Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

# КВАЛІФІКАЦІЙНА РОБОТА
# Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

**РОЗРОБКА ЗАСТОСУНКУ ДОСЛІДЖЕННЯ ТА ПРОГРЕСІЇ СПОРТСМЕНІВ ШВИДКІСНОГО ПІДВОДНОГО ПЛАВАННЯ**
(тема)


Виконав:
студент 4 курсу, групи ІТІНФ-20-1
Кондратова А.Ю.
(прізвище, ініціали)


Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник асист. Кобилін І.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кондратовій Анні Юріївні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку дослідження та прогресії спортсменів швидкісного підводного плавання

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 26 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі, мова гіпертекстової розмітки HTML5, каскадні таблиці стилів CSS, мова програмування JavaScript, об'єктно реляційна база даних PostgreSQL, фреймворк Nest.js, бібліотека React, середовище розробки Visual Studio.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз існуючих застосунків для спортсменів.
2. Виявлення вимог до застосунку.
3. Моделювання архітектури застосунку та бази даних.
4. Створення вебзастосунку для дослідження та прогресії спортсменів швидкісного підводного плавання.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми дослідження та прогресії спортсменів, постановка задачі, діаграми бізнес-процесів, схема бази даних, програмна реалізація.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

| Найменування розділу | Консультант (посада, прізвище, ім'я, по батькові) | Позначка консультанта про виконання розділу | |
|---|---|---|---|
| | | підпис | дата |
| | | | |

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Отримання завдання на кваліфікаційну роботу | 08.04.2024 | |
| 2 | Аналіз завдання, підбір літератури | 09.04.24-13.04.24 | |
| 3 | Аналіз літератури з досліджуваної проблеми | 14.04.24-19.04.24 | |
| 4 | Аналіз технічних засобів | 20.04.24-25.04.24 | |
| 5 | Моделювання бази даних та архітектури застосунку | 26.04.24-08.05.24 | |
| 6 | Програмна реалізація | 09.05.24-23.05.24 | |
| 7 | Оформлення пояснювальної записки | 24.05.24-27.05.24 | |
| 8 | Перевірка на плагіат | 28.05.24 | |
| 9 | Рецензування | 29.05.24 | |
| 10 | Підготовка презентації та доповіді | 30.05.24-04.06.24 | |
| 11 | Занесення роботи в електронний архів | 05.06.24 | |
| 12 | Попередній захист кваліфікаційної роботи | 05.06.24 | |

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ _____асист. Кобилін І.О._____
(підпис) (посада, прізвище, ініціали)

# РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 66 с., 5 табл., 27 рис., 30 джерел.

ВЕБЗАСТОСУНОК, REST API, SPA, HTML5, CSS, JAVASCRIPT, REACT, NODE.JS, NEST.JS, РЕЛЯЦІЙНА БАЗА ДАНИХ, POSTGRESQL.

Об’єктом роботи є створення клієнт-серверного вебзастосунку для дослідження прогресу спортсменів підводного плавання та організації змагань. Для розробки системи буде використано JavaScript, Node.Js, Nest.Js, Prisma ORM, React, Reacct Router, Redux та PostgreSQL.

Мета даної роботи полягає у створенні зручного вебсервісу для відстеження прогресу певними групами користувачів: спортсмени, судді, тренери та автоматизації процесу організації змагань, внесення результатів, забезпечуючи абсолютну прозорість процедури.

Застосунок забезпечує повне управління змаганням від створення та подачі заявок до автоматичного визначення призерів, отримання актуальної статистики спортсменів та команди, забезпечення зручного порівняння та автоматизацію паперової роботи для суддів.

WEB APPLICATION, REST API, SPA, HTML5, CSS, JAVASCRIPT, REACT, NODE.JS, NEST.JS, RELATIONAL DATABASE, POSTGRESQL.

The object of this work is to create a client-server web application for tracking the progress of underwater swimmers and organizing competitions. The system will be developed with JavaScript, Node.js, Nest.js, Prisma ORM, React, React Router, Redux, and PostgreSQL.

The goal of this work is to create a convenient web service for tracking progress by specific user groups: athletes, judges, and coaches. Additionally, it aims to automate the competition organization process, entry of results, ensuring complete transparency of the process.

The application provides complete management of the competition, from creation and application submission to automatic determination of winners, obtaining up-to-date statistics for athletes and teams, providing easy comparison, and automating paperwork for judges.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База Даних

ПЗ – Програмне Забезпечення

SPA – Single-Page Application

US – User Story

BPMN – Business Process Model and Notation

ORM – Object-Relational Mapping

HTTP – HyperText Transfer Protocol

REST API – Representational State Transfer

CORS – Cross-Origin Resource Sharing

JWT – JSON Web Token

CRUD (Create,read, update, delete)

# ВСТУП

Під час будь-якого виду діяльності істотно важливо відстежувати прогрес, оскільки не помічаючи прогресу неможливо коригувати плани для покращення своїх результатів або знаходження кращого шляху для досягнення своєї мети. Аналіз цифр на папері є незручним та займає багато часу. Також варто пам’ятати, що інколи прогрес може відбуватися, але він може бути недостатньо великим, для того щоб побачити його неозброєним оком.

Важливо відстежувати прогрес у спорті, адже кожна мить може стати вирішальною. Від результату одного спортсмена залежить успіх усієї команди. Багато тренерів зізнаються, що досить складно та виснажливо одночасно аналізувати велику кількість результатів, відстежуючи прогрес усіх спортсменів.

Також ключовим аспектом у відстеженні прогресу є візуалізація вона допомагає не лише, легше сприймати результат та аналізувати прогрес, а й є потужним мотиваційним інструментом для спортсмена, що надихає.

Актуальність роботи полягає у тому, що сьогодні досить велика кількість людей займаються спортивним підводним плавання, та є досить вибагливим видом спорту, де на рішення впливають мілісекунди, а успіх одного спортсмена може визначити перемогу всієї команди. Тому мати якісний застосунок у цій дисципліні стає істотно важливим. Запропонований застосунок стане у нагоді не лише тренерам та спортсменам для полегшення процесу відстеження та дослідження результатів, аналізу результатів та порівняння успіхів декількох спортсменів, а й суддям, оскільки він допоможе планувати змагання, отримувати заявки, автоматично підбивати командний залік, визначати переможців та значно спростить паперову роботу судді. А також забезпечить абсолютну прозорість процедури, що є надзвичайно важливим на шляху досягнення справедливих та чесних результатів.

# 1 ОГЛЯД СТАНУ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАДАЧІ

Говорячи про відстеження прогресу спортсменів у великому спорті, зазвичай маємо справу з великим обсягом даних, які є критично важливими для аналітики, але якщо ці данні зовсім неструктуровані, то досягнення мети (аналіз успіхів спортсмена та визначення його прогресу) стає складною справою, оскільки не має можливості швидко відсортувати дані та встановити їх у хронологічному порядку.

У контексті вирішення даної проблеми, є ще один критично важливий аспект, окрім хорошої візуалізації цих даних, потрібно реалізувати доступний та легкий у користуванні застосунок, який буде у нагоді при вирішенні будь-яких задач з процесу організації змагань та внесення їх результатів. Тому збір вимог також стає невід'ємним фактором успішності майбутнього сервісу.

## 1.1 Часові ряди як спосіб представлення даних у дослідженні прогресу спортсменів швидкісного підводного плавання та їх візуалізація

Часовий ряд – це ряд точок даних, які розташовані у хронологічному порядку. Тобто часовий ряд являє собою послідовність даних, які взяті у певні проміжках часу, що йдуть одна за одною. Прикладами часових рядів є температурні ряди – дані про температуру у певному регіоні, які фіксувалися щогодини, фінансові ряди – динаміка цін на акції певної компанії протягом певного періоду часу, спортивні дані – результати тренувань або змагань спортсменів за кожен день чи тиждень тощо.

Часові ряди можуть бути представлені у вигляді таблиці, діаграми, графіку. У таблиці кожен рядок відповідає певному моменту часу, а стовбці – даним спостереження. Діаграми здебільшого використовуються для відображення різних характеристик часових рядів. На графіках зазвичай вісь $X$ відображає час, а вісь $Y$ – дані спостережень (рис. 1.1).

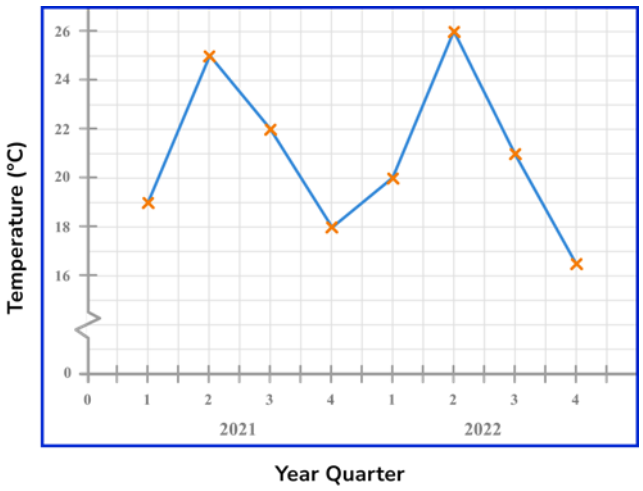


Рисунок 1.1 – Графік часового ряду

У контексті вирішення даної проблеми використання графіків є оптимальним рішенням, оскільки вони є легкими для візуального сприйняття. Але варто пам’ятати що на результат вимірювання, можуть впливати випадкові фактори, наприклад, погане самопочуття спортсмена або деякі технічні проблеми, які при аналізі часового ряду та прогресу можуть ввести в оману. Більш того досвідчені спортсмени, що давно займаються цією діяльність, мають надто багато даних, що може спричинити погане візуальне сприйняття графіку прогресу, тобто графіки будуть досить зашумлені, тому для вирішення цієї проблеми потрібно почати з визначення природи даних, а саме визначення трендів. Оскільки вони дозволять зосередитися на основних змінах у динаміці даних та уникнути дії випадкових факторів, які не є важливими.

Тренд – це довгострокова тенденція зростання, спадання або стабільності в часовому ряді. Він відображає основні зміни або тенденції в даних, ігноруючи тимчасові коливання та зашумленість, тобто ігнорує вплив дії випадкових факторів та згладжує данні. Зазвичай він зображений у формі лінії на графіку часового ряду, основною метою якого є показ загальної тенденції, тобто спадання чи зростання значень (рис. 1.2).

