старту стейджа в спортивних змаганнях.

Предметом дослідження є необхідність створення для інформаційної системи для керування датчиків в контексті проведення старту спортивних змагань.

У процесі дослідження була розглянута проблема створення апаратно-програмного комплексу для керування датчиками у контексті старту спортивних змагань з динамічною чергою атлетів.

Ключові слова: Arduino, програма, програмний застосунок, React Native, черга, API, бухгалтерія, інтерфейс.

SUMMARY

Pages: 83

Figures: 19

Source: 24

Lazariev R. O. Automated system of monitoring and control of sensors in the computer system of organizing sports competitions: master's qualification thesis of specialty 121 "Software engineering" / science. manager I. A. Skrypnyk. Zaporizhzhia: ZNU, 2023. 83 p.

The purpose of the work is to study the software and hardware solutions used for the start of sports competitions, their alternatives, and the implementation of an information system for the automated start of the stage in sports competitions.

The subject of the study is the need to create an information system for managing sensors in the context of the start of sports competitions.

In the process of research, the problem of creating a hardware and software complex for controlling sensors in the context of the start of sports competitions with a dynamic queue of athletes was considered.

Keywords: Arduino, program, software application, React Native, queue, API, accounting, interface.

ЗМІСТ

ВСТУП

Актуальність теми

Спортивні змагання стали несуперечливою частиною світової культури, об'єднуючи велику кількість людей і привертаючи увагу з усіх куточків світу.

Організація таких подій вимагає ретельного підходу до багатьох аспектів, розпочинаючи від вибору місця і закінчуючи системою обліку результатів. З інтенсивним розвитком комп'ютерних технологій багато організаторів спортивних подій все частіше приділяють увагу сучасним цифровим рішенням, які охоплюють практично всі аспекти проведення змагань або турніру: від реєстрації учасників та проведення заходу до автоматизованого підрахунку результатів та визначення призових місць. Ці інноваційні системи здатні автоматично фіксувати результати, надавати детальну статистику та спрощувати завдання суддів та організаторів. Вони взагалі роблять спортивні змагання більш привабливими для глядачів, оскільки забезпечують надійне та зручне відображення результатів у реальному часі. Такий підхід не лише збагачує враження від події, але й унеможливлює будь-яке нудьгування серед шанувальників спорту.

Багато організаторів, що ведуть змагання на місцевому рівні, використовують традиційні методи вимірювання результатів, такі як звичайні секундоміри та людина-оператор, яка фіксує момент перетину атлетом стартової та фінішної лінії. Цей підхід не лише піддається неточностям, але й може спричинити серйозні похибки через вплив людського фактору.

На жаль, використання таких застарілих методів може призводити до не достовірних результатів та втрати об'єктивності у визначенні переможців. Людська суб'єктивність та можливі помилки при вимірюванні часу можуть негативно впливати на інтегрітет та справедливість змагань. У світлі цього стає надзвичайно важливим переходити до сучасних технологічних рішень для обліку результатів. Використання електронних систем таймінгу та автоматизованих засобів може гарантувати точність та об'єктивність у визначенні часу

атлетів. Такі технології не лише зменшують ризик помилок, а й прискорюють весь процес, забезпечуючи точні та надійні результати, що є важливим для збереження іміджу та авторитету будь-якого спортивного заходу.

Однак існують кілька проблем, які перешкоджають впровадженню автоматизованих рішень для організації змагань. По-перше, це висока вартість обладнання, оскільки навіть оренда таких систем може обійтися організаторам у кілька тисяч доларів. Ця фінансова обтяжливість стає головною перепоною для багатьох, хто прагне вдосконалити свої спортивні заходи. Крім того, іншою проблемою є те, що деякі організатори змагань, зокрема у велоспорті, де траси розташовані у важкодоступних місцях, можуть вважати автоматизовані системи нерентабельними в таких умовах. Такі види спорту, наприклад, гірські велосипеди, можуть потребувати індивідуальних підходів, оскільки стандартні технічні рішення можуть бути важко адаптовані до екстремальних умов. Такі обставини ускладнюють вибір для організаторів, які можуть коливатися між впровадженням нових технологій та збереженням традиційних, менш витратних методів.

Таким чином, актуальним є створення апаратно-програмного рішення для автоматизації проведення старту стейджів на спортивних змаганнях системою.

Мета роботи

Мета роботи полягає у вивченні програмних та апаратних рішень, які використовують для проведення старту спортивних змагань, їхніх альтернатив та реалізації інформаційної системи для автоматизованого проведення старту стейджа в спортивних змаганнях.

Об’єкт дослідження

Об’єктом дослідження є процес проведення старту стейджа в спортивних змаганнях.

Предмет дослідження

Предметом дослідження є необхідність створення для інформаційної системи для керування датчиків в контексті проведення старту спортивних змагань.

Методи дослідження

Для вирішення поставленої задачі використовуються наступні методи дослідження:

1. Аналіз особливостей та існуючих рішень (аналогів) для проведення старту на спортивних змаганнях.
2. Аналіз технологій для реалізації інформаційної системи для створення автоматизованої системи проведення старту.

Наукова новизна одержаних результатів

Наукова новизна одержаних результатів дослідження полягає у тому, що емпіричним шляхом було створено декілька версій апаратного рішення для системи проведення змагань, котра може працювати в несприятливих умовах. Також була вирішена задача розрахунку часу виступу атлету в динамічній системі.

Практичне значення одержаних результатів

Практичне значення одержаних результатів дослідження полягає у тому, що дане рішення можливо використовувати для проведення спортивних змагань з різних дисциплін та в різних умова. Завдяки зрозумілому інтерфейсу та невеликого розміру обладнання ця система є зручним у використанні в важкодоступних місцях.

Апробація результатів

Результати роботи, викладені у кваліфікаційній роботі магістра, були опубліковані в матеріалах XVI Університетській науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених «Молода наука-

2023» та III Всеукраїнській науково-практичній конференції за участю молодих науковців «Актуальні питання сталого науково-технічного та соціально-економічного розвитку регіонів України» .

Глосарій

Android — це операційна система для мобільних пристроїв, таких як смартфони та планшети. Розроблена компанією Google, Android використовується на широкому спектрі пристроїв від різних виробників.

JavaScript — це високорівнева скриптова мова програмування, яка використовується в таких сферах програмування, як веб-програмування. розробка десктопних та мобільних застосунків.

React Native — JavaScript фреймворк розроблений компанією Facebook для створення кросплатформенних застосунків для платформ Android, iOS, tvOS, Web.

Arduino — це електронна платформа для прототипування з відкритим вихідним кодом, яка надає користувачам можливість створювати інтерактивні електронні об'єкти.

Bluetooth Low Energy — бездротовий протокол передачі даних між пристроями з обмеженим джерелом енергії за допомогою технології Bluetooth.

Фреймворк — структура або платформа з відповідною архітектурою, яка надає базові компоненти та інструменти для прискорення процесу розробки програмного забезпечення.

Хук — в React та React Native є функцією, яка дозволяє використовувати стан та інші можливості у функціональних компонентах.

Скрипт — інтерпретований програмний код, який виконує конкретну задачу або набір задач.

API — це набір інструментів чи функцій, які дозволяють різним програмам взаємодіяти між собою.

Стейдж — це позначення для окремого етапу в контексті спортивних змагань, котрі можуть визначатися за типом траси або маршрутом.

IDE — Це програмне забезпечення, яке надає розробникам інтегрований комплекс рішень та інструментів для зручного написання, редагування, відлагодження та компіляції програмного коду та сборки проекту.

Інтерфейс — це зовнішній вигляд програми або системи, що надає користувачеві можливість взаємодіяти з ними.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМ ВИКОРИСТАННЯ РІЗНИХ СИСТЕМ АВТОМАТИЗАЦІЇ ПРОВЕДЕННЯ

1.1 Огляд проведених досліджень та наукових джерел

Задача автоматизації спортивних змагань та обчислення результатів досі залишається актуальною, і для різних видів спорту використовуються специфічні рішення. Цей підхід до удосконалення процесів змагань знаходить широке застосування та адаптується відповідно до особливостей кожної галузі.

Автор у даній статті [20] описує використання системи захоплення руху на основі GPS та LPS (Local Positioning System). На екіпіруванні спортсменів закріплювалися трекери, які автоматично накопичували дані під час їхніх забігів і надавали інформацію про швидкість на конкретних ділянках маршруту. Цей метод забезпечує значний обсяг інформації, але через різноманітні фактори, такі як погодні умови та наявність будівель поруч, а також матеріали, наприклад метал, і обмежений частотний діапазон роботи трекерів, можуть виникати помилки. Наприклад, датчики, що працюють на частоті менше 20 Гц, можуть вказувати похибку до 20%, що є неприпустимим для проведення змагань, де різниця в результаті між атлетами може бути дуже мала, в десятках частин секунди. Ця система краще підходить для збору та аналізу даних впливу типу маршруту чи рельєфу на швидкість атлета, аніж для точного вимірювання результатів в умовах живих змагань.

Автор у даній статті описує метод вимірювання результатів є використання за допомогою камер лінійного сканування (Line-Scan Cameras). Дані камери можуть зафіксувати зі швидкістю від 1000 до 10 000 кадрів в секунду і додають мітку часу до кожного кадру [24].

Це дозволяє з точністю визначати момент, коли спортсмен перетинає фінішну лінію, і вирішувати ситуації, коли два учасники візуально фінішують одночасно. Камери лінійного сканування вважаються найбільш точною автоматичною системою таймінгу, але їхнє застосування обмежується прямолінійними трасами.

Також часто використовується пасивна технологія RFID як засіб реєстрації результатів учасників в бігових змаганнях. У даній статті [6] автор поділяє свій досвід створення системи спортивного таймінгу на основі пасивних RFID-антен, яка продемонструвала надійні результати під час тестування. Крім того, у статті докладно описано саме рішення, яке вирішує проблему розташування RFID-міток для атлетів різних видів спорту.

Наразі однією з найрозповсюдженіших систем старту є променева система вимірювання часу (Beam Timing System). Ця система складається з датчика, який випромінює світловий промінь, і фіксує час, коли атлет перетинає цей промінь. Існують два види систем цього типу: система з одним променем (Single Beam Timing System) і система з двома променями для вимірювання часу (Dual Beam Timing System). Відмінність між цими системами полягає в тому, що дволучева система використовує два сенсори одночасно, і для реєстрації результату обидва промені повинні бути перетнуті одночасно. Ця система внесла значну варіативність, оскільки, в порівнянні з однополучевою системою, вимагає, щоб атлети стартували з одного положення через вплив зміни висоти атлета на результат вимірювання [21].

Також автор в даній статті [18] висвітлює відмінність у вимірах результатів. Крім того, у цій статті автор вказує на різницю між двома воротами, які використовують різні системи. Порівняно з дволучевою системою, однолучева система показує похибку в результатах до 0,02 с у 64% випадків і до 0,05 с у 27% випадків.

1.2 Огляд систем для проведення змагань

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

Race Result — це комплексне апаратно-програмне рішення, яке забезпечує проведення та управління спортивними подіями, а також автоматизує підрахунок результатів змагань. Складова програмна частина включає веб-сайт, на якому доступний функціонал онлайн-реєстрації учасників на подію, управління списками учасників та редагування їхньої інформації. На дашборді події сайту виводиться комплексна інформація, необхідна для демонстрації результатів та їхнього аналізу. Race Result служить інструментом, що спрощує та автоматизує багато аспектів організації та проведення спортивних подій, забезпечуючи точність та доступність результатів для учасників та глядачів, а також сприяючи ефективній роботі організаторів.

Апаратна частина системи, розроблена компанією, представляє собою систему, що включає в себе декодер сигналу, пасивну UHF антену, яка отримує сигнали від міток на відстані до 4 метрів. Також можна використовувати активний модуль замість пасивної антени, який зчитує мітку з транспондера, розташованого неподалік, зі швидкістю до 250 км/год, при цьому точність вимірювання становить 0.004 сек. Це рішення добре адаптоване до різних видів спорту, від швидкісного байкінгу до плавання, що робить його високоефективним [14]. Основним недоліком цієї системи є висока вартість. За використання декодера сигналів, який зчитує сигнали та розраховує час приїзду, компанія встановлює вартість у 4200 євро. Також окрема цінова категорія визначена для антен та транспондерів, що призводить до того, що потенційні замовники в основному обмежуються організаторами великих спортивних заходів. Також недоліком є те, що декодер не надає зручного та інтуїтивного інтерфейсу для свого користування (див. Рис 1.1).

Рисунок 1.1 — інтерфейс декодеру сигналу

IT’s your race — це інтегрована система, розроблена для організації та управління спортивними змаганнями в реальному часі в різних видів спорту. Ця система складається з двох основних компонентів — веб-сайту та мобільного додатку. Веб-сайт надає зручний інтерфейс для користувача, де можна керувати всіма аспектами змагань, такими як створення, відкриття реєстрації та планування етапів. Крім того, передбачена інтеграція з соціальними мережами, що дозволяє поділитися інформацією про подію та результатами учасників у відповідному змаганні.

Мобільний додаток пропонує функції особистого кабінету та трекера (див. Рис 1.2). Під час участі в змаганні він використовує GPS-трекінг для відстеження вашого прогресу та шляху.

Проте важливими недоліками є те, що мобільний додаток не є доступним у всіх регіонах світу, що обмежує проведення змагань в деяких країнах. Також зауважується, що GPS-датчики не є найбільш ефективним способом отримання точних результатів забігу, як було (зазначено в п.1.1).

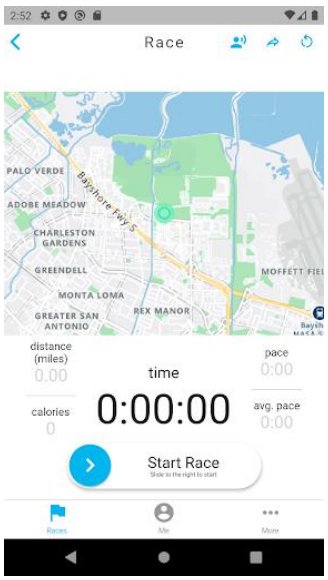

Рисунок 1.2 — Інтерфейс мобільного додатку IT’s your race

ChronoTrack — це апаратно-програмне рішення, яке створене для автоматизації проведення змагань однойменної компанії ChronoTrack, яка спеціалізується на розробці технологій для проведення спортивних подій [3]. Однією з ключових областей їхньої діяльності є системи таймінгу, які використовують різноманітні технології, такі як радіочастотний ідентифікатор (RFID), для точного вимірювання часу під час змагань. Це дозволяє отримати надійні та точні результати, особливо в галузі бігу, триатлону та велоспорту.

Крім того, ChronoTrack пропонує системи реєстрації для зручної участі учасників у спортивних подіях. Їхні рішення охоплюють онлайн-реєстрацію, інтеграцію з різними платформами та забезпечення організаторів необхідною інформацією про учасників.

Також дана компанія має своє програмне рішення, котре відслідковує, які атлети знаходяться в зоні дії RFID завдяки ручному натисканню зняти час старту з відповідного атлета (див. Рис 1.3). Також додаток дозволяє використовувати різні Ntp сервера, для більш детального заміру часу.

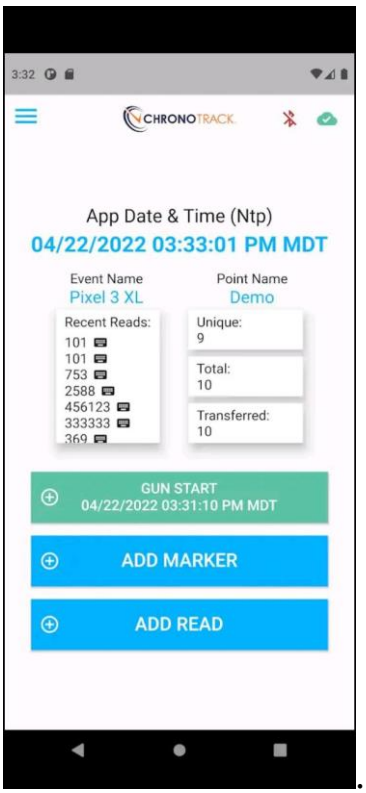

Рисунок 1.3 — Інтерфейс мобільного додатку ChronoTrack

Організація та управління спортивними заходами є іншим важливим компонентом діяльності ChronoTrack. Їхні технології допомагають в створенні розкладів, налаштуванні маршрутів та визначенні параметрів змагань. Компанія також працює над створенням мобільних додатків та інтеграцій для забезпечення зручності для учасників і глядачів під час спортивних подій.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ АПАРАТНИХ ТА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ СИСТЕМИ

2.1 Обрання платформи для реалізації застосунку

Першим етапом у створенні програмного застосунку було визначення платформи та операційної системи, на яких буде розроблятися продукт. Оскільки спортивні змагання зазвичай проводяться на різних природних ландшафтах, таких як ліси, гори та рівнини, було вирішено використовувати мобільні пристрої як основну платформу для програми. Це обумовлено тим, що це полегшує розгортання системи на старті змагань. Для мобільних пристроїв існують дві основні операційні системи, на яких розробляються застосунки: iOS та Android. Отже, враховуючи вимоги та особливості проведення спортивних подій, обрано мобільні пристрої як платформу, а операційні системи iOS та Android — як цільові для розробки програми.

Android — це операційна система для мобільних пристроїв, яка розробляється компанією Google. Вона є відкритою, що дозволяє розробникам вносити зміни та адаптувати її за своїми потребами, і базується на ядрі Linux, що забезпечує стабільність та безпеку. Однією з важливих характеристик Android є його відкритість до суспільства розробників та гнучкість, що дозволяє розробникам гнучко налаштовувати операційну систему відповідно до своїх власних вимог та потреб.

iOS — це операційна система, яку розробила та випускає компанія Apple спеціально для своїх мобільних пристроїв, таких як iPhone та iPad. Ця операційна система глибоко інтегрована в екосистему Apple та визначається як закрита платформа, що повністю контролюється компанією. У контексті використання iOS, важливо відзначити його спрямованість на максимальну сумісність зі специфікаціями та функціоналом пристроїв Apple. Це дозволяє користувачам отримувати максимальний комфорт в користуванні та взаємодії з різними пристроями в межах екосистеми. Закритий характер системи вказує на

те, що Apple має повний контроль над розробкою, удосконаленням та дистрибуцією iOS, що, з одного боку, гарантує високий рівень оптимізації та безпеки, а з іншого — може обмежувати свободу для змін та адаптації з боку сторонніх розробників.

Завдяки відкритій архітектурі Android, ця операційна система є сумісною з більшістю мобільних пристроїв у світі. Важливо також відзначити, що Android дозволяє розробникам створювати та тестувати збірки своїх додатків без обов'язкової публікації та оплати за неї. Це надає можливість розробникам ефективно тестувати та вдосконалювати свої продукти перед релізом. Оскільки операційна система Android надає розробникам більш гнучкі налаштування свого середовища, вибір саме її для застосунку видається логічним. Ця свобода сприяє створенню інноваційних та ефективних додатків, а також спрощує процес розробки та тестування, що важливо для досягнення високої якості та задоволення користувачів.

2.2 Огляд методів створення Android застосунків

Після того, як була обрана Android в якості ОС, треба розглянути основні способи розробки застосунку. Для цій платформі є два основних способи розробки, це нативна розробка або з використанням фреймворків.

Розробка нативних додатків для платформи Android передбачає використання Android SDK та мови програмування Java або Kotlin. Один із ключових плюсів цього підходу полягає в тому, що використання нативного Android SDK дозволяє повноцінно використовувати всі API, які надає операційна система, забезпечуючи при цьому максимальну швидкість, на відміну від використання фреймворків [7]. Однак високий поріг входження в процес розробки при цьому методі роблять його відповідним, переважно, для складних додатків, які включають в себе велику кількість операцій, що потребують значних обчислень, 3D графіку або високого рівня уваги до забезпечення безпеки. Такий підхід до розробки особливо ефективний для вимогливих до ресурсів та

високопродуктивних додатків, які часто мають специфічні вимоги та високі стандарти щодо швидкості та ефективності. Тим не менш, слід враховувати, що цей метод може вимагати більше часу і витрат, особливо на етапі початкового розвитку.

Більш простим методом створення застосунків є розробка через фреймворки. На даний час є два основних фреймворки, які використовують на практиці є React Native та Flutter.

React Native є відкритим та кросплатформенним фреймворком для створення мобільних додатків, який розроблений компанією Facebook. Особливість цього фреймворка полягає в тому, що він базується на бібліотеці React та побудований на основі NodeJS, що забезпечує зручний розвиток додатків за допомогою мови програмування JavaScript [16]. Крім того, React Native дозволяє використовувати широкий спектр пакетів та сторонніх бібліотек з репозиторію npm (node package manager) для NodeJS.

Однією з важливих особливостей React Native є його здатність створювати модулі на рівні нативного коду, які можна використовувати у JavaScript-коді через механізм Promise. Це особливо корисно, оскільки деякі функціонал можливо реалізувати безпосередньо на рівні нативного коду, і React Native дозволяє ефективно поєднувати цей підхід із зручністю та гнучкістю розробки на мові JavaScript. Таким чином, React Native стає потужним інструментом для швидкого та ефективного розгортання кросплатформенних мобільних додатків.

Flutter — це фреймворк для створення мобільних застосунків, який розробила компанія Google. Його висока ефективність особливо проявляється при створенні додатків зі складним та вдосконаленим інтерфейсом [8]. Основою Flutter є мова програмування Dart, що дозволяє розробникам працювати над додатками для обох платформ — iOS та Android — одночасно.

В порівнянні з React Native, важливо відзначити, що Flutter має меншу підтримку від спільноти, що може призвести до обмеженої наявності готових бібліотек та плагінів для використання в роботі. Незважаючи на це, заслуговує

уваги його популярність завдяки ефективності та можливості надавати високоякісні візуальні результати, що робить його значущим інструментом для розробників, особливо при створенні додатків зі складним та нетиповим інтерфейсом [13].

На сьогоднішній день мобільні пристрої збільшують свої обчислювальні можливості, і якщо ваша программа не є грою зі складною 3D графікою, то рекомендовано віддати перевагу використанню фреймворків у процесі розробки. Якщо йдеться про вибір фреймворку для розробки програми, то варто врахувати, що якщо потрібно створити програму з простим шаблонним інтерфейсом, не завантажену важкою анімацією, розумним вибором буде React Native. Це обумовлено тим, що розробка UI для Flutter може зайняти значно більше часу, і цей фреймворк наразі не надає належного API для взаємодії з системними функціями телефонів, за винятком підтримки Bluetooth та NFC.

В результаті було прийнято рішення використовувати фреймворк React Native для реалізації програмної частини проекту. Цей вибір зумовлено не лише простотою використання, але й наявністю значної кількості бібліотек, які можна легко використовувати в ході розробки. Це сприяє швидкій і ефективній реалізації функціональності, а також надає можливість ефективно використовувати готові рішення для різноманітних завдань у процесі створення програмного продукту.

2.3 Огляд середовищ розробки для створення Android застосунків

Після визначення платформи та фреймворку для розробки нашого додатку, наступним етапом є вибір редакторів та середовищ розробки, які найбільш оптимально підходять для даного стеку технологій. Основним середовищем розробки для Android додатків є рішення від компанії JetBrains — Android Studio.

Ця інтегрована середовища розробки спеціально адаптована для створення Android додатків і володіє надтою функціональністю та зручним інтерфейсом, що сприяє ефективній та швидкій роботі над проектами. Вона забезпечує високий рівень якості та продуктивності у процесі розробки мобільних додатків. Однією з ключових можливостей є інтеграція із емуляторами пристроїв, яка дозволяє розробникам тестувати функціонал та інтерфейс на різних версіях операційної системи та екранах без наявності фізичних пристроїв з різними форм-факторами. Дана IDE включає графічний редактор, що дозволяє легко створювати екрани програм за допомогою стандартних графічних компонентів Android SDK. Ці функції роблять його необхідним інструментом для тих, хто прагне оптимізувати робочий процес при створенні високоякісних та ефективних мобільних додатків [12].

Для зручності, важливо відділити вибір редактора для написання JavaScript коду. Android Studio не забезпечує належної підтримки скриптових мов, таких як JavaScript, і це може призвести до уповільнення розробки програми.

Отже, для оптимального продовження розробки, рекомендується обрати редактор, який забезпечує ефективну роботу з JavaScript кодом, забезпечуючи при цьому швидкість та зручність у процесі програмування.

Популярними редакторами для скриптового коду є Sublime Text та Visual Studio Code.

Sublime Text — це популярний текстовий редактор серед розробників, славиться своєю простотою використання та високим рівнем функціональності.

Цей редактор дозволяє ефективно редагувати окремі файли, а також зручно оглядати структуру цілого проекту, надаючи можливості, що схожі з можливостями інтегрованих середовищ розробки [19], наприклад, Visual Studio.

Завдяки підтримці синтаксису різних мов програмування та можливості встановлення різноманітних плагінів для розширення функціоналу, Sublime Text стає універсальним інструментом, який можна використовувати не тільки для зручного написання скриптів, та й для розробки проектів на різних компільованих мовах програмування.

Visual Studio Code — це безкоштовний та легкий текстовий редактор, розроблений компанією Microsoft, який став популярним інструментом серед розробників.

Принципова відмінність даного редактора полягає в інтеграції з Git на рівні ядра, що забезпечує природнішу взаємодію з системою контролю версій.

Крім того, надається зручніший репозиторій плагінів у вигляді структурованого списку, де кожен плагін супроводжується описом своєї функціональності [23]. Це надає користувачеві інформацію про те, який ефект очікувати від встановлення кожного конкретного плагіна.

На відміну від Sublime Text, де для вибору плагіна необхідно вводити текстову команду для встановлення плагіна, і тобі потрібно знати, як цей плагін називається.

Додатковим плюсом є те, що компанії, що підтримують мови програмування або фреймворки, розробляють власні офіційні плагіни для VS Code [22]. Це, в більшості випадків, забезпечує вичерпне покриття основних потреб у взаємодії з проектами, написаними на конкретній мові програмування.

Підбиваючи підсумки, раціональним вибором буде використання Android Studio для розгортання та редагування нативної структури проекту. Водночас, Visual Studio Code може бути використаний для комфортного написання JavaScript коду з використанням плагінів React Native та React Native Debug Tools для налагодження додатків.

2.4 Огляд можливостей Arduino в контексті апаратного рішення для інформаційної системи

Оскільки дана інформаційна система у реалізації представляє собою програмно-апаратний комплекс, необхідно використовувати керуючий блок, який може ефективно обробляти сигнали з обох джерел (від датчиків та додатків).

Загальнодоступним та популярним вибором для цього є програмовані плати з родини Arduino.

Це надійне та доступне рішення, яке може успішно взаємодіяти з різними аспектами системи, забезпечуючи необхідний контроль та обробку інформації.

Arduino — це апаратно-програмне рішення з відкритим вихідним кодом, спеціалізоване для створення самостійних систем та легкої інтеграції мікроконтролера у вже існуючі системи за допомогою USB або Bluetooth з'єднань. Arduino знаходить застосування в розробці рішень для Інтернету речей та систем розумного будинку, завдяки можливості підключення та обробки сигналів від різноманітних сенсорів та датчиків.

Скрипти для Arduino розробляються мовами програмування C/C++ з деякими особливостями, оскільки в API Arduino використовуються функції, які можуть відрізнятися від стандартних пакетів C++. Наприклад, такі функції, як Serial.readString() і Serial.println(), дозволяють зчитувати або записувати рядки в серійний порт, що є необхідним для взаємодії з іншими пристроями.

Arduino також підтримує розробку власних бібліотек, що спрощує процес написання коду.

Можливість створення власних модулів та функцій надає розробникам можливість використовувати їх у різноманітних проектах, що сприяє повторному використанню коду та забезпечує більш ефективний процес розробки програмних рішень для Arduino.

За допомогою цієї можливості спільнота створила значну кількість бібліотек, які реалізують різноманітний функціонал і мають зручний API для використання. Для своїх плат, компанія Arduino також створила власне інтегроване середовище розробки (IDE), що дозволяє повністю проектувати, компілювати та завантажувати скрипти на плату Arduino. У самому IDE також вбудований зручний монітор серійного порту, який, виводячи рядки зі з'єднаного порту, дозволяє тестувати та налагоджувати скрипти, завантажені на плату [9].