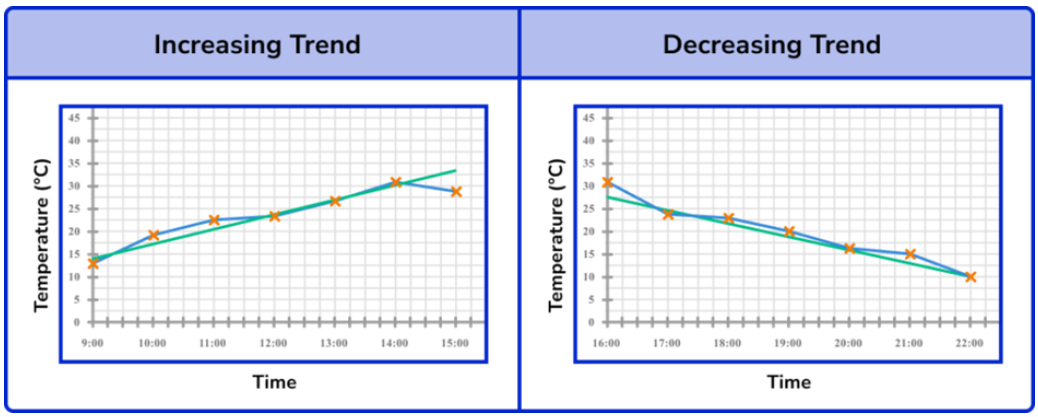


Рисунок 1.2 – Графік часового ряду зі спадаючим та зростаючим трендом

Існує багато способів визначення тренду: графічний метод, метод півсередніх, метод рухомої середньої, метод найменших квадратів.

Для вирішення даної проблеми оптимально застосовувати метод рухомої середньої. Оскільки він є досить простим у реалізації та забезпечує усі потреби. Існує два популярних підходи: проста рухома середня або експоненціальна рухома середня. Використання простої рухомої середньої є більш доречним у цьому випадку, тому що її застосовують для довгострокових аналізів, а саме коли цікавить загальний тренд без великих реакцій на кожну зміну та у випадках, коли тренд досить стабільний або рідко змінюється. Проста рухома середня обчислюється як середнє значення певної кількості попередніх точок у часовому ряді (1.1).

$$S_t = \frac{1}{m}\sum_{i=0}^{m-1} x_{t-i}, \tag{1.1}$$

де $S_t$ – значення рухомої середньої в момент часу t;

$x_{t-i}$ – значення часового ряду в момент часу $t - i$;

$m$ – період рухомої середньої.

## 1.2 Способи вимог у розробці програмного забезпечення

У розробці програмного забезпечення, процес збору вимог є одним з ключових етапів, оскільки неправильно зібрані вимоги можуть загальмувати процес створення програмного забезпечення та призвести до багатьох проблем, тому даний етап стає важливим для успіху майбутнього проєкту.

Основною метою збору вимог до програмного забезпечення є повна відповідність програмного рішення потребам зацікавлених сторін і кінцевих користувачів. Іншими словами, задача полягає у тому, щоб повністю з'ясувати усі потреби, цілі та подати їх у зрозумілій стислій формі. Вимоги мають бути повними, точними, недвозначними, а також легкими у тестуванні.

Існує досить багато технік для збору вимог: інтерв'ю, фокус-групи, опитування, аналіз конкурентів, спостереження тощо.

Інтерв'ю є найпопулярнішим методом збору вимог. Воно включає особисті обговорення із зацікавленими сторонами та кінцевими користувачами для збору інформації про їхні потреби та вимоги.

Опитування також є одним з найпоширеніших методом збору вимог, що передбачає створення анкет, які будуть містити найважливіші запитання зацікавленим сторонам і кінцевим користувачам у паперовій або електронній формі. Зазвичай опитування використовуються для швидкого збору великої кількості даних і для отримання відгуків від великої кількості людей.

Фокус-групи – спосіб збору вимог, що включає групу зацікавлених сторін і кінцевих користувачів, які разом обговорюють проєкт. Зазвичай використовуються для збору різноманітних думок та ідей, а також для створення нових ідей.

Спостереження – метод збору вимог, що передбачає спостереження за зацікавленими сторонами та кінцевими користувачами під час виконання ними своїх завдань, документуючи їхні потреби та вимог. Цей метод є

корисним для визначення вимог, про які зацікавлені сторони можуть не знати, а також для кращого розуміння предметної області.

У даному випадку, застосування техніки інтерв'ю є оптимальним способом збору вимог, тому що є можливість забезпечити комунікацію з невеликою кількістю кінцевих користувачів та час для повної реалізації проекту є досить обмеженим. Після чого будуть створені та досліджені групи користувачів та їх вимоги, що забезпечить краще розуміння вимог майбутнього застосунку.

## 1.3 Огляд існуючих застосунків для відстеження прогресу

Спорт завжди був популярним заняттям, тому зараз існує багато застосунків, які допомагають спортсменам відстежувати та аналізувати їх успіхи, стежити за режимом харчування, вести журнал тренувань тощо. Одними з найпопулярніших та найзручніших застосунків для спортсменів вважаються: MyFitnessPal, Endomondo, Strava.

MyFitnessPal – застосунок, який допомагає спортсмену відстежувати харчування та фізичну активність упродовж дня. MyFitnessPal є досить легким у користуванні, користувач може кожного дня вести щоденний журнал, тобто фіксувати прийоми їжі, автоматично вираховувати кількість спожитих калорій та аналізувати чи достатньо цього для досягнення поставленої мети. Також користувач має змогу керувати цілями та обмеженнями у додатку, тобто встановлювати бажану кількість спожитих калорій, макронутрієнтів (білків, жирів, вуглеводів) або інших показників. Програма буде допомагати відстежувати його прогрес щоденно. MyFitnessPal дозволяє інтегруватися з іншими пристроями такими як Fitbit або Garmin, які дозволяють вираховувати кількість спалених калорій за день та визначити чи відповідає кількість спалених калорій вашому харчуванню. Основним недоліком цього додатку є те, що він здебільшого орієнтований на

непрофесійних спортсменів – людей які просто хочуть вести здоровий спосіб життя та стежити за потребами власного тіла. Також відсутня змога фіксувати власні результати у конкретному виді діяльності, наприклад, спортсмен швидкісного підводного плавання не має змоги зафіксувати час свого запливу для подальшого аналізу та відстеження прогресу.

Endomondo є також одним з найпопулярніших додатків для спортсменів, перевагами цього додатку є змога краще фіксувати вид активності, на відміну від додатку MyFitnessPal, оскільки ти можеш обирати яким саме видом активності ти займаєшся та фіксувати час. Після завершення активності програма надає детальну статистику про ваші тренування, таку як середня швидкість, тривалість, кількість калорій тощо. Але так само як і попередній додаток, він не покриває усі потреби спортсменів та тренерів підводного плавання через специфіку цього виду спорту, немає можливості вказувати який саме вид запливу та дистанцію виконував спортсмен, також неможливо ділитися результатами з тренером, що робить дослідження прогресу незручним.

Усі існуючі сервіси мають певні переваги та недоліки, але повністю не забезпечують потреби спортсменів та тренерів підводного плавання, тому відстеження прогресу стає доволі важким завданням як і для тренера, так і для самого спортсмена.

## 1.4 Актуальність роботи

Швидкісне підводне плавання є досить популярним видом спорту в Україні для різних вікових категорій. Кожен спортсмен та тренер хоче щоб його підопічні ставали з кожним днем все краще та досягали нових висот, але без якісного застосунку для відстеження прогресу спортсменів – це завдання стає доволі складним.

На даний мамонт не існує застосунку який би міг покрити більшість потреб спортсменів, суддів та тренерів, тобто допомагав відстежувати прогрес, забезпечувати легку подачу заявки на змагання, організовувати змагання, автоматизувати паперово роботу та забезпечувати прозорість результатів змагань. Тренери повинні мати змогу з легкістю проаналізувати слабкі і сильні сторони спортсменів, виявити найуразливіших учасників команди, щоб покращити їх результати та результати усієї команди.

Отже, створення сучасного застосунку, який дозволить забезпечити ключові потреби кінцевих користувачів є необхідним. Після завершення розробки ми отримаємо не лише зручний та легкий у користування сервіс, а й потужний інструмент, який буде сприяти мотивації та само моніторингу спортсменів, буде допомагати підвищувати ефективність тренувань та проведення досліджень.

## 1.5 Постановка задачі

Розробка застосунку дослідження та прогресії спортсменів швидкісного підводного плавання є актуальним та важливим завданням для сприяння розвитку спортсменів цього виду спорту.

Об’єктом роботи є створення клієнт-серверного вебзастосунку для дослідження прогресу спортсменів підводного плавання та організації змагань. Для розробки системи буде використано JavaScript, Node.Js, Nest.Js, Prisma ORM, React, Reacct Router, Redux та PostgreSQL.

Мета даної роботи полягає у створенні зручного вебсервісу для відстеження прогресу певними групами користувачів: спортсмени, судді, тренери та автоматизації процесу організації змагань, внесення результатів, забезпечуючи абсолютну прозорість процедури.

Для досягнення мети необхідно вирішити такі завдання:

– провести аналіз предметної області;

– зібрати вимоги майбутнього застосунку;

– систематизувати вимоги та провести їх аналіз;

– розробити застосунок, що забезпечує організацію змагань та повне керування ним;

– додати функціонал, що дозволяє відстежувати прогрес спортсменів та команди згідно з результатами змагань;

– додати функціонал, що дозволить автоматизувати деяку паперову роботу: процеси присвоєння звань, підрахунку командного заліку та виявлення переможців;

– додати функціонал, що забезпечує повну прозорість змагань.

# 2 АНАЛІЗ ВИМОГ ТА БІЗНЕС-ПРОЦЕСІВ ПРЕДМЕТНОЇ ОБЛАСТІ

Вимоги – це специфікація того, що має бути реалізовано, в них описано поведінка системи, властивості системи або її атрибути.

Аналіз вимог відіграє важливу роль у створенні якісного ПЗ, оскільки вони дають підґрунтя для подальшої реалізації продукту. Аналіз вимог допомагає визначити подальшу стратегію, методи розробки та технології, які будуть використовуватися. Також мета дослідження вимог – переконатися, що усі зібрані та описані вимоги повністю забезпечують потреби зацікавлених сторін і призведе до результату, який вони очікують. Якісно зібрані вимоги – якісно створений продукт, тому що без зібрання та дослідження вимог розробникам було б важко зрозуміти що саме потрібно створити, як це має функціонувати та як це протестувати.