2.5 Огляд бібліотек для створення мобільного застосунку на React Native

Оскільки програма буде розроблятися на фреймворку React Native, можна користуватися готовими бібліотеками та рішеннями, створеними розробницькою спільнотою, для вирішення необхідних задач.

2.5.1 Огляд бібліотек для побудови інтерфейсу

React Native включає в себе вбудований набір нативних елементів, таких як кнопки та текстові поля, які можна змінювати за допомогою стилів. Однак кастомізація та розробка кожного елемента інтерфейсу з нуля може стати неефективною з точки зору часу. Для вирішення цієї проблеми використання React Native Paper є оптимальним варіантом.

Це кроссплатформенний набір готових елементів інтерфейсу, спроектований для створення мінімалістичного дизайну, який за замовчуванням відповідає рекомендаціям Google Material Design.

В ньому вже включено повний набір готових шаблонних інтерфейсів, що охоплюють все від кнопок та діалогових вікон до індикаторів завантаження.

Також є можливість створення своїх користувацьких тем, для підтримки, наприклад як темної так і світлої теми інтерфейсу (за бажанням користувача).

2.5.2 Огляд бібліотек для створення навігації

У фреймворку відсутній вбудований функціонал переміщення користувача між різними екранами. Для вирішення цієї задачі існують дві основні опції: використання бібліотеки React Navigation або бібліотеки React Native Navigation.

React Navigation — це набір інструментів для навігації та управління стеками екранів в React Native, спрямований на створення гнучких та динамічних інтерфейсів для мобільних додатків. Бібліотека надає можливість конструювати різноманітні навігаційні контейнери, такі як TabNavigator. При використанні TabNavigator, навігація до різних екранів відбувається за допомогою панелі, на якій розміщені навігаційні кнопки для кожного екрана. Переходи між екранами відбуваються при натисканні на відповідні кнопки, що забезпечує зручний та інтуїтивно зрозумілий спосіб переміщення в додатку.

Іншою важливою характеристикою є обробка подій, яка відбувається, коли екран є активним (onFocus) або неактивним (onBlur), а також можливість прикріплювати власну логіку до цих подій. Навігація в даній бібліотеці будується за допомогою компонентів навігації, котрі наглядно надають тебе уявлення, як побудована навігація в застосутку.

React Native Navigation — це бібліотека для навігації в React Native, розроблена командою Wix. Вона пропонує потужні інструменти для створення ефективних та швидких інтерфейсів для мобільних додатків на платформі React Native. Одна з ключових особливостей бібліотеки — це використання нативних компонентів для навігації, що призводить до високої продуктивності та природного відчуття для кінцевого користувача. Проте важливим недоліком є те, що процес створення навігації відбувається на етапі відкриття, і вся карта навігації формується у вигляді одного об'єкта JSON. Це дуже ускладнює проектування самої структури навігації.

З урахуванням усіх переваг та недоліків, буде використана бібліотека React Navigation, за більш простий і ефективний метод проектування самої навігації.

2.5.3 Огляд бібліотек для зберігання та відновлення даних

Для нормальної роботи, програма має зберігати дані для доступу з будь-якої частини програми та відновлювати їх після критичного збою програми або її закриття. Є декілька рішень для створення локального сховища даних в застосунку.

Redux є бібліотекою для керування станом програми, спрощуючи зберігання та оновлення даних в єдиному сховищі [10]. Це дозволяє зручно та ефективно керувати станом застосунку, адже всі дані зберігаються у єдиному об’єкті. Завдяки цьому підходу, Redux легко маштабується, уникнувши проблем доступу чи продуктивності при великому обсязі даних. Бібліотека є подієво-орієнтованою, що робить його інтеграцію в бізнес-процес застосунку дуже простою. Ідеально поєднується з React Native, завдяки високоякісному Redux API для React, такому як React Redux.

Важливо відзначити, що Redux сховище не зберігає свій стан після закриття застосунку. Для вирішення цього, можна використовувати плагін redux-persist. За допомогою цього плагіна можна налаштувати, які елементи в сховищі слід відновлювати при наступному запуску програми.

SQLite є компактною реляційною базою даних, широко використовуваною у мобільних додатках для зберігання та керування різноманітною інформацією. У сфері розробки для Android, вона є найпоширенішим рішенням для опрацювання даних. Вона відзначається легкістю, економічністю, високим рівнем безпеки і підтримкою стандартних SQL-запитів, що робить її універсальною для використання на різних платформах [1].

SQLite застосовується для зберігання різноманітних даних, таких як налаштування програм, історії відвідувань, контактні списки та повідомлення в мобільних додатках. При використанні SQLite слід враховувати обмеження, такі як максимальний розмір бази даних і обсяг рядків, і вдосконалювати SQL-запити для ефективної обробки великих обсягів інформації. В контексті React

Native, існує бібліотека react-native-sqlite-storage, що забезпечує API для роботи з SQL-базою даних (запис, читання, створення транзакцій), яка реалізується нативно.

Використання бібліотеки Redux є найбільш вигідним, оскільки вона спрощує управління сховищем даних завдяки вбудованій підтримці у фреймворку. Додатково, за допомогою плагіна для збереження стану, можна уникнути необхідності створення власного механізму збереження стану програми.

2.5.4 Огляд бібліотек для підключення до Arduino на React Native

Як вже зазначалося, доступ до Arduino можна отримати за допомогою двох методів: прямого підключення через USB та за допомогою Bluetooth. Для встановлення з'єднання та передачі даних через USB використовується бібліотека react-native-serialport, яка є обгорткою JavaScript для нативної Java-бібліотеки UsbSerial. Однак важливо відзначити, що ця бібліотека більше не підтримується розробником. Для впровадження цієї бібліотеки в сучасні версії фреймворку необхідно внести зміни в код сборки цієї бібліотеки.

Ефективним та широко використовуваним варіантом для взаємодії з Bluetooth-пристроями у React Native є бібліотека react-native-ble-plx. Ця бібліотека забезпечує можливість сканувати всі пристрої, що підтримують Bluetooth Low Energy (BLE), в даному оточенні та встановлювати з'єднання з конкретним пристроєм.

РОЗДІЛ 3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ СТАРТУ СПОРТИВНИХ ЗМАГАНЬ

3.1 Вимоги до інформаційної системи

У рамках проведення стейджових змагань потрібно створити інформаційну систему, що дозволяє автоматизувати проведення старту. Ця інформаційна система передбачає використання як апаратної, так і програмної складових. Перша компонента відповідає за автоматичний збір даних та функцію воріт, що дозволяють контролювати рух учасників. Друга складова призначена для використання оператором старту стейджа, який має можливість відстежувати стан черги, вносити зміни в її порядок та керувати воротами за допомогою команд.

3.2 Вимоги до програмного продукту

Для початку проектування та створення повноцінного програмного застосунку необхідно було виявити усі функціональні та нефункціональні вимоги, котрим вона повинна задовольняти.

3.2.1 Функціональні вимоги до програмного продукту

Для створення повноцінного застосунку необхідно, котрий задовольняв усім умовам інформаційної системи було необхідно створити можливість:

- логіну до програми у якості оператора спортивного змагання компанії, котра проводить дані змагання;
- перегляд поточних змагань компанії, до якої залучен оператор старту;
- приєднання оператора до відповідному стейджа івента;
- відкриття та закриття стейджа;
- менеджмент черги атлетів під час виступу (зміни порядку черги, дискваліфікація атлета).

- Під'єднання застосунку до воріт за допомогою Bluetooth або Usb з`єднання.

3.2.2 Нефункціональні вимоги до програмного продукту

Ефективність:

- Програмний продукт має бути сумісним з більшістю мобільних пристроїв, що працюють під операційною системою Android.
- Програмний застосунок повинен мати гарну швидкодію на більшості пристроях.

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.
- Інтерфейс застостосунку повинен бути інтуїтивно зрозумілим та не перевантаженим для максимального комфортного досвіду роботи з інформаційною системою.

Безпека:

- Доступ до інформації повинен бути обмеженим.
- Перед використанням системи повинні пройти аутентифікацію користувача.

3.3 Розробка інформаційної системи

Розробка програмного продукту була розділена на 4 частини: проектування інформаційної системи, визначення методів взаємодій з сервером (реалізація API), реалізація програмної частини та 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. Рис. 3.1).

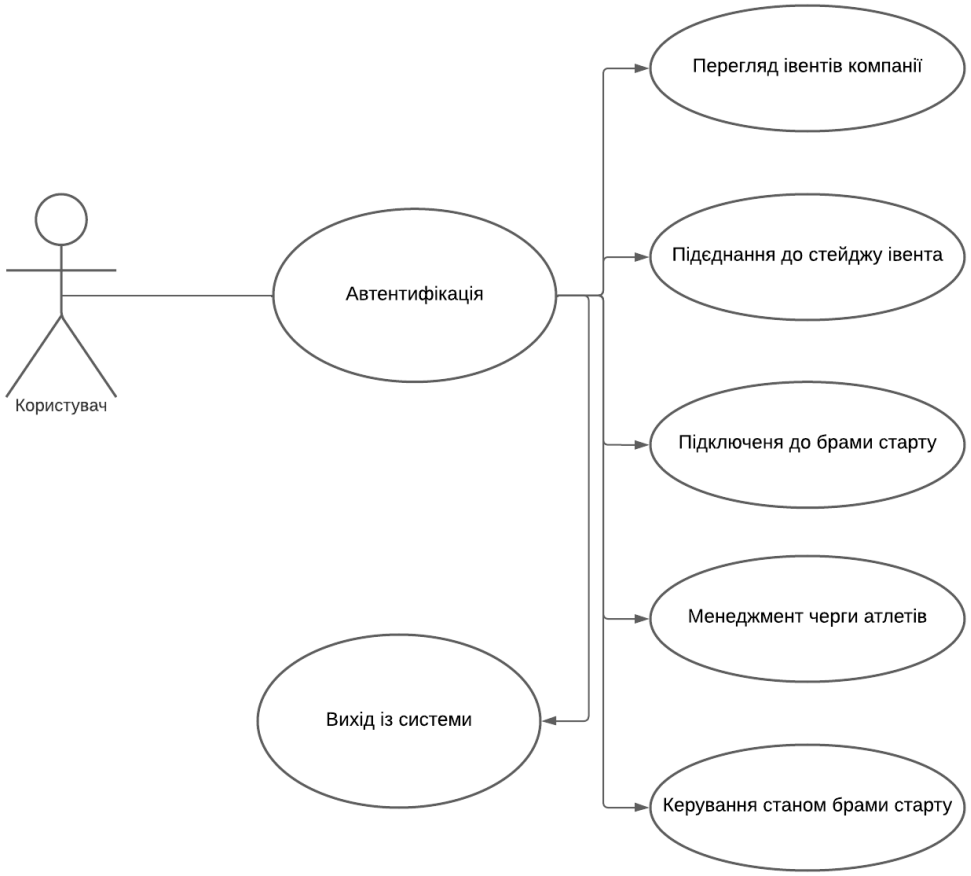

Рисунок 3.1 — Use-case діаграма мобільного застосунку для старту атлетів

На даної Use Case діаграмі видно, що додаток виконує роль керуючого блоку, взаємодіючи як з апаратною частиною комплексу (воротами), так і забезпечуючи можливість управління атлетами та їх порядком, включаючи зміну порядку виступу, введення заборон на виступ або повернення атлета на етап виступу. Також в додатку є можливість аутентифікації, оскільки лише конкретні користувачі, яким була видана роль оператор (oper), перед початком

змагань у системі, можуть пройти процедуру входу та користуватися додатком. Крім того, у додатку існує можливість, за якої оператори можуть заздалегідь приєднатися до конкретного етапу змагань до їх початку.

Для визначення структури класів та їх взаємодії між собою була створена діаграма класів, на якій видно всю архітектуру програмного застосування.

В програмному застосунку реалізован паттерн програмування Ін`єкція залежностей (Dependency Injection). Даний паттерн реалізує підхід до керування залежностями в програмному забезпеченні. В основі ін`єкції залежностей лежить ідея про те, що об'єкти не повинні самостійно створювати або залежати від інших об'єктів, які вони використовують (залежностей). Замість цього, залежності надаються об'єкту ззовні [11].

В застосунку контейнером залежностей є класс BaseContext, котрий реалізує інтерфейс IContextContainer, який описує усіх серверів та воркерів в програмі.

Класи, які спадкують клас Entity — копіюють структуру схеми бази даних та визначають її в застосунку. Також в ціх классах знаходиться реалізація API до відповідних таблиць бази даних, реалізованих на серверної частини всієї системи.

Однією з окремих категорій класів є ті, які відповідають за відображення контенту на відповідних екранах і обробку подій, таких як натискання кнопок чи зміна стану класу, що виконується на поточному екрані.

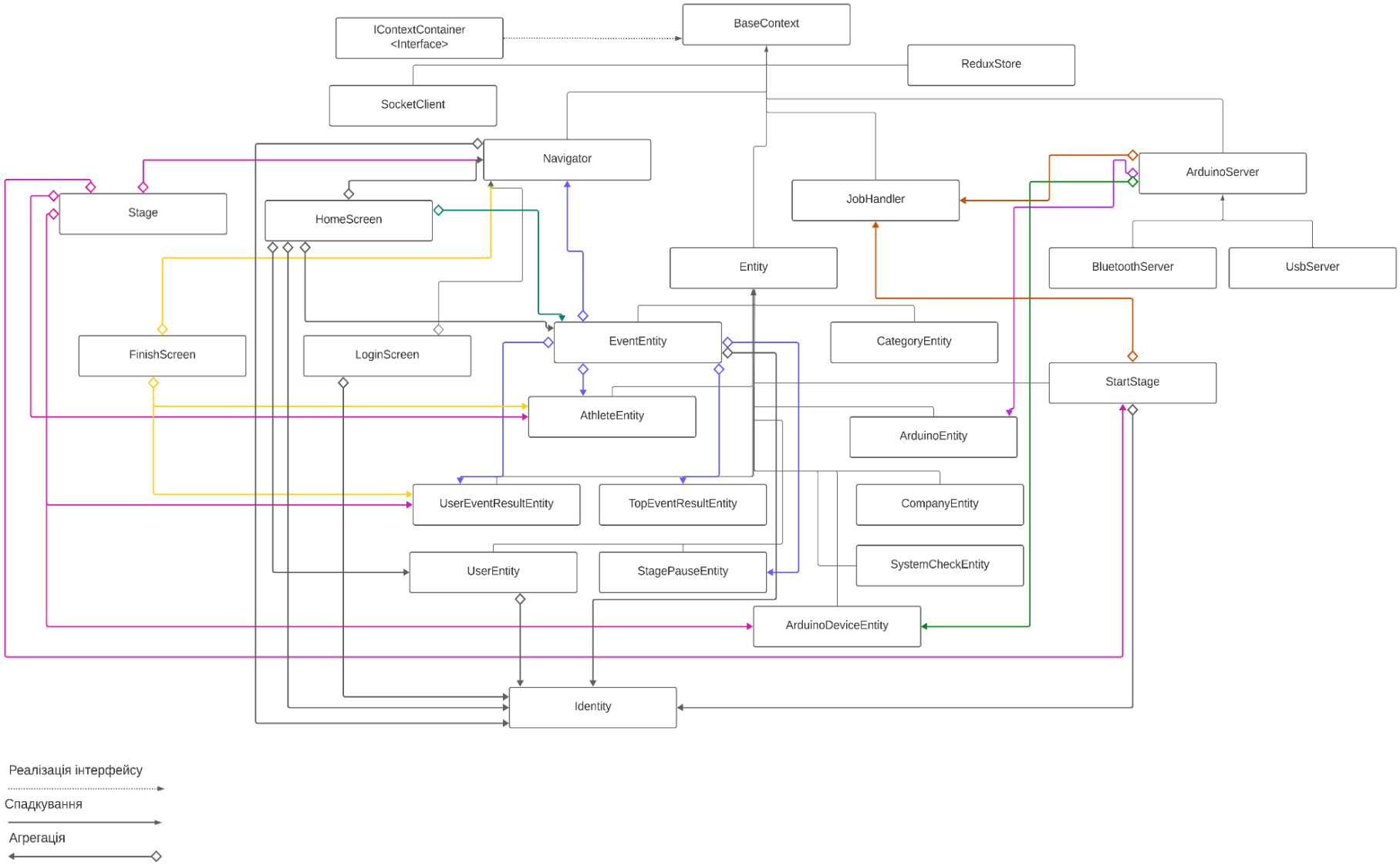

Рисунок 3.2 — Діаграма класів програмного застосунку

Оскільки система старту є лише складовою частиною автоматизації спортивних стейджових змагань, то основна обробка даних та передача інформації до кінцевих систем виконується централізованим сервером (див. Рис. 3.3).

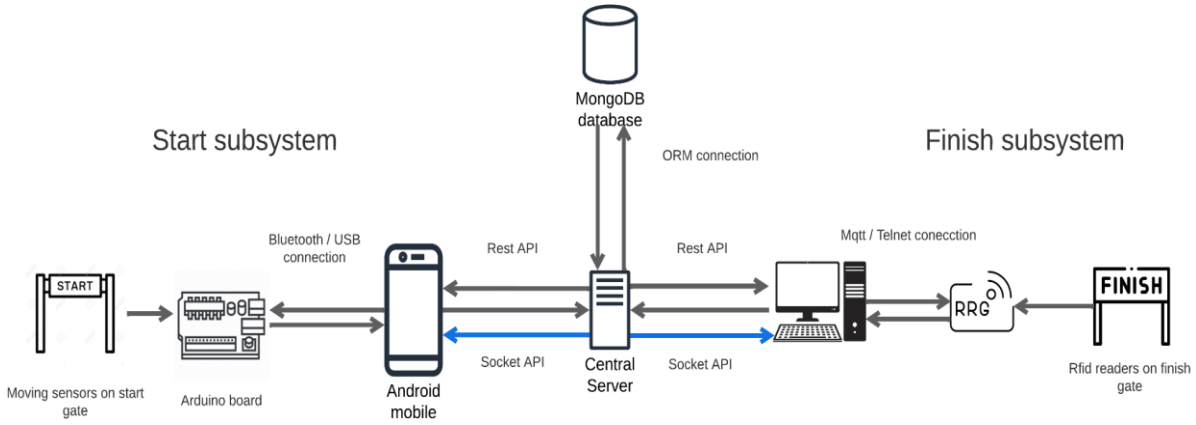

Рисунок 3.3 — Схема роботи системи проведення стейджових спортивних змагань

Взаємодія зі системою старту та сервером здійснюється за допомогою REST API для обміну інформацією до старту змагань (автентифікація, отримання списку змагань та атлетів до цих змагань) та і під час старту самих змагань (відправка часу старту атлета до серверу). Для отримання інформації про зміну статусу змагання, або інформації про атлета під час змагання використовується WebSocket-з'єднання для швидкого інформації від сервера.

3.3.2 Реалізація програмної частини

1. Реалізація контейнеру для інєкції залежностей є в інтерфейсі IContextContainer, в котрому реєструється усі класи типу Entity да деякі сервері, котрі необхідні для комунікації між програмою та сервером або Arduino. Контейнер залежностей створюється за допомогою бібліотеки для інверсії керування awilix. Вона надає простий та потужний механізм для створення, управління та виправлення залежностей в програмному забезпеченні [2]. Ос-

новна ідея Awilix полягає в тому, що вона дозволяє організувати ваше програмне середовище у вигляді контейнера залежностей. Цей контейнер реєструє та зберігає інформацію про залежності вашої програми, тобто об'єкти, які вона використовує. Одним з головних переваг Awilix є його можливість автоматично розрішувати залежності за допомогою інжектора. Можна вказати, які залежності потрібні для вашого об'єкта, і Awilix автоматично створить ці об'єкти та надасть їх при необхідності (див. Лістинг 1).

Лістинг 1 — інтерфейс awilix-контейнеру «IContextContainer»

```
import * as awilix from 'awilix';

export interface IContextContainer {
   config: any;
   redux: ReduxStore;
   navigator: Navigator;
   socket: SocketClient;
   arduino: ArduinoServer;
   job: Job;
   queue: (
     eventId: string,
     stageId: string,
     filter?: QueueFilter,
   ) => EntityList<IOperativeItem>;
   initSocket: () => void;
   toast: (text: string, msgType?: MessageType) => void;
}

const container: awilix.AwilixContainer<IContextCon-
tainer> =
   awilix.createContainer({
     injectionMode: awilix.InjectionMode.PROXY,
   });
```

Цей контейнер сприяє реалізації ще одного програмного патерну — "Singleton", завдяки реєстрації в якості єдиного екземпляра та веденню відстеження за екземпляром, який заноситься в контейнер [4].

Ця функціональність дозволяє забезпечити, що в програмі існує лише один екземпляр об'єкта, реалізованого через цей контейнер, сприяючи ефективному управлінню ресурсами та унікальною ідентифікацією цього об'єкта (див. Лістинг 2).

Лістинг 2 — реєстрація класів до контейнеру залежностей

```
const initSocket = (ctx: IContextContainer) => {
    return () => {
      const socket = ctx.socket;
      socket.addListener(
           ctx.UserEventResultEntity.entityName,
           ctx.UserEventResultEntity.socketMiddleware,
      );
      socket.addListener(
           ctx.AthleteEntity.entityName,
           ctx.AthleteEntity.socketMiddleware,
      );
      socket.addListener(
           ctx.EventEntity.entityName,
           ctx.EventEntity.socketMiddleware,
      );
      socket.start();
    };
};

const toast = (ctx: IContextContainer) => {
    return (
      text: string,
      msgType: MessageType = MessageType.SUCCESS,
      title: string = '',
    ) =>
      Toast.show({
           type: msgType,
           text1: title ? title : text,
           position: 'bottom',
      });
};

container.register({
    ...entities,
    queue: awilix.asFunction(athleteQueue).singleton(),
    toast: awilix.asFunction(toast).singleton(),
    config: awilix.asValue(coreConfig),
    redux: awilix.asClass(ReduxStore).singleton(),
    navigator: awilix.asClass(Navigator).singleton(),
```

```
    socket: awilix.asClass(SocketClient),
    arduino:    awilix.asClass(BluetoothServer).single-
ton(),
    job: awilix.asClass(Job).singleton(),
    initSocket:   awilix.asFunction(initSocket).single-
ton(),
});
```

2. Для оперативного отримання даних для програмним застосунком від сервера під час змагань створюється сокет-з'єднання. Зі сторони клієнтського застосунку використовується бібліотека для створення WebSocket з`єднання socket.io-client. Клас SocketClient є обгорткою над socket.io і реалізує увесь функціонал бібліотеки (створення, знищення з`єднання, обробка подій на передачу даних) та надає у вигляді методів (див. Лістинг 3).

Лістинг 3 — Ініціалізація екземпляру класса «SocketClient»

```
export default class SocketClient extends BaseContext {
      private socket: any = null;
      private rooms = {};

      constructor(opts: IContextContainer) {
      super(opts);
      this.onSocketData = this.onSocketData.bind(this);
      }

      public async start() {
      if (!this.socket) {
        this.stop();
        const url =
            config.socket.host +
            (config.dev && config.socket.port
                ? ':' + config.socket.port
                : '');

        let sessionKey = await AsyncStorage.getItem('@ses-
sion_key');
        if (sessionKey === null) {
            sessionKey = shortid.generate();
            await     AsyncStorage.setItem('@session_key',
sessionKey);
        }

        this.socket = io(url, {
```

```
            query: {
                sessionKey,
            },
          });
          this.socket.on('connect', this.onConnect);
          this.socket.on('data', this.onSocketData);
          this.socket.on('connect_error',        this.onSock-
etError);
        }
        }
public stop() {
        if (this.socket) {
          this.socket.disconnect();
          this.rooms = {};
        }
        }
public async acceptRequest(hash: string) {
        const   key   =   await   AsyncStorage.getItem('@ses-
sion_key');
        this.socket.emit('accept', {hash, key});
        }

        private onSocketData(data) {
        Object.keys(this.rooms).map(entityCallback => {
          this.rooms[entityCallback](data);
        });
        if (data.hash) {
          this.acceptRequest(data.hash);
        }
        }
}
```

3. Для роботи з Arduino в залежності від типу підключення має свої відмінності, тому було зроблено уніфікований клас ArduinoServer, який надає інтерфейс для комунікації між Arduino та програмним застосунком, а класи котрі його успадковують — реалізують специфіку під'єднання та трансферу даних між пристроями (див. Лістинг 4).

Лістинг 4 — Класс «ArduinoServer»

```
export default abstract class ArduinoServer extends BaseCon-
text {
    constructor(opts: IContextContainer) {
        super(opts);
        this.onConnected = this.onConnected.bind(this);
```

```
        this.onReadData = this.onReadData.bind(this);
    }

    protected abstract checkDevice(): void;
    protected abstract startListener(): void;
    protected abstract stopListener(): void;
    protected abstract writeMessage(data: string): void;

       protected onConnected(deviceName: string = '') {
       const {
         redux: {dispatch, persistor},
       } = this.di;
       dispatch(
         updateStartStage({
              sensor: GateSensor.CLOSED,
              deviceName: deviceName,
         }),
       );
       persistor?.flush();
       }

    private turnOffSensor() {
        const {
          redux: {state, dispatch},
        } = this.di;
        const sensor = state().startStage?.get('sensor');
        if (sensor !== GateSensor.UNAVAILABLE) {
          dispatch(
          updateStartStage({
                    sensor: GateSensor.UNAVAILABLE,
          }),
          );
        }
    }
}
```

Arduino взаємодіє із мобільним пристроєм, відправляючи дані у формі пакетів, які формуються у вигляді рядка <string:value>, де вказується тип пакета та відповідне значення, розділені двокрапкою.

Кожна мітка вказує на різні характеристики або операції. Реакція мобільного додатка на отримані пакети визначається типом мітки, що дозволяє виконувати різноманітні операції або виводити різні результати відповідно до програмної логіки додатку (див. Лістинг 5).

Лістинг 5 — Функція обробки вихідних пакетів Arduino

```
  protected onReadData(data: string) {
    const {
         StartStage,
         SystemCheckEntity,
         redux: {dispatch, state},
    } = this.di;
    const payload = base64.decode(data);
    console.log('Arduino sent: ', payload);
    let splitted: string[] = payload.split(':');
    if (splitted.length === 2) {
         let  pingAt  =  state().startStage?.get('pin-
gAt');
         let  sensor  =  state().startStage?.get('sen-
sor');
         switch (splitted[0]) {
              case 'first':
                   dispatch(StartStage.actions.open-
Gate());
                   break;
              case 'checked':
                   console.log('onReadData CHECK to ar-
duino');
                   dispatch(SystemCheckEntity.ac-
tions.sendCheckResults());
                   break;
              case 'ping':
                   pingAt = Date.now();
                   break;
              case 'tone':
                   console.log('onReadData toneeee');
                   break;
              default:
              case 'info':
                   break;
         }
         const value = Number.parseInt(splitted[1], 10);
         if (
              (sensor === GateSensor.OPENED && value >=
1) ||
              (sensor === GateSensor.CLOSED && value <
1)
         ) {
              dispatch(
                   updateStartStage({
                        pingAt,
                        sensor:
```

```
                                value >= 1 ? GateSen-
    sor.CLOSED : GateSensor.OPENED,
                        }),
                    );
                }
            }
        }
```

4. Класс ArduinoServer наслідують два класи : BluetoothServer та UsbServer. Дані класи відповідні за те, яким чином телефон буде підєднатися до Arduino : через Usb зєднання, або по Bluetooth. Для з'єднання з платої Arduino використовується окремий Bluetooth-модуль та бібліотека “react-native-ble-plx” , котра дозволяє дозволяє відстежити всі найближчі Bluetooth пристрої та приєднатися до них. Для того щоб працювати з Arduino через Usb-з'єднання в класі UsbServer реалізована підєднання за допомогою бібліотеці “react-native-serialport” (див Лістинг 6) .