## 2.1 Збір та аналіз вимог груп кінцевих користувачів

Для виявлення вимог було застосовано одну з популярних технік збору вимог – інтерв’ю, суть якої полягає у структуровані розмови з представником певної групи користувачів, де було поставлено заздалегідь продумані запитання та зафіксовано їхні відповіді. Усіх користувачів було поділено на три групи: тренери, спортсмени та судді. Кожна група має певні інтереси, труднощі та потреби.

Для кращого розуміння потреб зацікавлених сторін було створено портрети груп користувачів. Портрет групи користувачів – це детальний опис типових представників цільової аудиторії, який створюється на основі зібраної інформації під час етапу виявлення вимог. Цей інструмент допоможе розробникам, аналітикам краще розуміти потреби, пріоритети користувачів, що дозволить створити продукт, який повністю відповідає їх очікуванням.

Судді – люди жіночої та чоловічої статі, середньої та старшої вікової категорії, які мають вищу освіту, зазвичай зі спортивно-фізичною або спортивно-технічною спрямованістю. Для цієї групи користувачів важливо мати швидкий та легкий в управлінні застосунок, який допоможе зменшити кількість паперової роботи, швидко створювати та керувати змаганнями (табл. 2.1).

Таблиця 2.1 – Портрет групи користувачів Судді

| **Критерій** | **Опис** |
|---|---|
| Група користувачів | Судді |
| Опис групи | Вік: 25-60 років;<br>Стать: жіноча, чоловіча;<br>Освіта: вища освіта;<br>Висококваліфіковані спеціалісти, відповідальні за оцінку виступів плавців під час змагань. |
| Що важливо для групи | – швидкий та легкий доступ до застосунку;<br>– не гаяти час на паперову роботу;<br>– прозорість результатів. |
| Проблеми | – відсутність сервісу для швидкого планування змагань;<br>– самостійно вираховувати командний залік та результати;<br>– необхідність створення протоколів змагань;<br>– ризик втратити дані про результати змагань. |
| Потреби | – швидко вносити данні про результати;<br>– управління змаганнями;<br>– генерування протоколів змагань;<br>– автоматичне вирахування результатів та присвоювати звання. |

Тренери – люди жіночої та чоловічої статі, середньої та старшої вікової категорії, які мають вищу спортивну освіту. Для цієї групи користувачі важливо мати застосунок, який допоможе відстежувати успіх спортсменів, виявляти їх прогрес, слідкувати за новими змаганнями та швидко подавати заявки, переглядати результати змагань та аналізувати результати опонентів (табл. 2.2).

Таблиця 2.2 – Портрет групи користувачів Тренери

| **Критерій** | **Опис** |
|---|---|
| **1** | **2** |
| Група користувачів | Тренери |
| Опис групи | Вік: 25-60 років;<br>Стать: жіноча, чоловіча;<br>Освіта: вища освіта;<br>Висококваліфіковані спеціалісти, відповідальні за підготовку спортсменів до змагань. |
| Що важливо для групи | – швидкий та легкий доступ до застосунку;<br>– прозорість результатів;<br>– не витрачати багато часу на аналіз результатів. |
| Проблеми | – гаяння часу під час збору та систематизації результатів змагань команди та спортсмена;<br>– складність опрацювання великої кількості інформації;<br>– пропускання змагань через відсутність актуальної інформації;<br>– ризик втратити дані про результати змагань. |

Продовження таблиці 2.2

| 1 | 2 |
|---|---|
| Потреби | – швидка подача заявки на змагання;<br>– отримання даних у зручній формі про прогрес спортсмена;<br>– отримання даних про прогрес команди.<br>– легке виявлення слабких та сильних учасників;<br>– можливість порівняння спортсменів, команди;<br>– швидкий доступ до змагань які відбулися або плануються. |

Спортсмени – люди жіночої та чоловічої статі, молодшої та середньої вікової категорії, які активно займаються спортом, беруть участь у змаганнях та прагнуть покращити їх результати (табл. 2.3).

Таблиця 2.3 – Портрет групи користувачів Спортсмени

| **Критерій** | **Опис** |
|---|---|
| **1** | **2** |
| Група користувачів | Спортсмени |
| Опис групи | Вік: 10-40 років;<br>Стать: жіноча, чоловіча;<br>Учасники змагань. |
| Що важливо для групи | – швидкий та легкий доступ до застосунку;<br>– прозорість результатів;<br>– не витрачати багато часу на аналіз результатів. |

Продовження таблиці 2.3

| 1 | 2 |
|---|---|
| Проблеми | – важко відстежувати прогрес та вести статистику;<br>– важко порівняти себе з іншими спортсменами. |
| Потреби | – отримання даних у зручній формі про прогрес спортсмена;<br>– можливість порівняння спортсменів, команди;<br>– швидкий доступ до результатів змагань які відбулися або плануються. |

Отже, створені портрети допоможуть проаналізувати зібрані дані, зрозуміти потреби та проблеми кожної з груп користувачів у контексті підводного плавання та знайти ефективне рішення кожної проблеми.

## 2.2 Функціональні та нефункціональні вимоги

Зазвичай вимоги поділяються на два типи: функціональні та нефункціональні вимоги.

Функціональні вимоги – це тип вимог, що описують поведінку системи, дії, які система повинна виконувати при виконанні певних умов. Функціональні вимоги зазвичай складаються як з характеристик продукту, так і з вимог користувачів, які можна побачити безпосередньо в кінцевому продукті, на відміну від нефункціональних вимог.

Після опрацювання усіх користувацьких вимог, було виявлено такі функціональні вимоги:

– аутентифікація користувачів з використанням логіну та паролю: користувач повинен мати змогу увійти у систему;

– ролі користувачів: у системі повинно бути визначено три ролі: суддя, тренер, спортсмен, відповідно до них надавати доступу та функції;

– CRUD операції зі змаганнями: суддя повинен мати змогу створювати, редагувати, видаляти змагання;

– CRUD операції зі заявками: тренер повинен мати змогу створювати, редагувати, видаляти заявки на змагання;

– керування інформацією про результати змагань: після проходження дати змагань, суддя має мати змогу додавати, редагувати результати змагань;

– закриття змагань: суддя повинен мати змогу закрити змагання, після чого редагування змагань буде заборонене та опубліковано результати;

– відображення результатів змагань та командного заліку: кожна група користувачів повинна мати змогу переглядати інформацію про результати змагань;

– автоматичне вираховування результатів змагань: система повинна після закриття змагання вираховувати рейтинг спортсменів та командний залік;

– надання доступу до інформації про змагання: кожен користувач повинен мати доступ до перегляду будь якої інформації про змагання, які відбулись або плануються;

– перегляд статистики про спортсмена: система повинна надавати інформацію про кількість зайнятих призових місць, кількість виконаних розрядів, статистику по результатам змагань у певні й категорії за певний рік;

– перегляд статистики про команду: результати команди за певний рік, кількість зайнятих призових місць, розряди учасників команди;

– додавання спортсменів: система повинна надавати змогу обирати спортсменів для порівняння, натиснувши на символ ваг, щоб видалити спортсмена з порівняння, натиснути повторно;

– генерування протоколу результатів у PDF: система повинна надавати змогу суддям генерувати протоколи змагань;

– пошук із фільтрацією змагань: система повинна надавати змогу здійснювати пошук: за назвою, датою проведення та доступною категорією після натискання на кнопку «Search»;

– пошук із фільтрацією спортсменів: система повинна надавати змогу здійснювати пошук: за ім’ям, віком, категорією після натискання на кнопку «Search»;

– пошук команд: система повинна надавати змогу здійснювати пошук за назвою після натискання на кнопку «Search»;

– пошукова стрічка: у разу використання пошуку, усі обрані фільтри мають додаватися до пошукової стрічки;

– перегляд профілю команди: при натисканні на профіль команди, система здійснює навігацію користувача на сторінку з повною інформацією про обрану команду;

– перегляд профілю спортсмена: при натисканні на профіль спортсмена, система здійснює навігацію користувача на сторінку з повною інформацією про обраного спортсмена;

– навігація: у Header повинно бути розташовані усі посилання на ключові сторінки;

– нотатник: система повинна надавати змогу робити замітки користувачу у віртуальному нотатнику, інформація додана до нотатника повинна не бути прив’язана до БД, а зберігатися у локальному сховищі;

– підтвердження CRUD операцій: перед будь-якою CRUD операцією система повинна виводити повідомлення для підтвердження цієї операції, після підтвердження операція повинна бути здійснена;

– вивід повідомлення про результат асинхронної дії: після надсилання запиту на виконання асинхронної дії, користувачу у pop-up вікну повинно з’являтися статус виконання дії.

Нефункціональні вимоги – це тип вимог, що описують властивості або особливості, якою має володіти система. Вони є описом різних аспектів продукту, які важливі для користувачів або для розробників і тих, хто буде

обслуговувати систему. Вони описують такі характеристики: продуктивність, надійність, безпека, зручність використання, сумісність тощо.

Юзабіліті:

– додаток повинен мати простий та зрозумілий інтерфейс, зручну та інтуїтивно зрозумілу навігацію;

– середній час для створення одного змагання, заявки користувачем без сторонньої допомоги повинно займати не більше 30 секунд;

– користувач повинен вміти вільно користуватися додатком на 3-4 раз використання;

– усі шрифти у застосунку повинні бути прості та читабельні;

– у застосунку повинні використовуватися кольори: #a39eff, #ff3f7d, #FFB6C1, #FF69B4, #484848;

– весь застосунок повинен буди у одному дизайні у застосунку повинні буди однакові за стилем кнопки, поля для вводу, модальні вікна тощо.

Безпека: персональні дані користувачів мають бути захищені.

Надійність:

– після здійснення CRUD операцій, ці функції можуть бути готовими до повторного використання після завершення зміни даних у БД;

– форми повинні містити валідацію даних, тобто при введенні некоректних даних вони не можуть бути записані до БД;

– застосунок повинен забезпечувати надійне збереження даних користувачів;

– застосунок повинен бути стабільним та стійким при довготривалих сеансах використання (1-2 години).

Сумісність: застосунок має гарно відображатись у будь-якому браузері.

## 2.3 Опис бізнес-процесів та формулювання вимог

Бізнес-процеси – це послідовність пов’язаних дій та подій, які відбуваються в рамках організації для досягнення конкретної мети або результату. Для досягнення результату та розробки системи, яка буде вирішувати значну кількість проблем кінцевих користувачів, важливо розуміти бізнес-процеси та правильно подати опрацьовані вимоги розробнику.