Лістинг 6 — Функції підключення до Arduino на Usb та Bluetooth

```
 public startListener() {
        DeviceEventEmitter.addListener(actions.ON_SER-
VICE_STARTED, (response) => {
              if (response.deviceAttached) {
                   this.onDeviceAttached();
              }
          }
        );
        RNSerialport.setAutoConnectBaudRate(parseInt("9600",
10));
        RNSerialport.setInterface(parseInt("-1", 10));
        RNSerialport.setAutoConnect(true);
        RNSerialport.startUsbService();
        }
private async connectedToDevice(device: Device) {
        this.device = device;
        const services = await device.discoverAllServicesAnd-
Characteristics();
        const characteristic: any = await this.get-
ServicesAndCharacteristics(services);
        this.serviceId = characteristic.serviceUUID;
        this.characteristicsId = characteristic.uuid;
        const deviceName = device.name || device.localName ||
'';
```

```
        this.manager.requestMTUForDevice(device.id, 128);
        device.services().then((services: Service[]) => {
          const snifferService = services.find(
               (service: any) => service.uuid === this.ser-
viceId,
          );
          snifferService.characteristics().then((character-
istics: Characteristic[]) => {
                    const voltageCharacteristic = characteris-
tics[0];
                    if (voltageCharacteristic) {
                         this.voltageCharacteristic = volt-
ageCharacteristic;
                         this.onConnected(deviceName);
                         voltageCharacteristic.monitor((er-
ror: any, c: any) => {
                              if (error) {
                                   this.onError(error, 4);
                                   return;
                              }
                              this.onReadData(c.value);
                         });
                    }
               })
               .catch((error: any) => this.onError(error, 3));
        });
        return      device.discoverAllServicesAndCharacteris-
tics();
        }
```

В класі BluetoothServer реалізован власна функція сканування пристроїв, так як бібліотека, котра працює з підєднанням до BLE пристроїв дозволяє не використовувати дефолтні функції.

Дана функція дозволяє фільтрувати пристрої за названням, та й у разі підключення записує в Redux запис про те, що даний девайс колись був під’єднаний до телефону (див. Лістинг 7).

Лістинг 7 — Функція сканування пристроїв з увімкненим Bluetooth в окрузі.

```
     private scanDevices(error: any, device: Device | null) {
          if (error) {this.onError(error, 1); return;
          }
          const {redux: {state, dispatch},} = this.di;
```

```
          const deviceName = state().startStage?.get('device-
Name');
          const nameToCheck = deviceName && deviceName.length
> 0 ? deviceName : '--';

          if (device?.name?.startsWith('--')) {
               const {ArduinoDeviceEntity} = this.di;
               const  actionFn  =  ArduinoDeviceEntity.getAc-
tion();
               let tempDevice = state().arduinoDevices
                    ? {
                         ...state().arduinoDevices.toJS(),
                         [device.id]: {
                              id: device.id,
                              deviceName: device.name,
                              localName: device.localName,
                         },
                         }
                    : {
                         [device.id]: {
                              id: device.id,
                              deviceName: device.name,
                              localName: device.localName,
                         },
                         };
               dispatch(actionFn.success(null, {entities:
                    { arduinoDevices: tempDevice,}
                    ,}
                    ),);
          }

          if (this.manager &&
               device?.name?.startsWith(nameToCheck) &&
               this.connectingToDeviceId !== device.id
          ) {
               setTimeout(()   =>   {this.manager.stopDeviceS-
can();}, 60000);
               this.connectingToDeviceId = device.id;
               if  (!this.device  ||  this.device.id  !==  de-
vice.id) {
                    this.manager.connectToDevice(device.id,
{autoConnect: true})
                         .then(this.connectedToDevice)
                         .then(
                              () => {this.connectingToDeviceId
= '';},
                              error   =>   this.onError(error,
2),);
               }
```

```
        }
      }
```

5. У кожного класу-сутності в системі створюються саги, які призначені для очікування та обробки відповідних подій, які можуть виникнути в Redux.

Ці саги впроваджують логіку, яка виконується перед тим, як дані будуть відправлені до сховища Redux. Використання даного підходу реалізовано завдяки бібліотеці Redux Saga та створеному декоратору, який спрощує та структурує процес взаємодії між класами та сагами (див. лістинг 8).

Лістинг 8 — Декоратор «saga»

```
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[enti-
tyName][actionName];
                         if (!act.isAdded) {
                              const watcherFunc =
                                   en-
tity[act.watcher].bind(entity);
                              let func = function* () {
                                   while (true) {
                                   const data = yield take(ac-
tionName);
                                   delete data.type;
                                   yield     fork(watcherFunc,
data);
                                   }
                              };
                              Entity.mSagas.push(fork(func));
                              act.isAdded = true;
                         }
                    }
               } else {
```

```
                    throw new Error(`Action [${actionName}]
does not belong to the entity`,);
                }
          });
        } else {
          Entity.mSagas = [];
        }
        };
};
```

6. Для отримання або сповіщення про події, на які спостерігають саги в класах сутностей, в застосунку використовуються функціональні хуки. Хук useEntity повертає посилання на об'єкт класу-сутності а хук useActions повертає об'єкт з анонімними функціями створення події, на яку дивляться саги відповідної сутності (див. Лістинг 9).

Лістинг 9 — Хуки «useEntity, useActions»

```
import {useContext} from 'react';
import {useDispatch} from 'react-redux';

export default function useEntity(entityName: string) {
   const di = useContext(ContainerContext);
   return di.resolve(entityName);
}

export function useActions(entityName: string) {
   const dispatch = useDispatch();
   const entity: any = useEntity(entityName);
   const keys = Object.keys(entity.actions);
   const dispatches = {} as any;
   for (let i = 0; i < keys.length; i++) {
     dispatches[keys[i]] = (data: any) =>
          dispatch(entity.actions[keys[i]](data));
   }
   return dispatches;
}
```

6. Щоб забезпечити єдність усіх класів-сутностей у проекті, був створений клас Entity. Цей клас є батьківським класом для тих класів, котрі ініціаліхують схему моделі даних у Redux та надає API для створення запитів та запису даних до Redix-сховища напряму. (див. Лістинг 10).

Лістинг 10 — Ініціалізація схеми в класі « Entity»

```
protected mEntityName: string;
    protected mSchema: Schema;

    private static mStore: Store;
    public static mSagas: any[] = [];
    private static mUser: IIdentity = null;
    private static mContext: any;
    public static mActions: ISagaItem = {};



    public get schema() {
        return this.mSchema ? this.mSchema : null;
    }

    constructor(opts: IContextContainer) {
        super(opts);
        this.initSchema();
        this.syncReduxWithServer         =         this.syn-
cReduxWithServer.bind(this);
        this.directlyWriteToRedux      =      this.directly-
WriteToRedux.bind(this);
    }


 public static addSaga(...args: any[]) {
        args.forEach(item => {
            if (item instanceof Function) {
                Entity.mSagas.push(fork(item));
            }
        });
    }
    protected initSchema(
        name = 'entity',
        definitions: any = {},
        options: any = {},
    ) {
        this.mEntityName = name;
        this.mSchema =
          name !== 'entity'
          ? [new schema.Entity(name, definitions, options)]
          : null;
    }
```

Додаток використовує бібліотеку normalizr для створення структури, аналогічної тій, яка є в базі даних. Ця бібліотека дозволяє нормалізувати вкладені об'єкти JSON, застосовуючи основні принципи нормалізації баз даних [5]. Ця бібліотека ефективно застосовує процес нормалізації до як простих, так і складних структур, включаючи ті, що містять вкладені сутності (див Лістинг 11).

Лістинг 11 — Приклад нормалізації простої та вкладеної структури.

```
// Проста структура
this.initSchema(ENTITY.CATEGORY, {
        companyId: new schema.Entity(ENTITY.COMPANY),
});
// Вкладена структура
this.initSchema(ENTITY.ATHLETE, {
          user: new schema.Entity(ENTITY.USER, {
stages:      [{currentResult:      new      schema.Entity(EN-
TITY.USER_EVENT_RESULT,
          {},
          {idAttribute: 'uuid'},
        ),
bestAthlete: new schema.Entity(ENTITY.ATHLETE, {
        user: new schema.Entity(ENTITY.USER, {
stages:      [{currentResult:      new      schema.Entity(EN-
TITY.USER_EVENT_RESULT,{},
          {idAttribute: 'uuid'},
        ),
        },
],
}),
}),
},
],
```

```
}),
event: new schema.Entity(ENTITY.EVENT),
category: new schema.Entity(ENTITY.CATEGORY),
});
```

Додатково, клас Entity в системі відповідає за основний функціонал для надсилання API-запитів на сервер. Цей клас об'єднує в собі засоби для формування та відправлення різноманітних запитів, що дозволяє ефективно взаємодіяти з сервером. Його функціональність може включати в себе встановлення параметрів запиту, обробку відповідей сервера, та інші операції, які є важливими для взаємодії з серверною частиною програми. Використання класу Entity дозволяє стандартизувати та узагальнити процес надсилання API-запитів, забезпечуючи їхню коректну та ефективну обробку у всіх класах системи.

Це покращує роботу програми та спрощує розробку, дозволяючи легко розширювати чи модифікувати взаємодію з сервером в майбутньому (див. Лістинг 12).

Лістинг 12 — Функція обробки даних та створення запиту в класі Entity

```
protected *actionRequest(
      uri: string,
      crud: CRUD,
      method: HTTP_METHOD,
      data?: any,
      ) {
      const action = this.getAction(crud);
      yield put(action.request(data));
      let success = true;
      let message = {} as IMessageBlock;
      let pager = null;
```

```
        const token = yield select((state: AppState) =>
(state.identity && state.identity.user && state.iden-
tity.user.token) || null,);
        const result = yield call(this.xFetch, uri, method,
data, token);
        success = result.success;
        if (result.response.message && !isEmpty(result.re-
sponse.message)) {
          this.di.toast(result.response.message, success ?
MessageType.SUCCESS : MessageType.ERROR,);
        }
        const query = result.response.data;
        let response = {} as any;
        if (success && this.mSchema && query) {
          response = normalize(camelize-
Keys(JSON.parse(JSON.stringify(query))), this.mSchema);
        } else if (query) {
          response = query.data;
        }
        if (success) {
          yield put(action.success(data, response));
        } else {
          yield put(action.failure(data, response));
        }
        return {response, message};
        }
```

Для виконання задачі синхронізації зміни статусу атлета, такої як дискваліфікація чи пропуск власного старту на стейджі, використовується функція directlyWriteToRedux. Ця функція дозволяє створювати запис для негайної синхронізації даних із сервером (див. Лістинг 13).

Лістинг 13 — Функція дострокового запису зміни атлета до сховища

```
    public *directlyWriteToRedux(data: any) {
         if (this.mSchema) {
              data = Array.isArray(data) ? data : [data];
              const  curdAction  =  this.getAction(CRUD.UP-
DATE);
              try {
                   const  response  =  normalize(camelize-
Keys(JSON.parse(JSON.stringify(data))),this.mSchema,);
                   const idField = this.mSchema?.length > 0 ?
this.mSchema[0].idAttribute : 'id';
                   for (let i = 0; i < data.length; i++) {
                        yield put(putUpdates(this.entityName,
data[i][idField]));
                   }
                   yield  put(curdAction.success(data,  re-
sponse));
                   if (this.entityName !== 'entity') {
                        const type = 'sync' +
                             this.entityName[0].toUpperCase()
+
                             this.entityName.slice(1);
                        yield put(action(type, {}));
                   }
              } catch (e) {
                   console.log('error:
directlyWriteToRedux()', e);
              }
          }
        }
```

6. Класи які наслідують клас Entity потрібні для збереження та адаптації структури бази даних сервера в умовах мобільного застосунку з використанням Redux-сховища. До таких класів відносяться : EventEntity, CategoryEntity, ArduinoEntity, ArduinoDeviceEntity, SystemCheckEntity, Identity, StagePauseEntity, UserEventResultEntity, TopEventResultEntity, AthleteEntity.

Клас EventEntity впроваджує структуру таблиці для відображення та управління інформацією про спортивні змагання в системі. Даний клас виконує важливі функції, включаючи отримання та збереження оновлень від відповідних компаній, в яких працює оператор.

Також, він надає можливість приєднання та закріплення оператора за конкретним етапом змагань. Завдяки цьому класу реалізована обробка повідомлень, що надходять через сокет, при скиданні події, а також відтворення всіх етапів змагань з самого початку (див. Лістинг 14).

Лістинг 14 — Функціонал роботи з подіями в класі EventEntity.

```
protected onSocketEvent(data: any, code: string = '') {
       const {redux: {state, dispatch}, AthleteEntity, UserEventResultEntity, TopEventResultEntity, StagePauseEntity, navigator,} = this.di;
       const {startStage} = state();
       if (code === 'reset_event' && Array.isArray(data.result) && data.result.length > 0) {
         if      (data.result.includes(startStage.get('eventId'))) {
              dispatch(updateStartStage({eventId:        '', stageId: '', leftTime: 0, athleteId: '', startTime: 0, status: StageStatus.OPENED,gateStatus: GateStatus.CLOSED,}),
              );
         }
         dispatch(AthleteEntity.clear());
         dispatch(UserEventResultEntity.clear());
         dispatch(TopEventResultEntity.clear());
         dispatch(StagePauseEntity.clear());
         const navTo = {menu: 'Home', screen: 'Home', params: {}};
         navigator.replace(navTo.menu,              {params: navTo.params, screen: navTo.screen,});
       }
       }
       @action()
       public *saveEvent(data: any) {
       const    {response}    =    yield    call(this.xSave, '/event/save', data);
       const {Identity} = this.di;

       if (response) {
         yield put(Identity.actions.updateIdentity());
       }
       }
       @action()
       public *bindOperToStage(stage: IDisplayStage) {
       const {events} = yield select(state => {
         return {
              events: state[ENTITY.EVENT],
```

```
        };
      });
      if (stage.eventId && stage.stageId) {
        const event = events.find(e => e.get('id') ===
stage.eventId);
        if (event) {
            const ev = event.toJS();
            const st = ev.stages.find(s => s.id ===
stage.stageId);
            if (st) {
                st.userId = stage.userId;
                yield call(this.directlyWriteToRedux, ev);
            }
        }
      }
      }
```

Клас CategoryEntity реалізує структуру таблиці категорій, в котрих виступає атлет. Клас ArduinoEntity надає API, яке реалізує відправку команд до Arduino, такі як забрати перший сигнал від датчиків брами, ігнорувати всі сигнали від датчиків. В цьому класі також існує можливість автоматичного створення запису про початок гонки при отриманні сигналу про розрив датчиків на старті (див. Лістинг 15).

Лістинг 15 — Функція запису старта атлета класса ArduinoEntity

```
       @action()
      public *createArduinoRecord() {
      const {eventId, stageId, userId, categoryId, ath-
leteId} = yield select(
        store => {
            return {
                ...store.currentStage,
                athleteId: store.currentRider?.curr?._id,
                userId:                    store.curren-
tRider?.curr?.user?._id,
                categoryId:                store.curren-
tRider?.curr?.category,
            };
        },
      );

      const uuid = uuidv4();
      yield put(
        dispatchActions.putOscillationTimeFirst({
```

```
            eventId,
            uuid: uuid,
            categoryId,
            stageId,
            userId,
            athleteId,
            startTime: Date.now(),
        }),
      );
      }
```

Клас SystemCheckEntity реалізує перевірку датчиків брами перед початком старту етапу та відправляє даний результат до серверу (див. Лістинг 16).

Лістинг 16 — Функція обробки та відправки результатів перевірки брами

```
    @action()
      public *sendCheckResults() {
        const {arduino} = this.di;
        this.successCheckCount = this.successCheckCount +
1;
        if (this.successCheckCount >= CHECK_COUNT) {
            let {stageId, systemCheck} = yield se-
lect((store: any) => ({stageId: store.flagger.Selected-
Stage.stageId, eventId: store.flagger.SelectedStage.eventId,
                systemCheck: store[ENTITY.SYSTEM_CHECK]
                    .get(store.flagger.startCheckTimer)
                    ?.toJS(),}));
            if (systemCheck) {
                let signal = systemCheck.stageSig-
nals.find((stageCheck: any) => stageCheck.stage ===
stageId,);
                systemCheck.needToCheck = false;
                if (!signal) {signal = { stage: stageId,};
systemCheck.stageSignals.push(signal);}
                signal.status = true;
                yield call(this.directlyWriteToRedux, sys-
temCheck);
                this.successCheckCount = 0;
                arduino.sendClosePrefix();
                yield put(setFlagger('startCheckTimer',
false));
            }
        }
      }
```

Клас ArduinoDeviceEntity є посередником між для Arduino сервера та класів відображення який надає API для зміни підключення девайса та надає команду серверу підключитися до нового девайса та перезапустити свої слухачі для коректної праці з еовим пристроєм (див. Лістинг 17).

Лістинг 17 — API класу ArduinoDeviceEntity

```
@action()
   public *handleChangeDevice(data: any) {
     const {
          redux: {state, dispatch},
     } = this.di;

     if    (state().startStage?.get('deviceName')    !==
data.device) {
          dispatch(
               updateStartStage({
                    deviceName: data.device,
               }),
          );
          this.di.arduino.start();
     }
   }
```

Клас Identity надає API для логінації у якості оператора етапу змагання. Також, після логінації до застосунку надходять токени відповідного користувача, які потрібні для здійснення усіх останніх запитів в застосунку.

Клас StagePauseEntity відповідає за накопичення та сінхронізації паузи етапа змагання з сервером для відображення данної паузи на фініші або вебсторінці а також для перерахунку часу виступу усіх атлетів, котрі ще не встигли виступити на відповідному етапі.

Клас UserEventResultEntity реалізує структуру таблиці сервера userEventResult та надає API для створення та синхронізаії запису з результатом старту з сервером та отримання результатів атлетів, котрі вже фінішували даний етап змагання (див. Лістинг 18).

Лістинг 18 — API для отримання та синхронізації створених результатів

```
@action()
        public *syncUserEventResults() {
        yield  call(this.syncReduxWithServer,  '/userEventRe-
sults/saveResults');
        }

        @action()
        public *fetchResults(data) {
        const startStage: StartStageType = yield select(
          state => state.startStage,
        );
        const eventId = data.eventId ? data.eventId : start-
Stage.get('eventId');
        const stageId = data.stageId ? data.stageId : start-
Stage.get('stageId');

        if (eventId && stageId) {
          yield call(
               this.xRead,
               `/userEventResults/getByStage/${even-
tId}/${stageId}`,
          );
        }
        }
```

Клас TopEventResultEntity реалізує структуру таблицю сервера topEventResult та API для отримання найкращих результатів за категоріями для таблиці лідерів.

7. Для початкової налаштування локального сховища використовується клас ReduxStore. Цей клас ініціалізує Redux-сховище у програмі та відновлює дані, які раніше були збережені в кеші програми та надає API для керування станом збережених даних поза цим класом.

Для збереження та відновлення даних використовуються плагіни "redux-persist" та "redux-persist-transform-immutable". Остання з них дозволяє зберігати дані в сховищі у формі, яка залишається імутабельною, забезпечуючи стійкість даних у сховищі (див. Лістинг 19).

Лістинг 19 — Ініціалізація Redux-сховища в класі «ReduxStore»

```
export default class ReduxStore extends BaseContext {
        private _persistedReducer;
        private _store: Store;
        private _persistor: Persistor | null;
        public get persistor(): Persistor | null {return
this._persistor;}

        public state = (): AppState => {return
this._store.getState();};

        public dispatch = (args: any): Dispatch => {return
this._store.dispatch(args);};

        constructor(opts: any) {
        super(opts);
        this._persistedReducer = persistReducer(persistCon-
fig, rootReducer);
          const {store, persistor} = this.configure-
ProdStore();
          this._store = store;
          this._persistor = persistor;
        }
        private configureDevStore = (initialState?: AppState
& PersistPartial) => {
         const middleware = [];
         const enhancers = [];

         const sagaMiddleware = createSagaMiddleware();
         middleware.push(sagaMiddleware);

         const composeEnhancers = window.__REDUX_DEVTOOLS_EX-
TENSION_COMPOSE__
             ?         window.__REDUX_DEVTOOLS_EXTENSION_COM-
POSE__({})
             : compose;
         enhancers.push(applyMiddleware(...middleware));
         const enhancer = composeEnhancers(...enhancers);
         const store: Store<AppState> = createStore(
             this._persistedReducer,
             initialState,
             enhancer,
         );
         const persistor: Persistor = persistStore(store);

         sagaMiddleware.run(this.rootSaga);
         return {store, persistor};
    };
```

```
        private configureProdStore(initialState?: AppState &
PersistPartial) {
        const sagaMiddleware = createSagaMiddleware();
        const store: Store<AppState> = createStore(
          this._persistedReducer,
          initialState,
          applyMiddleware(sagaMiddleware),
        );
        const persistor: Persistor = persistStore(store);
        sagaMiddleware.run(this.rootSaga);
        return {store, persistor};
        }
}
```

8. Клас JobHandler відповідає за обробку подій та завдань, визначених для конкретних інтервалів часу у додатку. У своїй роботі він використовує кілька таймерів, кожен із яких виконує визначені завдання зі своїм власним інтервалом. Один з таймерів відповідає за синхронізацію всіх змінених даних з додатку до сервера кожні 90 секунд. Ще один таймер слідкує за станом воріт кожну секунду. У випадку встановленої паузи на етапі, інший таймер накопичує та відправляє оновлені дані про паузу до сервера кожні 5 секунд. (див. Лістинг 20).

Лістинг 20 — Реалізація таймерів в класі «JobHandler»

```
        public start = () => {
        DeviceEventEmitter.addListener('timer', this.tick);
        };
        public stop = () => {
        DeviceEventEmitter.removeAllListeners();
        };
          private tickGate() {
        const {
          StartStage,
          redux: {state, dispatch},
        } = this.di;
        const {startStage} = state();
        const eventId = startStage?.get('eventId') || null;
        const stageId = startStage?.get('stageId') || null;
        const status = startStage?.get('status') || null;
        if (eventId && stageId && status === StageSta-
tus.STARTED) {
          dispatch(StartStage.actions.gateTick());
```

```
        }
        }
        private synchronization() {
        const {
          redux: {state, dispatch},
        } = this.di;
        const {changes} = state();

        if (changes?.size > 0) {
          changes.mapKeys((k: string) => {
               const type = 'sync' + k[0].toUpperCase() +
k.slice(1);
               dispatch(action(type, {}));
          });
        }
        }
  private updatePauseStage() {
        const {
          StartStage,
          redux: {state, dispatch},
        } = this.di;
        const {startStage} = state();
        const eventId = startStage?.get('eventId') || null;
        const stageId = startStage?.get('stageId') || null;
        const gateStatus = startStage?.get('gateStatus') ||
null;
        if (eventId && stageId && gateStatus === GateSta-
tus.PAUSE) {
          dispatch(StartStage.actions.updatePause());
        }
        }
```

Створення таймеру виконано у класі MainActivity додатку на мові програмування Java. Це було обрано через те, що на мові програмування Node.js неможливо забезпечити точний вимір часу через використання асинхронних функцій у циклі подій, які можуть надавати лише приблизний час простою таймера в системі (див. Лістинг 21).

Лістинг 21 — Реалізація таймеру в класі «MainActivity» на мові Java

```
private Timer timer = new Timer();
  @Override
  protected void onCreate(Bundle savedInstanceState) {
        SplashScreen.show(this);
        super.onCreate(savedInstanceState);
```

```
        this.timer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
          //Called at every 1000 milliseconds (1 second)
          WritableMap map = Arguments.createMap();
          map.putString("key", "tick");
          try {
               getReactInstanceManager().getCurrentReactCon-
text()
                    .getJSModule(DeviceEventManagerMod-
ule.RCTDeviceEventEmitter.class)
                    .emit("timer", map);

          } catch (Exception e){
               Log.e("ReactNative", "Caught Exception: " +
e.getMessage());
          }
        }
        },
        0,
        100);
  }
```

9. Класс LoginScreen є класом відображення сторінки для автентифікації користувача до системи, яка використовує валідацію введених даних на формі та направляє їх до запиту логінації (див. Лістинг 22)

Лістинг 22 — Функціонал сторінки LoginScreen

```
import isEmail from 'validator/lib/isEmail';
import isEmpty from 'validator/lib/isEmpty';
import  isStrongPassword  from  'validator/lib/isStrongPass-
word';
const pswdOptions = {
        minLength: 8,
        minLowercase: 1,
        minUppercase: 0,
        minNumbers: 1,
        minSymbols: 0,
        returnScore: false,
        pointsPerUnique: 0,
        pointsPerRepeat: 0.0,
        pointsForContainingLower: 0,
        pointsForContainingUpper: 0,
        pointsForContainingNumber: 0,
```

```
        pointsForContainingSymbol: 0,
};export default function LoginScreen(props: any) {
        const     [password,     setPassword]     =     us-
eState('qwerty123');
        const [email, setEmail] = useState('go@t.ua');
        const {t} = useTranslation();

        const isEmailEmpty = isEmpty(email);
        const isEmailValid = isEmail(email);
        const isPswdEmpty = isEmpty(password);
        const  isPswdValid  =  isStrongPassword(password,  psw-
dOptions);

        const loginButtonOnClick = () => {
        props.loginUser({
          userEmail: email,
          password: password,
        });
        };

        return (
        <View>
          <Appbar.Header             style={{backgroundColor:
'#fbc02d'}}>
               <Appbar.Content title={title} />
          </Appbar.Header>
          <LoginForm {...this.props} />
        </View>
        );
}
```

Клас HomeScreen представляє собою клас, який відповідає за відображення сторінки інформації про поточні змагання, що проводяться у компанії, в якій діє даний оператор. На цьому екрані можно переглянути поточні змагання, та приєднатися до стейджу одного з них (див. Лістинг 23).

Лістинг 23 — Функціонал сторінки HomeScreen

```
export default function HomeScreen(props) {
        const {t} = useTranslation();
        const dispatch = useDispatch();
        const EventEntity = useEntity('EventEntity');
        const {fetchCompanyEvents, bindOperToStage} = useAc-
tions('EventEntity');
        const {logoutUser} = useActions('Identity');
```

```
        const evs: IEventType = useSelector((state: any) =>
state[ENTITY.EVENT]);
        const user = useSelector((state: any) => state.iden-
tity?.user);
        const events: IDisplayEvent[] = useMemo(() => Even-
tEntity.getEventsList(),[EventEntity, evs],);
        const bindOperatorToStage = useCallback(
        (stage: IDisplayStage) => {
          stage.userId = user.userId;
          bindOperToStage(stage);
          dispatch(setFlagger('SelectedStage', stage));
          props.navigation.replace('StageScreen',    {screen:
'Stage', params: {stage},});},
        [bindOperToStage, user],
        );
        const handleStageClick = useCallback(
        (stage: any) => {
          if  (!stage.userId)  {dispatch(setFlagger('Confirm-
ConnectionModal', stage));}
          if (stage.userId === user.userId) {bindOperatorTo-
Stage(stage);}
          if (stage.userId && stage.userId !== user.userId)
{dispatch(setFlagger('ConfirmAnotherConnectedModal',
true));}
        },
        [dispatch, bindOperatorToStage, user.userId],
        );
        return (
        <View style={styles.container}>
          <Appbar.Header style={styles.header}>
               <Appbar.Content title={t('event-list')} />
               <HomeMenu data={user} />
          </Appbar.Header>
          <EventList
               events={events}
               fetchEvents={fetchCompanyEvents}
               onStageClick={handleStageClick}
          />

        </View>
        );
}
```

Клас Stage визначає відображення, необхідний для представлення контенту сторінки етапу, на якому оператор приєднався. У цьому класі виконуються різноманітні операції, які є важливими для ефективного проведення етапу змагань.

Ці операції включають в себе запуск етапу, управління атлетами (зміна порядку виступів, дискваліфікація) та установлення етапу на паузу.

Класс RefuseScreen визначає відображення, необхідний для представлення контенту сторінки дискваліфікованих атлетів, котрі вибули за бажанням оператора, або вони пропустили свій старт стейджу. Окрім відображення на даному екрані реалізована можливість повернення атлета до черги (див. Лістинг 24).