User story – є одним з популярних підходів, це загальне представлення інформації, коротке формулювання намірів, функцій, що повинна виконувати система, подане з позиції кінцевого користувача. Але варто пам’ятати, що User story не є повноцінною вимогою, а скоріше являє собою обговорюване подання намірів. Вони повинні відповідати правилу INVEST, тобто бути:

– I – Independent (незалежна від інших історій, історії можуть бути реалізовані в різному порядку);

– N – Negotiable (викликає обговорення, відображає суть);

– V – Valuable (ціннісна, відображає цінність для замовника чи кінцевих користувачів);

– E – Estimable (оцінювана, команда може оцінити її складність);

– S – Small (формулювання маленьке, компактне, щоб команда розробників могла вирішити її за одну ітерацію);

– T – Testable (та, яка може бути протестована, обов’язково присутні критерії прийняття).

Текст User story (користувацької історії) повинен пояснювати роль та дії користувача, його потреби та цінність, яку вона несе для користувача. Для правильного формулювання історії, потрібно використовувати шаблон: «Як [роль користувача], я [хочу зробити певні дії], для того щоб …» .

Але лише формулювання User story недостатньо для правильного розуміння задачі, тому також до кожної історії важливо описувати Acceptance Criteria. Acceptance Criteria (або критерії прийняття) – умови,

точні деталі, які є унікальними для кожної User story та які проєкт повинен повністю забезпечувати, також є набором з чітких визначень, що описують функціональні або нефункціональні характеристики. Завдяки Acceptance Criteria розробник знає як повинен виглядати кінцевий результат. Однією з хороших практик є використання Gherkin language під час опису критеріїв прийняття.

Gherkin language – мова, яка використовується для опису поведінки системи, використовуючи сценарії. Що містить десять ключових слів: Given, When, Then, And, But, Scenario, Feature, Background, Scenario Outline, Examples. Кожен непустий рядок у Gherkin починається з ключових слів і представляє собою крок. Сценарій повинен бути зрозумілим та описувати повністю певну частину функціоналу.

Приклад структури:

– GIVEN (дано) – певний стан до того, як користувач увійшов до сценарію.

– WHEN (коли) – щось відбувається, користувач здійснює певні дії.

– THEN (тоді) – система реагує на дії користувача.

Для наглядного зображення бізнес-процесів, використовують побудову BPMN діаграм. BPMN (Модель та нотація бізнес-процесу) – це графічне представлення для визначення бізнес-процесу та його моделі бізнес-процесу, системи умовних позначень, призначених для їх візуалізації. Оскільки елементи нотації є інтуїтивно зрозумілі, їх використання є ефективним для демонстрації команди розробників.

Далі представлені декомпозовані найскладніші у розумінні бізнес-процеси та вимоги.

US№1 Як суддя, я хочу додавати/редагувати інформацією про результати змагань, щоб забезпечити актуальні дані.

*Acceptance Criteria*

**Сценарій**: Додавання/редагування нових результатів:

– GIVEN: Суддя знаходиться на сторінці обраного змагання;

– AND Суддя відкрив вікно «Add results»;

– WHEN: Суддя ввів напроти потрібного спортсмена час;

– AND Натиснув на кнопку «Confirm results»;

– THEN: Данні збережено;

– AND Оновлено вікно «Add results».

**Сценарій**: Дискваліфікація спортсмена:

– GIVEN: Суддя знаходиться на сторінці обраного змагання;

– AND Суддя відкрив вікно «Add results»;

– WHEN: Суддя ввів час, що дорівнює нулю;

– THEN: Учасника дискваліфіковано.

**Сценарій**: Керування результатами, введення некоректних даних

– GIVEN: Суддя знаходиться на сторінці обраного змагання;

– AND Суддя відкрив вікно «Add results»;

– WHEN: Суддя ввів нечислові символи;

– THEN: Виведено попередження про некоректно введені данні.

US№2 Як суддя, я хочу мати змогу переглядати кінцеву залікову таблицю перед закриттям змагання, щоб впевнитися у коректності результатів.

*Acceptance Criteria*

**Сценарій**: Перегляд кінцевої таблиці:

– GIVEN: Суддя відкрив вікно «Add results»;

– WHEN: Суддя відкрив секцію «Close competition»;

– THEN: Відображено кінцеву залікову таблицю результатів.

US№3 Як суддя, я хочу мати змогу закрити змагання, щоб опублікувати результати та забезпечити прозорість змагань.

*Acceptance Criteria*

**Сценарій**: Закриття змагання:

– GIVEN: Суддя відкрив секцію «Close competition»;

– WHEN: Натиснув на кнопку «Close competition»;

– THEN: Змагання завершено з забороною редагування заборонено;

– AND Сторінка змагань містить інформацію про результати.

Діаграма показана на рисунку 2.1 зображує бізнес-процес «Управління результатами змагань» описаних у користувацьких історіях

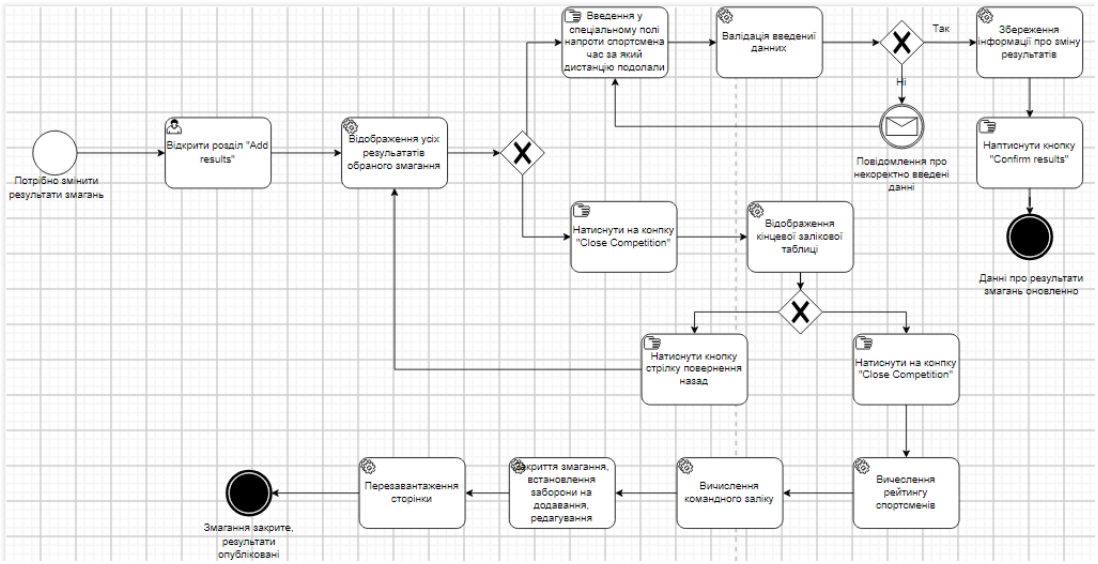


Рисунок 2.1 – BPML-діаграма бізнес процесу «Управління результатами змагань»

US№4 Як тренер, я хочу мати змогу переглядати список спортсменів, які можуть взяти участь у змаганні, щоб швидко подати заявку на змагання.

*Acceptance Criteria*

**Сценарій**: Перегляд доступних спортсменів:

– GIVEN: Тренер знаходиться на сторінці обраного змагання;

– WHEN: Натиснув на кнопку «Add particspants»;

– THEN: Відкрилося вікно з посортованими по віковій категорії учасниками команди, які можуть брати участь у змаганні.

US№5 Як тренер, я хочу мати змогу переглядати які категорії доступні для участі у змаганнях, щоб подати вірно заявку.

*Acceptance Criteria*

**Сценарій**: Подання заявки:

– GIVEN: Тренер знаходиться у секції «Add particspants»;

– WHEN: Натиснув на обраного спортсмена;

– THEN: Виведено інформацію про доступні категорії у змаганнях.

US№6 Як тренер, я хочу мати змогу подавати заявку на участь у змаганнях, щоб забезпечити участь своїх спортсменів у змаганнях.

*Acceptance Criteria*

**Сценарій**: Подання заявки:

– GIVEN: Тренер знаходиться у секції «Add particspants»;

– WHEN: Ввів час у відповідному полі напроти потрібної категорії;

– AND Натиснув кнопку «Confirm»;

– THEN: Заявку подано.

**Сценарій**: Подання заявки введення некоректних даних:

– GIVEN: Тренер знаходиться у секції «Add particspants»;

– WHEN: Ввів нечислові символи у відповідному полі напроти потрібної категорії;

– AND Натиснув кнопку «Confirm»;

– THEN: Виведено попередження про некоректно введені данні.

US№7 Як тренер, я хочу мати змогу редагувати заявку на участь у змаганнях, щоб надавати коректну інформацію.

*Acceptance Criteria*

**Сценарій**: Редагування часу заявки:

– GIVEN: Тренер знаходиться у секції «Add particspants»;

– WHEN: Ввів час у відповідному полі напроти потрібної категорії;

– AND Натиснув кнопку «Confirm»;

– THEN: Час заявки змінено.

**Сценарій**: Видалення заявки:

– GIVEN: Тренер знаходиться у секції «Add particspants»;

– WHEN: У відповідному полі напроти зазначив час заявки, як нуль;

– AND Натиснув кнопку «Confirm»;

– THEN: Заявку видалено.

Діаграма показана на рисунку 2.2 зображує бізнес-процес «Управління заявками» описаний у користувацьких історіях.