Лістинг 24 — Функціонал сторінки RefuseScreen

```
export default function RefuseScreen() {
       const stage = useSelector((state: any) => state.flag-
ger.SelectedStage);
       const {eventId, stageId, eventSlug, stageName} =
stage;
       const {fetchEventAthletes} = useActions('AthleteEnti-
ty');
       const {restoreAthlete} = useActions('StartStage');
       const di = useContext(ContainerContext);
       const navigator = di.resolve('navigator');
       useEffect(() => {fetchEventAthletes({eventSlug});},
[eventSlug]);
       const event: IEventType = useSelector((state: any)
=>state[ENTITY.EVENT].get(eventId),);
       const ats: IAthleteType = useSelector((state: any) =>
state[ENTITY.ATHLETE],);
       const startStage = di.resolve('redux').state().start-
Stage;
       const aID = startStage?.get('athleteId');
       const athletes = useMemo(() => {const athleteQueue =
di.resolve('queue');
       return athleteQueue(eventId, stageId, QueueFilter.RE-
FUSED);
       }, [eventId, stageId, event, ats, aID]);
       const handleMenuItemClick = useCallback((item: Opera-
tiveItem) => {
       restoreAthlete(item);
       }, []);
       return (
       <View style={styles.container}>
         <HeaderTabs
              tabName={''}
              iconName={'cancel'}
```

```
                startStage={startStage}
                stageId={stageId}
                eventId={eventId}
                stageName={stageName}
                navigator={navigator}
            />
            <RefuseList
                athletes={athletes}
                loadEventAthletes={()     =>     fetchEventAth-
letes({eventSlug})}
                onGoBackClick={handleMenuItemClick}
            />
        </View>
        );
}
```

Клас Navigator впроваджує функціонал бібліотеки React Navigation. Він не лише надає API для класів-компонентів, а також дозволяє здійснювати зміни у стані навігації з будь-якої частини програми. Такий підхід робить можливим більш гнучке та інтегроване управління навігацією в вашому додатку, що спрощує реалізацію складних взаємодій та навігаційних сценаріїв в взаємодії з іншими API застосунку. Клас надає функціонал зміни стану навігації або продвинути чи замінити нинішній екран чи інший (див. Лістинг 25).

Лістинг 25 — API класу Navigator.

```
export default class Navigator extends BaseContext {
    private _ref;
    constructor(opts: any) {
        super(opts);
        this._ref = createNavigationContainerRef();
        this._ref.addListener('state',
this.onRouteChange);
    }
    public get ref(): any {
        return this._ref;
    }
```

```
        private onRouteChange = () => {
            const {
                Identity,
                redux: {dispatch, state},
            } = this.di;
            const {identity} = state();
            if (identity?.user?.userId) {
                dispatch(Identity.actions.up-
dateIdentity());
            }
        };
        public navigate = (name: string, params?: object) =>
{
            if (this._ref.isReady()) {
                this._ref.navigate(name, params);
            }
        };
        public push = (name: string, params?: object) => {
            if (this._ref.isReady()) {
                this._ref.dispatch(StackActions.push(name,
params));
            }
        };
        public replace = (name: string, params?: object) =>
{
            if (this._ref.isReady()) {
                this._ref.dispatch(StackActions.re-
place(name, params));
            }    };}
```

3.3.3 Розробка інтерфейсу застосунку

Через те, що для розробки програмної частини інформаційної системи почергового старту на спортивних змаганнях буде виконана для мобільних телефонів з операційною системою Android, для реалізації інтерфейсу було використано JavaScript фреймворк ReactNative. Першою сторінкою, з яким взаємодіє оператор — це сторінка для автентифікації користувача (див. Рис. 3.4) Вона скалдається с двох полів та кнопки для логінації.

Перше поле призначене для логіну (username), друге — для паролю (пароль при введені прихован символами ‘*’).

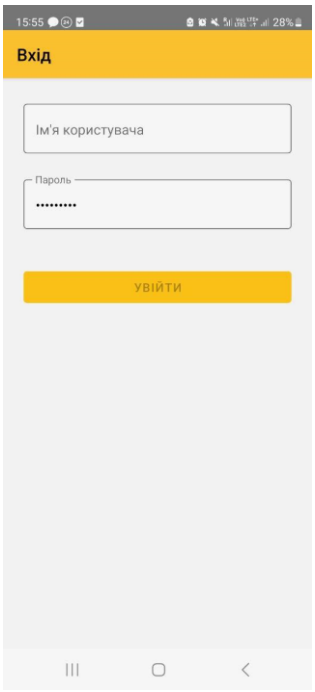

Рисунок 3.4 — Сторінка для автентифікації

У разі коректного логіну відкривається головна сторінка, в якій показані усі відкриті спортивні змагання компанії, до якої долучен оператор (див. Рис. 3.5).

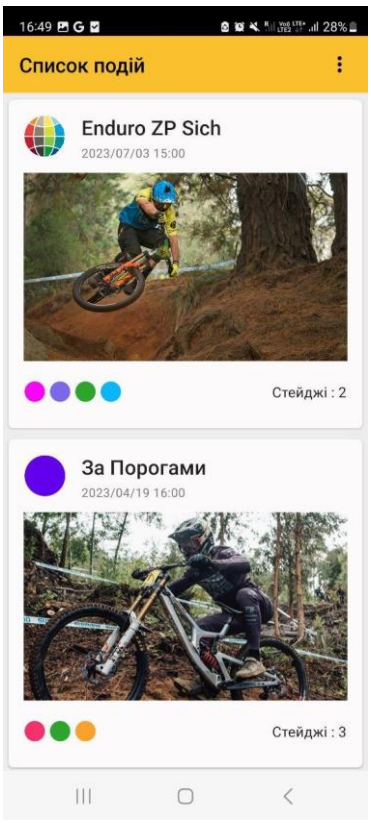

Рисунок 3.5 — Головна сторінка зі спортивними подіями компанії

При виборі стейджу івента, відкривається сторінка стейджу, де є черга виступаючих атлетів на цьому стейджі (див. Рис. 3.6). На кожному рядку з атлетом є його ім`я, номер атлета та час його виступу, якщо натиснути на іконку трьох точок, то відкриється модальне вікно, за допомогою якого можна керувати атлетом, а саме змінити його позицію в черзі виступу або дискваліфікувати на стейджі (див. Рис. 3.7).

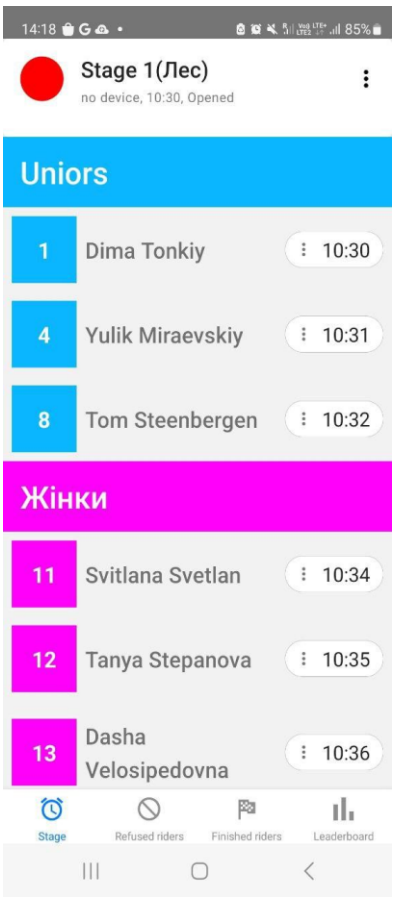

Рисунок 3.6 — Черга атлетів

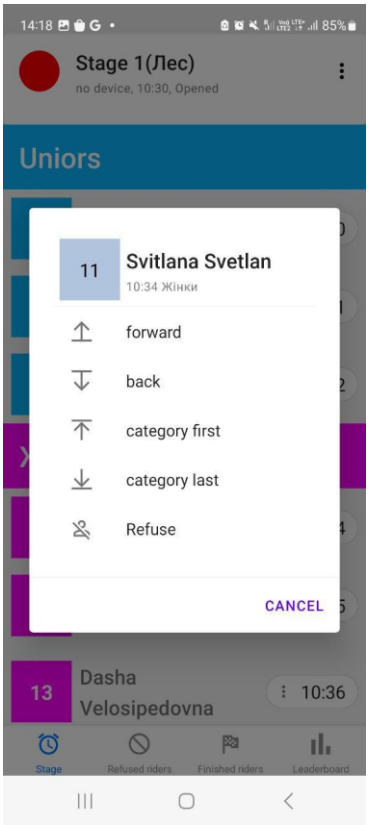

Рисунок 3.7 — Модальне вікно

Коли оператор підключається до відповідного стейджу на події, він має доступ до декількох екранів, крім основного екрану з чергою атлетів. Ці екрани включають:

1. Екран дискваліфікованих атлетів, на якому можна переглянути атлетів, що були дискваліфіковані, але можуть бути повернуті до черги для виступу (див. Рис. 3.8).

2. Екран фінішованих атлетів, на якому можна побачити інформацію про атлетів, які вже завершили свої виступи (див. Рис. 3.9).

3. Екран з таблицею лідерів для даного стейджу, на якому відображається перелік найкращих результатів атлетів, які виступають на цьому стейджі (див. Рис. 3.10).

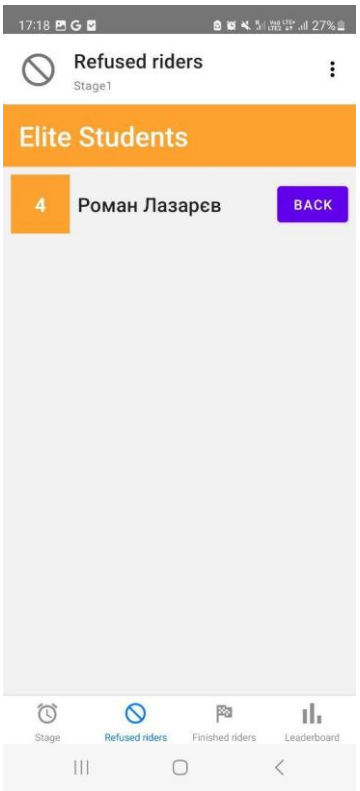

Рисунок 3.8 — Екран дискваліфікованих атлетів

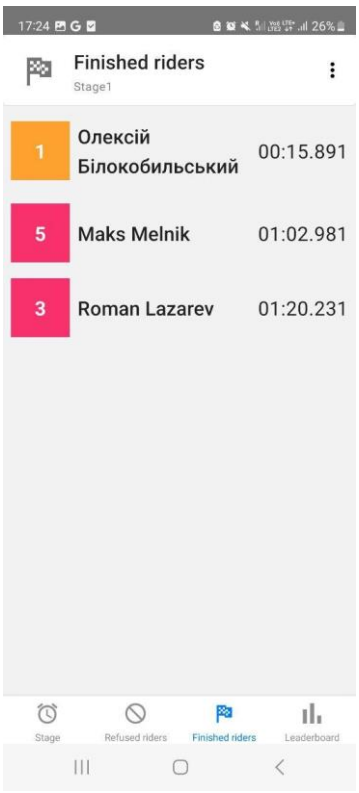

Рисунок 3.9 — Екран фінішованих атлетів

Рисунок 3.10 — Таблиця лідерів стейджу

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОЗРОБКИ СИСТЕМИ МОНІТОРИНГУ І КЕРУВАННЯ ДАТЧИКАМИ В СИСТЕМІ ОРГАНІЗАЦІЇ СПОРТИВНИХ ЗМАГАНЬ

4.1 Створення та оптимізація апаратного рішення системи проведення старту спортивних змагань

Для автоматизації проведення старту спортивних змагань необхідно створити комплексне апаратне рішення для ефективного зчитування результатів старту атлетів. Початковим етапом в цьому процесі було використання Arduino Uno в ролі логічного блока. Цей мікроконтролер виступає як посилка між датчиками та бізнес-логікою програмного рішення, забезпечуючи надійне збирання та передачу даних. Для використання в якості першого компоненту датчикового комплексу були обрані однопроменеві датчики Lightwell BI-20L (див. Рис. 4.1). Ці датчики здатні працювати на відстані до 10 метрів, що повністю відповідає вимогам для проведення послідовних стартів по ширині.

Рисунок 4.1 — Інфрочервоні датчики Lightwell BI-20L

Для забезпечення роботи Arduino необхідне постійне живлення або можливість подачі енергії через підключення по USB, де підключений пристрій виступає джерелом живлення. З метою поліпшення зручності використання платформи Arduino у поєднанні з портативним зовнішнім акумулятором (power bank) прийнято рішення створити контейнер, який виконуватиме

функцію захисту плати від впливів зовнішнього середовища. Крім того, у цьому контейнері будуть вбудовані роз'єми для зручного підключення телефону через USB (див. Рис. 4.2).

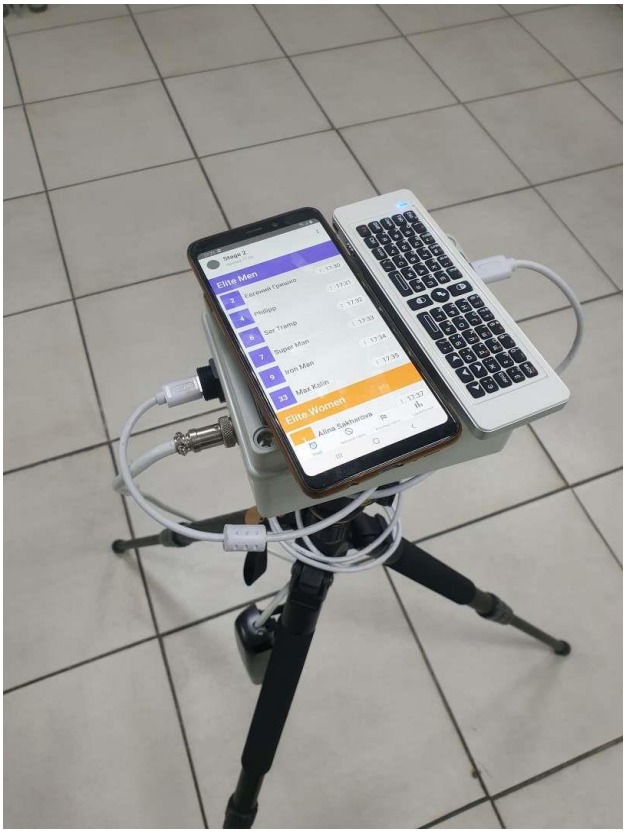

Рисунок 4.2 — Перша версія контейнеру для Arduino Uno

Під час випробувань цього рішення було виявлено значну кількість недоліків. Зокрема, виявлено, що встановлені датчики не відповідають вимогам для ефективного використання. Виявилось, що при розташуванні датчиків на відстані 1 метра один від одного не завжди відбувається зчитування сигналу. Крім того, датчики не здатні вловлювати сигнал при швидкості атлета понад 5 км/год, що є неприйнятним, особливо у видах спорту, наприклад, в маунтинбайку, де стартова швидкість може перевищувати 10 км/год. Також виникла значна трудність у тому, що обидва датчики були з'єднані з Arduino за допомогою проводів. Це становило ризик пошкодження конструкції під час

проведення старту в умовах нерівного ландшафту та наявності великої кількості атлетів.