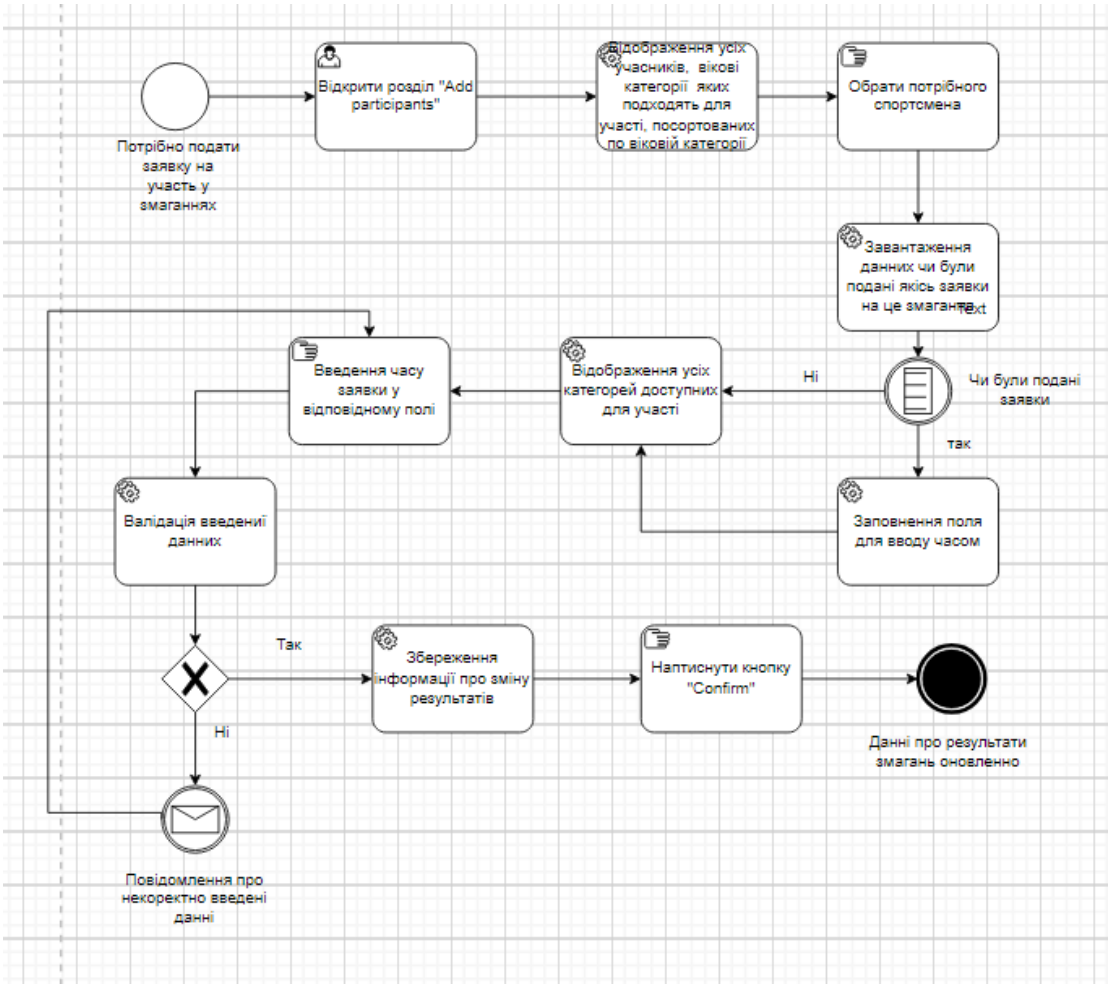


Рисунок 2.2 – BPML-діаграма бізнес процесу «Управління заявками»

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАСТОСУНКУ

## 3.1 Розробка структури БД

Наступним етапом після завершення етапу визначення та формування вимог до застосунку – є розробка конфігурації, структури та організація БД.

Системою управління БД було обрано об’єктно-реляційна система PostgreSQL. PostgreSQL вважається стабільною та надійною БД, оскільки вбудовані механізми дозволяють забезпечити цілісність даних та їх відновлення в разі відмов. Обрана СУБД підтримує різні рівні розширюваності, горизонтальну та вертикальну масштабованість, що є досить корисним механізом, тому що дозволяє пристосовувати його під зростаючі потреби застосунку. PostgreSQL має великий набір функцій: підтримку різних типів даних, відносин, геоданих, текстових пошуків, аналітичних функцій. Також обрана СУБД гарантує дотримання властивостей ACID (атомарності, консистентності, ізоляції та тривалості), щоб забезпечення надійності та цілісності транзакцій. PostgreSQL має велику активну спільноту розробників та користувачів, тому забезпечує швидку підтримку у вирішенні проблем. Крім того, PostgreSQL має дружній інтерфейс та короткий час освоєння, що сприяє швидкому вирішенню завдань і проблем.

В даному випадку було обрано створення реляційної БД, тому що у розробленому застосунку дані мають складну структуру з багатьма взаємозв’язками між об’єктами, що робить реляційну базу даних таку як PostgreSQL, більш придатною для зберігання та організації даних.

Для створення конфігурації схеми бази даних, що описує структуру даних, їх зв’язки та типи даних було обрано Prisma ORM. Prisma ORM полегшує взаємодію з базою даних, що робить процес розробки застосунків більш швидким. Наведемо кілька переваг використання цієї ORM:

– зручний та інтуїтивно зрозумілий інтерфейс для взаємодії з базою даних сприяє спрощуванню роботи розробника;

– Prisma допомагає уникнути помилок і забезпечує безпеку даних, оскільки має строгу типізацію та автоматичну валідації даних;

– Prisma використовує високопродуктивний механізм для запитів до бази даних, що покращує продуктивність програм;

– Prisma підтримує популярні бази даних: MySQL, PostgreSQL, SQLite та Microsoft SQL Server;

– Prisma надає простий у використанні інструмент міграції, який дає змогу легко вносити зміни до схеми бази даних, зберігаючи дані;

Після встановлення Prisma ORM у проєкт, першим кроком у створенні конфігурації є підключення до БД, у лістингу 3.1 наведено приклад підключення до БД.

Лістинг 3.1 Приклад створення відношень:

```
generator client {
  provider = "prisma-client-js"
}
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}
```

Наступним кроком є створення таблиць та відношень, що наведено у лістингу 3.2.

Лістинг 3.2 Приклад створення відношень та таблиць:

```
model User {
  id Int @id @unique @default(autoincrement())
  login String @unique
```

```
  password  String @default("1111")
  role ROLE
  team Team?
  sportsman Sportsman?
  judge     Judges?
}
model Sportsman {
  name String
  sex SEX
  year Int
  category String
  user User @relation(fields: [id], references: [id])
  id Int @id @unique
  team Team? @relation(fields: [idTeam], references: [id])
  idTeam Int? competitionCategoryResult CompetitionCategoryResult[]
}
enum ROLE {
  sportsman
  judge
  team
}
```

Під час розробки бази даних для даного застосунку було визначено наступні сутності: User, Team, Judges, Sportsman, CategorySportsman, Competition, Category, CompetitionCategory, CompetitionCategoryResult, TeamResult (табл. 3.1).

Таблиця 3.1 – Структура БД

| **Сутність** | **Атрибут** | **Тип та домени** |
|---|---|---|
| **1** | **2** | **3** |
| User | id | int, PK, FK |
| | login | text, NOT NULL, UNIQUE |
| | password | text, NOT NULL |
| | role | role, NOT NULL |
| Team | id | int, PK, FK |
| | name | text, NOT NULL |
| Judges | id | int, PK, FK |
| | name | text, NOT NULL |
| Sportsman | id | int, PK, FK |
| | name | text, NOT NULL |
| | sex | sex, NOT NULL |
| | idTeam | int |
| | category | text, NOT NULL |
| | year | int ,NOT NULL |
| CategorySportsman | id | int, NOT NULL, PK |
| | category | text, NOT NULL |
| | sex | int, NOT NULL |
| | MSUMK | double precision, NOT NULL |
| | MSU | double precision, NOT NULL |
| | KMSU | double precision, NOT NULL |
| | I | double precision, NOT NULL |
| | II | double precision, NOT NULL |
| | III | double precision, NOT NULL |
| Competition | id | int, FK, NOT NULL |
| | name | text, NOT NULL |
| | date | date, NOT NULL |
| | description | text, NOT NULL |
| | completed | boolean, NOT NULL |
| Category | id | int, PK |
| | category | text, NOT NULL |
| | ageGroup | ageGroup, NOT NULL |

Продовження таблиці 3.1

| 1 | 2 | 3 |
|---|---|---|
| CompetitionCategory | id | int, PK |
| | idCompetition | int, NOT NULL |
| | idCategory | int, NOT NULL |
| CompetitionCategoryResult | id | int, PK |
| | idCompetitionCategory | int, NOT NULL |
| | idSportsman | int, NOT NULL |
| | timeInvoice | double precision, NOT NULL |
| | timeResult | double precision, NOT NULL |
| | place | int, NOT NULL |
| | categorySportsmanDone | text, NOT NULL |
| TeamResult | id | int, PK |
| | idCompetition | int, NOT NULL |
| | idTeam | int, NOT NULL |
| | mark | int, NOT NULL |
| | place | int, NOT NULL |

Структура створеної бази даних, дозволяє швидко та ефективно керувати даними про змагання, користувачів, суддів, команд, результати команди та спортсменів, що є важливим для забезпечення безперебійної роботи вебзастосунку, оскільки можна легко керувати даними з бази даних. Розроблена база даних відповідає усім нормальним формам.

Таким чином після усіх вказаних раніше маніпуляцій, отримуємо таку БД (рис. 3.1).

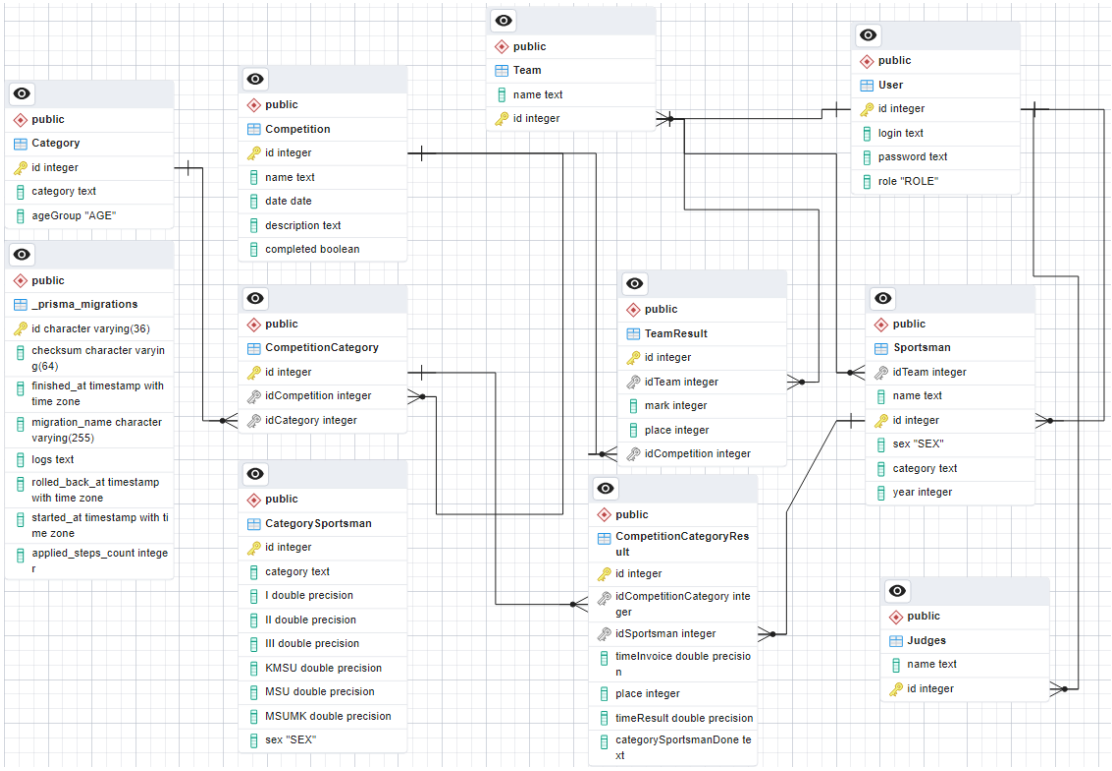


Рисунок 3.1 – ERD-діаграма БД

## 3.2 Розробка клієнтської частини застосунку

Клієнтська частина застосунку – це та частина програмного забезпечення, яка виконується на боці клієнта, тобто комп’ютер на стороні користувача, за допомогою чого той відправляє запит до сервера задля надання інформації чи з проханням інших дій.