Було вирішено замінити поточні датчики на інфрачервоні бар'єри Bar'єр TRX-100M/4CH TRINIX. Ці бар'єри мають граничну відстань виявлення руху на відкритому повітрі від 80 до 100 метрів, область огляду в горизонтальній площині ±90° і вертикальній площині ±5°. Це дозволить успішно встановити датчик з можливістю моніторингу значної території. Крім того, лише один із цих бар'єрів отримує живлення від Arduino за допомогою проводу, що зменшує ймовірність пошкодження воріт старту (див. Рис 4.3).

Рисунок 4.3 — Конструкція воріт з використанням TRX-100M/4CH

Внаслідок проведеного полевого тестування даної конструкції було встановлено, що датчики не придатні для використання, оскільки даний бар'єр не фіксує перетин при швидкості від 8 км/год. У результаті проведених змін, ІЧ-бар'єр було замінено фотоэлектрическим датчиком AC90-250V. Цей датчик

придатний для різноманітних застосувань, таких як уникнення перешкод у робочому процесі, підрахунок деталей в складальних ланцюгах та інші автоматизовані завдання (див. Рис 4.4).

Рисунок 4.4 — Датчик AC90-250V

Конструкція воріт також підверглась змінам. Для контейнера Arduino розроблено новий дизайн, в якому поточні датчики інтегровані безпосередньо у структуру воріт. Також доданий Bluetooth модуль для бездротового передавання даних, а також встановлено датчик заряду батареї для моніторингу рівня заряду акумулятора та передачі цих даних на мобільний пристрій. Ці зміни не лише поліпшили ергономіку конструкції, але й розширили її функціональні можливості, забезпечуючи зручність у використанні та моніторинг стану енергозабезпечення (див. Рис. 4.5). Оскільки нові датчики не вбудовані з

внутрішніми джерелами живлення, було потрібно створити контейнер для них, який не мав прямого підключення до Arduino.

Рисунок 4.5 — Третя ітерація конструкції старту воріт.

Під час тестів, які включали участь запрошених атлетів з маунтінбайкінгу, було виявлено декілька значущих проблем. Першою несправною ситуацією було те, що датчики, опромінені прямим сонячним світлом, залишалися в постійно активному режимі, змушуючи організаторів розміщувати ворота у тінь для забезпечення їх правильної роботи під час змагань. Крім того, через людський фактор оператор пропустив одного атлета поза його чергою, що призвело до призначення його результату іншому учаснику. Це також означало, що час старту цього атлета не було коректно зафіксовано в системі, що система розцінила як дискваліфікацію.

Для останньої, на даний момент ітерації, системи були використані рефлекторні датчики S3N-PR-2-B01-P RRX POL PNP L/D CABLE [17]. Конструкцію також було перероблено. Тепер вона складається з контейнера, де розташований новий датчик, а також був доданий циферблат, який відображає

номер поточного атлета (див. Рис. 4.6). Крім того, була переписана транспортна логіка в мобільному застосунку. Тепер, окрім команд на відкриття чи

закриття воріт, передається і номер атлета. Це робить простішим відстеження черги виступу для оператора.

Рисунок 4.6 — Конструкція контейнеру Arduino з середини.

На даний момент, остання версія воріт ще не була піддана повному тестуванню. Однак початкові тести, що включають аналіз розриву сигналу при високій швидкості, пройшли успішно.

4.2 Вирішення проблеми менеджменту атлетів в черзі, що має динамічний порядок

Більшість систем проведення спортивних змагань спрямовані на створення статичної черги виступів атлетів. Це означає, що якщо атлет з яких-небудь серйозних чи інших причин не з'являється на змагання у визначений момент часу, він автоматично дискваліфікується. Для аматорських змагань ці системи не лише дуже витратні у використанні, але й непрактичні через велику кількість людського фактора, який може впливати на відсутність атлета. Дана система старту була орієнтована на проведення змагань, котра має динамічну чергу виступів.

Основне завдання полягало в розрахунку тривалості виступу під час зміни порядку в черзі. Існує кілька факторів, які впливають на розрахунок старту відповідного атлета : час початку виступу, інтервали між виступами атлета, паузи між виступами атлетів різних категорій, порядок атлета, час витрачений на фальшстарти попередніх учасників. Деякі з цих факторів виявилися змінними, що ускладнювало процес розрахунку. Першим рішенням було збереження часу старту в базі даних сервера та внесення змін при необхідності. Однак цей підхід виявився неефективним через те, що зміни значень для одного атлета тягнули за собою зміну значень для всіх атлетів. Це призводило до великої кількості запитів на синхронізацію з сервером, а в умовах нестабільного інтернет-з'єднання, спричиняло повну десинхронізацію.

Другим способом рішення було зробити час старту атлета розрахунковим значенням, що з кількох значень. Першим значенням був дійсний розрахунок порядку виступу атлета по всій черзі.

$$cA = 10000 * cO + aO, \tag{4.1}$$

де cO — це порядок виступу всієї категорії, та aO — це порядок виступу атлета в даній категорії. Значення є дійсним, і якщо ми змістимо атлета на іншу позицію, то він візьме значення верхнього атлета та опловинить його. (4.2)

$$cA_0 = 10000 * cO + (aO_1 / 2). \tag{4.2}$$

Далі, після сортування всіх атлетів за заданим параметром здійснюється первинний розрахунок часу виступу всіх атлетів. Розрахунок старта атлета призводиться шляхом суми значення затримок перед стартом атлета, суми фальш стартів та затримок стэйджа (4.3). Де M_i — це кількість хвилин, витрачених на фальш старти данного атлета, S — це значення, котре було розраховано для попереднього атлета, $f_x(a_i)$ — це функція розрахунку часу затримки для відповідного атлета (4.4, 4.5, 4.6), D — значення паузи стейджа.

$$\sum^{n}_{i=1} = M_i + S_{i-1} + f_x(a_i) + D. \quad (4.3)$$

Деякі спортсмени мають відмінний спосіб обчислення своєї затримки. Той атлет, який виступає першим в черзі, взагалі не має затримки і готовий виступати негайно (4.4).

$$f_0(a_i) = 0. \quad (4.4)$$

Для звичайних атлетів застосовується стандартна затримка, установлена для всіх учасників даного змагання (4.5).

$$f_1(a_i) = aD_i . \quad (4.5)$$

Щодо атлетів, які виступають першими у своїй категорії, використовується затримка між категоріями, яка пов'язана з цим конкретним змаганням (4.6).

$$f_1(a_i) = aD_i . \quad (4.6)$$

Отже, для досягнення синхронізації з сервером тепер необхідно передавати унікальне значення порядку одного атлета разом із запитом. Програми клієнта тепер можуть витягувати час старту кожного атлета, використовуючи розроблені вище формули, та забезпечувати унікальні ідентифікатори для кожного спортсмена. Це дозволить серверу точно визначити, який саме атлет почав гонку у конкретний момент часу, забезпечуючи надійну синхронізацію подій на стороні клієнта та сервера.

ВИСНОВКИ

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

1. Визначена предметна область та проаналізована існуюча проблематика. Під час аналізу проблеми та взаємодії з атлетами виявлено, що більшість локальних змагань досі використовують традиційні методи фіксації та обчислення результатів атлетів, що може призвести до значної похибки у визначенні місць. Були розглянуті альтернативні системи для автоматизації проведення спортивних змагань, і вивчено їхні механізми старту. Виявилося, що такі системи є досить вартісними на ринку, і більшість надійних рішень працює лише зі статично заданою чергою. Отже, розробка автоматизованої системи моніторингу датчиків для старту спортивних змагань з динамічною зміною черги залишається невивченою областю.

2. Були розглянуті різні платформи та технології для створення системи. При вивченні переваг та недоліків перевага була віддана платформі Android. Оптимальними засобами реалізації програмного застосунку були обрані: JavaScript, React Native, Redux, VS Code, Android Studio, Arduino IDE.Для реалізації апаратної частини була використана платформа Arduino UNO.

3. Було розроблено бізнес-обгрунтування та вимоги до програмного продукту, після чого спроектовані та розроблені інформаційна система для проведення старту спортивних змагань. Також описано її інтерфейс та найважливіші модулі.

4. Було проведення дослідження з оптимізації апаратної зборки системи старту атлетів. Також була вирішена проблема перерахунку часу виступу атлетів в рамках динамічної черги.

Таким чином, розроблена інформаційна система відповідає всім вимогам, які були висунуті до програмного застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. About SQLite. sqlite.org веб-сайт. URL: https://www.sqlite.org/about.html (дата звернення: 20.11.2023).

2. Awilix. github.com : веб-сайт. URL: https://github.com/jeffijoe/awilix#awilix (дата звернення: 20.11.2023).

3. ChronoTrack Hardware. chronotrack.com : веб-сайт. URL: https://chronotrack.com/hardware/ (дата звернення: 20.11.2023).

4. Creational Patterns, Singleton. refactoring.guru : веб-сайт URL: https://refactoring.guru/design-patterns/singleton (дата звернення: 20.11.2023).

5. Description of the database normalization basics. learn.microsoft.com : веб-сайт. URL: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description (дата звернення: 20.11.2023).

6. Effectivity of Sports Timing RFID System, Field Study. ieeexplore.ieee.org : веб-сайт. URL: https://ieeexplore.ieee.org/abstract/document/889210 (дата звернення: 21.11.2023).

7. Evaluating the performance of Android based Cross-Platform App Development Frameworks. dl.acm.org : веб-сайт. URL: https://dl.acm.org/doi/abs/10.1145/3442555.3442561 (дата звернення: 21.11.2023).

8. Flutter for Android developers. docs.flutter.dev веб-сайт. URL: https://docs.flutter.dev/get-started/flutter-for/android-devs (дата звернення: 21.11.2023).

9. Getting Started with Arduino IDE 2. docs.arduino.cc: веб-сайт URL: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2 (дата звернення: 21.11.2023).

10. Getting Started with Redux. redux.js.org веб-сайт. URL: https://redux.js.org/introduction/getting-started (дата звернення: 21.11.2023).

11. Inversion of Control Containers and the Dependency Injection pattern. martinfowler.com: веб-сайт URL: https://www.martinfowler.com/articles/injection.html (дата звернення: 22.11.2023).

12. Meet Android Studio. веб-сайт. developer.android.com. веб-сайт. URL: https://developer.android.com/studio/intro (дата звернення: 22.11.2023).

13. Performance Comparison Between React Native And Flutter. diva-portal.org: веб-сайт. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1349917&dswid=-1404 (дата звернення: 22.11.2023).

14. RACE RESULT SYSTEM Data Sheet. raceresult.com : веб-сайт. URL: https://www.raceresult.com/en/support/document.php?name=System_Datasheet (дата звернення: 22.11.2023).

15. React Native Introduction. reactnative.dev : веб-сайт. URL: https://reactnative.dev/docs/getting-started (дата звернення: 22.11.2023).

16. React Native Introduction reactnative.dev. веб-сайт. URL: https://reactnative.dev/docs/getting-started (дата звернення: 22.11.2023).

17. S3N-PR-2-B01-P RRX POL PNP L/D CABLE Miniature photoelectric sensor. datasensing.com: веб-сайт URL: https://www.datasensing.com/prodotti/cubic-sensors/95B010592/s3n-pr-2-b01-p-rrx--pol-pnp-l-d-cable (дата звернення: 22.11.2023).

18. Sprint time differences between single and dual-beam timing systems. researchgate : вебсайт. URL: https://www.researchgate.net/publication/260218925_Sprint_Time_Differences_Between_Single-_and_Dual-Beam_Timing_Systems (дата звернення: 23.11.2023).

19. Sublime Text Introduction. docs.sublimetext.io : веб-сайт. URL: https://docs.sublimetext.io/guide/ (дата звернення: 23.11.2023).

20. Validity and reliability of GPS and LPS for measuring distances covered and sprint mechanical properties in team sports. journals.plos.org : веб-сайт

URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0192708 (дата звернення: 23.11.2023).

21. Validity of Single-Beam Timing Lights at Different Heights. journals.lww.com : веб-сайт. URL: https://journals.lww.com/nsca-jscr/FullText/2017/07000/Validity_of_Single_Beam_Timing_Lights_at_Different.29.aspx (дата звернення: 23.11.2023).

22. VSCode vs Sublime Text: Which Is the Best Option? tms-outsource.com : веб-сайт. URL: https://tms-outsource.com/blog/posts/vscode-vs-sublime-text/ (дата звернення: 24.11.2023).

23. Why did we build Visual Studio Code? code.visualstudio.com : веб-сайт. URL: https://code.visualstudio.com/docs/editor/whyvscode (дата звернення: 24.11.2023).

24. Лазарєв Р.О., Скрипник І.А., Створення автоматизованої системи проведення старту спортивних змагань : матеріали Міжнародної науково-практичної конференції 25-26 травня 2023 року. С.461-463.

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Лазарєв Роман Олександрович, студент 2 курсу, денної форми навчання, Інженерного навчально-наукового інституту ім. Ю.М. Потебні ЗНУ, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти p314154265@gmail.com,

- підтверджую, що написана мною кваліфікаційна робота на тему: «**Автоматизована система моніторингу і керування датчиками в комп'ютерній системі 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 30.11.2023 Підпис _____ Лазарєв Роман Олександрович

(студент)

Дата 30.11.2023 Підпис _____ Скрипник Ірина Анатоліївна

(науковий керівник)