Завдання клієнтської частини додатку включає:

– надання користувальницького інтерфейсу;

– відправка запитів до сервера;

– прийом і обробка відповідей від сервера;

– взаємодія з локальними даними (зберігання, кешування даних локально на пристрої користувача);

– виконання локальної логіки (валідація даних, обробка подій і дій користувача тощо).

Для розробки фронтенд-частини (клієнтської частини) вебзастосунку було обрано бібліотеку React. Вона підтримує співпрацю з ключовими фронтенд-технологіями: HTML, CSS і JavaScript. Але найбільша перевага цієї бібліотеки – використання компонентного підходу. Компоненти можуть бути повторно використані, замість одного великого монолітного компоненту, отримуємо змогу створити велику кількість малих компонентів, що забезпечить більшу модульність системи та швидкість розробки, кожен компонент може бути незалежним, що спрощує тестування. Також компонентний підхід дозволяє покращити читабельність коду, що сприяє легкій підтримці коду.

Для створення якісного застосунку використання однієї бібліотеки недостатньо, тому доповнюють функціональність React такі бібліотеки як Redux, React Router та інші, що допоможуть створити або розширити функції застосунку. Усі використані бібліотеки описані у Таблиці 3.2.

Таблиця 3.2 – Опис використаних бібліотек

| **Назва** | **Опис** |
|---|---|
| **1** | **2** |
| Redux | Відкрита JavaScript бібліотека призначена для керування станом програм. |
| Redux Toolkit | Інструмент, головним призначенням якого є спрощення та полегшення розробки з використанням бібліотеки Redux. |
| React Router | Дозволяє забезпечити маршрутизацію на стороні клієнта, таким чином дозволяючи створювати SPA застосунки з багатосторінковою поведінкою без перезавантаження сторінок. |
| Axios | Бібліотека для виконання HTTP-запитів у проєкті, що дозволяє зручно взаємодіяти з сервером для отримання та відправлення даних. |

Продовження таблиці 3.2

| 1 | 2 |
|---|---|
| Chart.js | Бібліотека призначена для покращення візуалізації даних у вебзастосунках, надає можливість побудови різних графіків і діаграм, що дозволяє відображати дані у зручному форматі. |
| Classnames | Невелика бібліотека, яка допомагає зручно працювати з CSS класами у JavaScript, завдяки якій можливо динамічно управляти класами, що дозволяє легко змінювати стиль компонентів у відповідь на різні зміни умов. |
| Date-fns | Бібліотека основною метою якої є полегшення роботи з датами та часом у JavaScript. Великий набір функцій: парсинг, форматування, операції зі зміною дат та інше. |
| React-loader-spinner | Простий компонент React SVG spinner, що застосовується під час очікування асинхронної операції. |

Першим кроком у створенні front-end частини застосунку є створення React проєкту, за допомогою інструменту командного рядка для швидкого створення нового проєкту – create-react-app, у лістингу 3.3 наведено приклад створення проєкту.

Лістинг 3.3 Приклад створення React проєкту:

```
npx create-react-app my-app
cd my-app
npm start
```

Важливою частиною React проєкту файл package.json, у якому міститься інформація конфігурації, метадані застосунку: назва, версія, опис, головний файл програми, залежності, які потрібні для його роботи, скрипти тощо. Також важливу інформацію про проєкт, яку містить файл package.json, можуть використовувати інші розробники, що дозволяє легко взаємодіяти з кодом під час командної розробки.

Лістинг 3.4 package.json файл:

```
{
  "name": "swim_front",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint . --ext js,jsx --report-unused-disable-directives --max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@reduxjs/toolkit": "^1.9.7",
    "axios": "^1.6.0",
    "chart.js": "^4.4.0",
    "classnames": "^2.3.2",
    "date-fns": "^3.2.0",
    "node-sass": "^9.0.0",
    "react": "^18.2.0",
    "react-chartjs-2": "^5.2.0",
    "react-dom": "^18.2.0",
```

```
    "react-loader-spinner": "^5.4.5",
    "react-redux": "^8.1.3",
    "react-router-dom": "^6.18.0"
  },
  "devDependencies": {
    ...
  }
}
```

Тепер перейдемо до налаштування головного файлу проєкту main.js. Цей файл містить основний скрипт, який відповідає за ініціалізацію React проєкту. У ньому весь застосунок обгортається компонентом Provider, що передає Redux store всім компонентам проєкту, щоб вони могли отримувати доступ до стану Redux. Компонент BrowserRouter надає маршрутизацію, тобто дозволяє визначати маршрути для різних URL-адрес. Також ми підключаємо компонент App – головний компонент застосунку, що містить всі інші компоненти та функціональність. Далі рядком ReactDOM.createRoot(document.getElementById("root")).render() створюємо корінь проєкту за допомогою методу createRoot, після чого він рендерить кореневий компонент додатку, обгорнутий в компоненти Provider та BrowserRouter.

Лістинг 3.5 main.js файл:

```
import React from "react";
import ReactDOM from "react-dom/client";
import { BrowserRouter } from "react-router-dom";
import { Provider } from "react-redux";
import store from "./store";
import App from "./App.jsx";
import "./index.scss";
```

```
ReactDOM.createRoot(document.getElementById("root")).render(
  <Provider store={store}>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </Provider>
);
```

Після завершення налаштування головних файлів перейдемо до огляду папок у проєкті (рис. 3.2). Оптимальним рішенням у проєкті є використання підходу «групування файлів за їх типами»:

– assets – папка з картинками;

– constans – папка, де зберігаються сталі значення;

– components – папка з компонентами;

– hooks – папка з створеними хуками;

– http – налаштування для використання Axios для взаємодії з сервером;

– localStorage – функції для роботи з localStorage;

– pages – папка, де зберігаються компоненти сторінки;

– SCSS – містить файли з загальними налаштуваннями стилів у проєкті;

– service – містить файли з наборами функцій для комунікації з сервером;

– shared – містить корисні функції, які можуть бути використані у будь-якій частині проєкту;

– store – файли для роботи зі стором Redux.

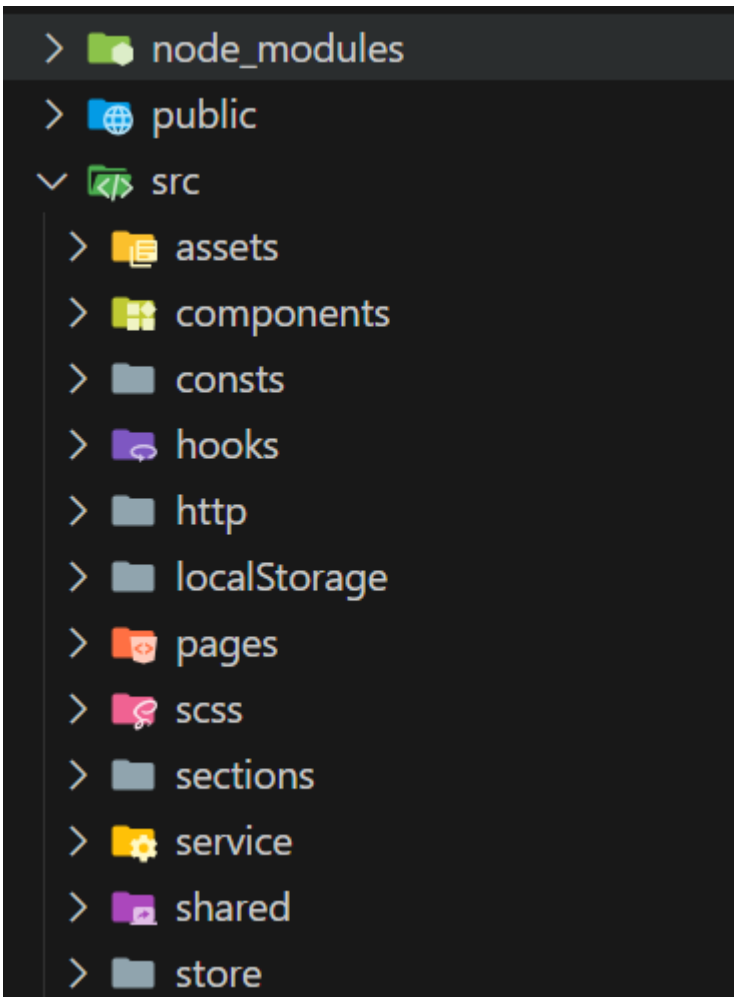


Рисунок 3.2 – Організація папок у проєкті

У свою чергу компоненти поділені на звичайні компоненти та UI компоненти – повторно використовувані компоненти, що відповідають за відображення різноманітних елементів користувацького інтерфейсу, тобто інпути, кнопки, селекти тощо (рис. 3.3).

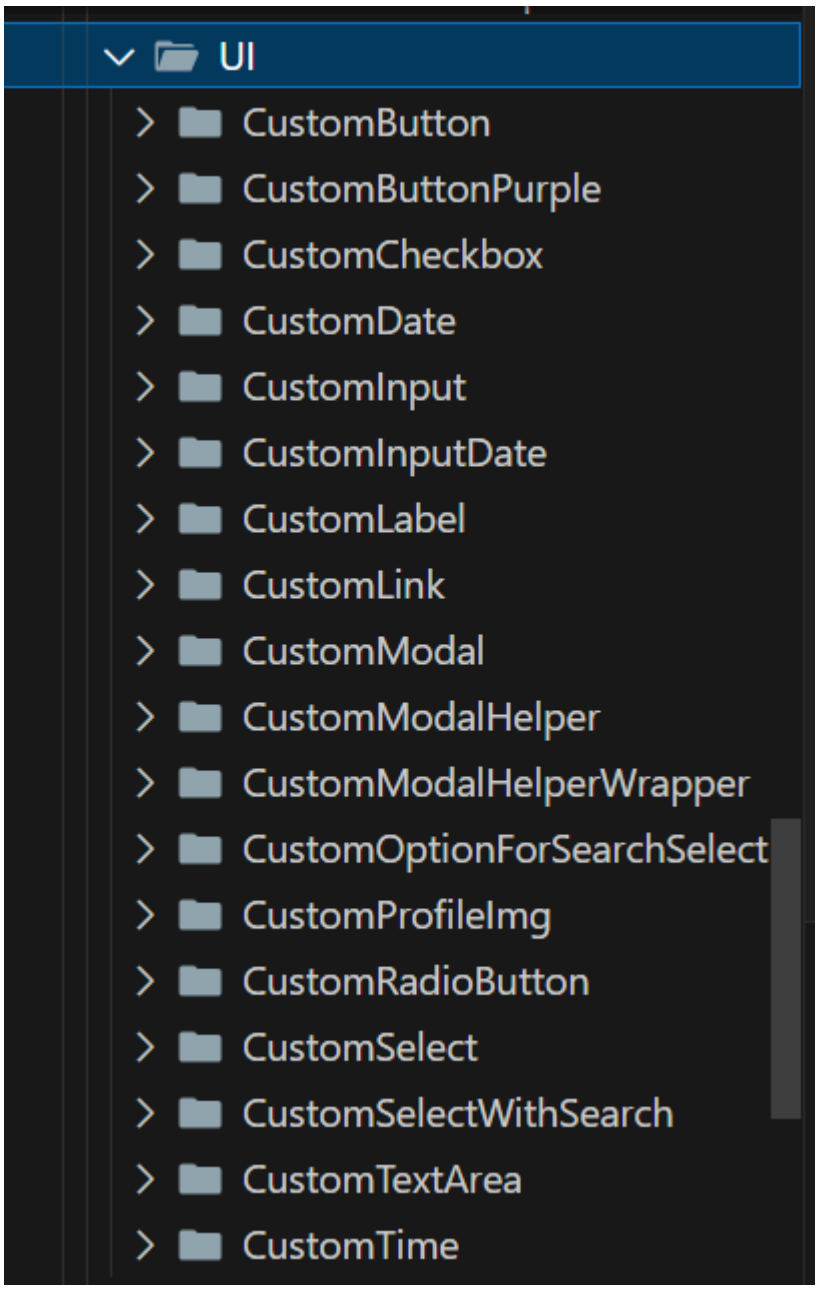


Рисунок 3.3 – Усі UI компоненти

Розроблений застосунок містить вісім сторінок:

– Home – сторінка з профілями користувачів;
– SighIn – сторінка авторизації;
– SerchPage – сторінка з пошуком змагань, спортсменів, команд;
– CompetitionPage – сторінка, де відображається обране змагання;
– SportsmanPage – сторінка, де відображається обраний спортсмен;
– TeamPage – сторінка, де відображається обрана команда;

– CompereSportsmanPage – сторінка, де здійснюється порівняння спортсменів;

– CompereTeamPage – сторінка, де здійснюється порівняння команд.

Приклад створення компоненту сторінки наведено у лістингу 3.6.

Лістинг 3.6 Компонет сторінка Home

```
const HomePage = () => {
  const userInfo = useSelector((state) => state.user.userInfo);
  return (
    <div className={styles.wrapper}>
      {userInfo.role == "team" ? (
        <TeamSection />
      ) : userInfo.role == "judge" ? (
        <JudgeSection />
      ) : (
        <SportsmanSection />
      )}
    </div>
  );
};
export default HomePage;
```

Після створення компонентів сторінок необхідно створити маршрути, за якими користувач зможе переходити на потрібно сторінку. У головний компонент додатку App будуть визначатися всі маршрути та компоненти, які будуть відображатися при переході за цими маршрутами. Для цього потрібно використати контейнер для маршрутів – компонент Routes, маршрути визначаються всередині цього компонента, після чого використовуємо компонент Route, де вказуємо параметр path – маршрут та element – елемент що буде відображатися на сторінці.

Лістинг 3.7 Навігація застосунку

```
function App() {
  return (
    <>
      <Routes>
        <Route path="/login" element={<SignInPage />} />
        <Route path="/" element={<RequiredAuth />}>
          <Route index element={<HomePage />} />
          <Route path="search" element={<SearchPage />} />
          <Route path="teamStatistic/:id" element={<TeamPage />} />
          <Route path="sportsmanStatistic/:id" element={<SportsmanPage
/>} />
          <Route
            path="competitionStatistic/:id"
            element={<CompetitionPage />}
          />
          <Route
            path="compareSportsman/:id1/:id2"
            element={<CompareSportsmanPage />}
          />
          <Route path="compereTeams/:id1/:id2"
element={<CompereTeamsPage />} />
        </Route>
      </Routes>
    </>
  );
}
```

Оскільки усі сторінки, окрім *SignInPage,* не можуть бути використані неавторизованими користувачами, потрібно обгорнути усі інші маршрути у компонент, який буде відслідковувати чи авторизований користувач та чи валідний його JWT. У данному випадку було створено компонент RequiredAuth. За допомогою створеного хук useAuth отримуємо інформацію про статус аутентифікації користувача, тобто отримуємо токен користувача та надсилаємо запит на сервер з проханням перевірити валідність. У позитивному випадку надається загальна інформація про користувача так як id тощо. Після цього надається доступ до потрібної сторінки. У іншому випадку – навігуємо користувача на сторінку для авторизації.

Лістинг 3.8 Компонент RequiredAuth

```
const RequiredAuth = () => {
  const auth = useAuth();
  const dispatch = useDispatch();
  const status = useSelector((state) => state.user.status);
  useEffect(() => {
    dispatch(fetchUserInfo());
  }, []);
  useEffect(() => {
    if (status == "rejected") {
      localStorage.clear();
    }
  }, [status]);
  useCompare();
  return auth ? (
    <>
      {status == "rejected" ? (
        <Navigate to="/login" />
      ) : status == "fulfilled" ? (
```

```
        <>
          <HeaderPage />
          <Outlet />
          <FooterPage />
        </>
      ) : ( <LoaderReact /> )}
    </>
  ) : (  <Navigate to="/login" /> );
};
```

## 3.3 Розробка серверної частини застосунку

Серверна частина застосунку – це та частина програмного забезпечення, яка виконується на сервері і відповідає за обробку запитів від клієнтської частини. Серверна частина виконує такі завдання:

– обробка запитів від клієнтів і надання їм відповідей;

– забезпечення взаємодії з базою даних;

– виконання бізнес-логіки застосунку;

– забезпечення безпеки та захисту даних;

– масштабування та оптимізація для забезпечення високої продуктивності та доступності.

Архітектурним підходом для створення веб-сервісів обрано REST API. REST API використовує HTTP протокол та використовує стандартні методи такі як GET, POST, PUT, DELETE, для взаємодії з ресурсами, тобто даними через визначені URL-адреси.

Для розробки бекенд-частини (серверної частини) вебзастосунку було обрано фреймворк NestJS. NestJS – це платформа для створення ефективних, масштабованих серверних програм Node.js, що повністю підтримує використання TypeScript та поєднує у собі в собі елементи об’єктно-

орієнтованого програмування та функціонального програмування. У середині NestJS використовує Express, але забезпечує значно вищий рівень абстракції ніж Express, що дозволяє розробникам використовувати незліченну кількість сторонніх модулів, які доступні для основної платформи.

Важливим аспектом під час реалізації серверної частини застосунку також є файл package.json, у якому знаходиться інформація конфігурації, метадані застосунку.

Наступним кроком після створення проєкту та налаштування файлу package.json є налаштування головного файлу main.ts. Необхідно створити функцію, яка буде запускати програму, у якій потрібно створити екземпляр програми Nest, за використання базового класу NestFactory, що надає певні статичні методи, що дозволяють створити екземпляр програми, після чого метод create() повертає об’єкт програми. А також налаштувати політику CORS, викликавши метод enableCors, який налаштовує параметри CORS такі як дозвіл на всі джерела, дозволені HTTP методи і підтримка cookie. Після налаштування CORS запускаємо сервер, тобто викликається метод listen для запуску сервера на певному порту.

Лістинг 3.9 Файл main.ts

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors({
    origin: true,
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS',
    credentials: true,
  });
  await app.listen(3000, () => {
    console.log('start app');
  });}
bootstrap();
```

Наступним етапом є розбиття програми на окремі сутності. Для цього ми використовуємо ключовий архітектурний принцип NestJS – концепцію модулів, контролерів та сервісів. Кожна сутність розглядається як окремий модуль, в рамках якого створюються відповідні контролери, сервіси та об’єкти передачі даних (DTO). Такий підхід дозволяє краще організувати код і сприяє підтримці програми. У результаті було створено п’ять сутностей: спортсмени, команда, суддя, змагання та результат змагання.

Модулі – контейнери для різних частин програми таких як контролери, сервіси та інші пов’язані компоненти, забезпечуючи організацію та інкапсуляції функціональності програми.

Сервіси інкапсулюють бізнес-логіку та проводять маніпуляції даними, зазвичай використовуються контролерами для виконання запитів, тобто відповідають за функціональність програми, зберігаючи контролери зосередженими на обробці запитів. Для створення сервісу необхідно створити новий клас, використовувати декоратор @Injectable() для позначення класу як сервісу, що підлягає ін’єкції залежностей. Далі у конструкторі класу вказати необхідні залежності, які будуть ін’єктовані за допомогою NestJS Dependency Injection (у лістингу 3.10 використано сервіс по роботі з базою даних PrismaService), після чого створити методи до класу, які будуть виконувати певні операції.

Лістинг 3.10 Приклад сервісу

```
@Injectable()
export class SportsmanService {
  constructor(private prisma: PrismaService) {}
  async getSportsmanById(id: number) {
    const sportsman = await this.prisma.sportsman.findUnique({
      where: { id: Number(id) },
      include: { team: true},
    });
```

```
    return sportsman;
  }
}
```

Контролери відповідають за обробку вхідних HTTP запитів та надання відповідей, тобто вони служать точками входу в логіку вашої програми. Кожен контролер пов’язаний із певним маршрутом та відповідає за визначення методів обробки запитів, контролери використовують декоратори для визначення шляху маршруту, методу HTTP (GET, POST, PUT, DELETE тощо) і параметрів запиту. Вони також взаємодіють зі сервісами – службами для виконання бізнес-логіки. Для створення контролера використовується декоратор @Controller('sportsman'), який вказує, що створений клас є контролером для маршрутів та їх шляху. Після цих маніпуляцій необхідно створити безпосередньо клас (у лістингу 3.11 це SportsmanController, конструктор якого приймає сервіс SportsmanService), який буде використовуватися для обробки запитів. Наповнюємо клас методами, кожен метод відповідає за обробку певного типу запиту, використовуючи відповідні декоратори HTTP методів (@Get, @Put) та маршрутів (@Query, @Body, @Param) для отримання доступу до параметрів запиту. Наприклад, метод getAllTeamMembers відповідає за отримання даних про членів команди за їх ідентифікатором, а метод deleteSportsmanFromTeam відповідає за оновлення інформації про команду спортсмена.

Лістинг 3.11 Приклад контролеру сутності

```
@Controller('sportsman')
export class SportsmanController {
  constructor(private sportsmanService: SportsmanService) {}
  @Get('/sportsmanWithoutTeam')
  getAllSportsmanWithoutTeam() {
    return this.sportsmanService.getAllSportsmanWithoutTeam();
```

```
  }
  @Put('/deleteSportsmanFromTeam/:id')
  deleteSportsmanFromTeam(@Param('id') id: number) {
    return this.sportsmanService.deleteSportsmanFromTeam(id);
  }
...
}
```

Для того щоб обмежити можливість надсилати деякі запити неавторизованим користувачам, було прийнято рішення скористатися технологією Guards. Єдина задача яких полягає у визначенні чи буде даний запит виконано обробником маршруту чи ні, залежно від певних умов. У лістингу 3.12 показано створений екземпляр Guard класу, який імплементує CanActivate і має Injectable декоратор. Він відповідає за перевірку JWT для автентифікації користувачів. Метод canActivate() перевіряє, чи є користувач аутентифікованим на основі JWT, отриманого вхідного запиту. У випадку якщо токен не є дійсним або відсутній у заголовку запиту, генерується повідомленням про помилку.

Лістинг 3.12 Приклад створення Guard класу

```
@Injectable()
export class JwtAuthGuard implements CanActivate {
  constructor(private jwtService: JwtService) {}
  canActivate(
    context: ExecutionContext,
  ): boolean | Promise<boolean> | Observable<boolean> {
    const req = context.switchToHttp().getRequest();
    try {
      const authHeader = req.headers.authorization;
      const [bearer, token] = authHeader.split(' ');
```

```
        if (bearer !== 'Bearer' || !token) {
          throw new UnauthorizedException({ message: 'User isn`t authorized' });
        }
        const user = this.jwtService.verify(token);
        req.user = user;
        return true;
      } catch (error) {
        throw new UnauthorizedException({ message: 'User isn`t authorized' });
      }
    }
  }
```

## 3.4 Інструкція користувача

**Авторизація:** Якщо ви неавторизований користувач, введіть логін у поле «Login» та пароль у поле «Password» та натисніть кнопку «Sign In» (рис. 3.4).

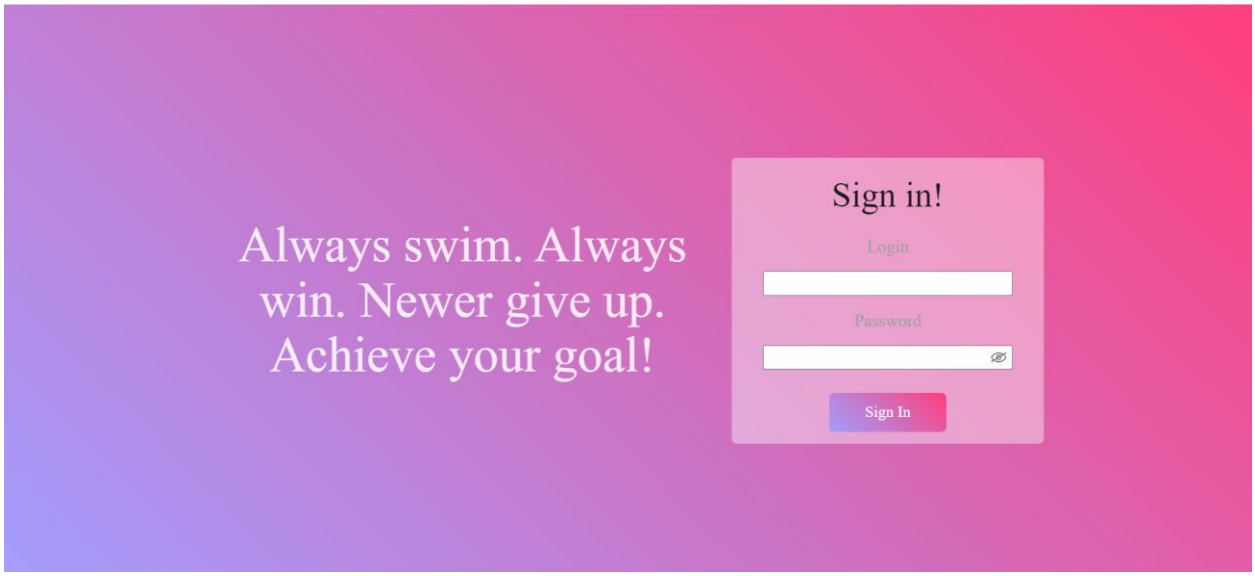


Рисунок 3.4 – Вікно авторизації

**Навігаційна панель у Header:** Після успішної авторизації перед користувачем з’являється його профіль з доступними йому функціями в залежності від його ролі у системі (рис 3.5). Якщо ви хочете вийти з вашого облікового запису – натисніть кнопку «Log out», якщо ви хочете перейти на сторінку пошуку – натисніть «Search», якщо ви хочете переглянути спортсменів або команди, які обрано для порівняння – натисніть «Compare», для повернення на головні сторінку – натисніть «Home».


Рисунок 3.5 – Навігаційна панель

**Пошук:** При відкритті сторінки завантажується інформація про всі команди\спортсменів\змагання в залежності від обраної секції. Оберіть категорію пошуку (рис. 3.6). Введіть назву та оберіть фільтри, натиснувши на відповідний значок (рис. 3.7), натисніть на кнопку «Search». Завантаживши список, оберіть потрібний профіль та натисніть на копку «View» (рис. 3.8) щоб перейти на сторінку з більш детальною інформацією.


Рисунок 3.6 – Категорії пошуку


Рисунок 3.7 – Фільтрація


Рисунок 3.8 – Профілі знайдені за пошуком

**Додавання у порівняння:** Щоб додати спортсмена або команду у порівняння натисніть, на відповідну іконку ваг напроти потрібного профілю (рис. 3.9). Якщо іконка змінила колір на жовтий – профіль додано, щоб видалити профіль з порівняння – натисніть повторно. Щоб переглянути список доданих профілів – натисніть «Compare» у навігаційній панелі (рис. 3.5), після чого з’явиться вікно з потрібною інформацією (рис. 3.10), щоб перейти на сторінку порівняння натисніть на іконку ваг між спортсменами.


Рисунок 3.9 – Додавання профілю у порівняння

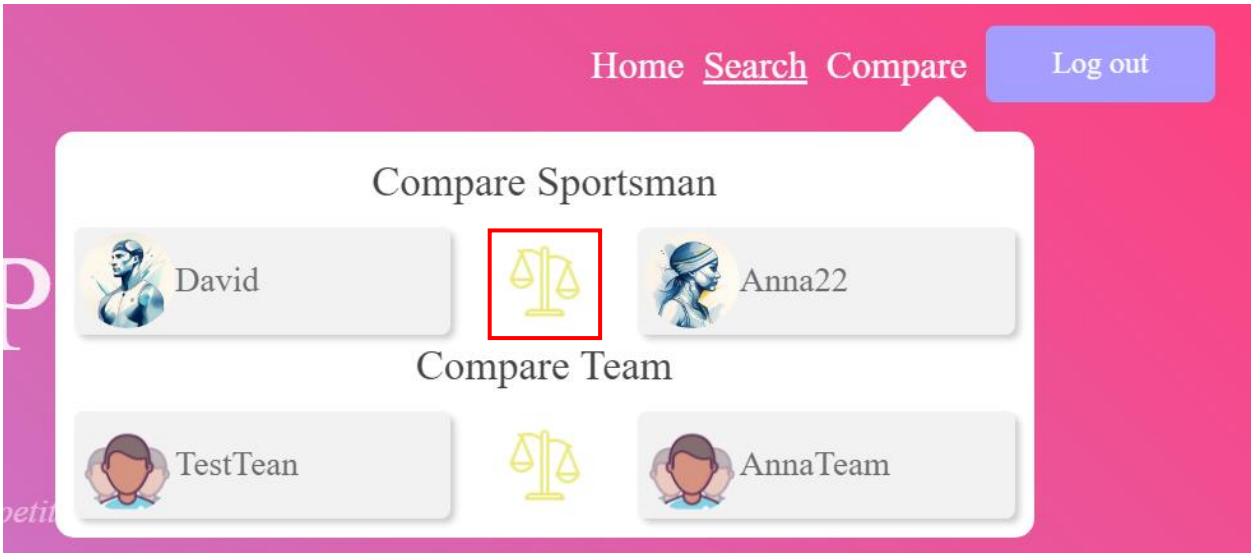


Рисунок 3.10 – Порівняння профілів

**Порівняння спортсменів:** Після переходу на сторінку з порівнянням спортсменів перед користувачем з’явиться інформація про спортсменів, статистика по виконаним розрядам та здобутим місцям (рис. 3.11). Для того щоб провести порівняння спортсменів у певній дисципліні, натисніть на неї (рис. 3.12).

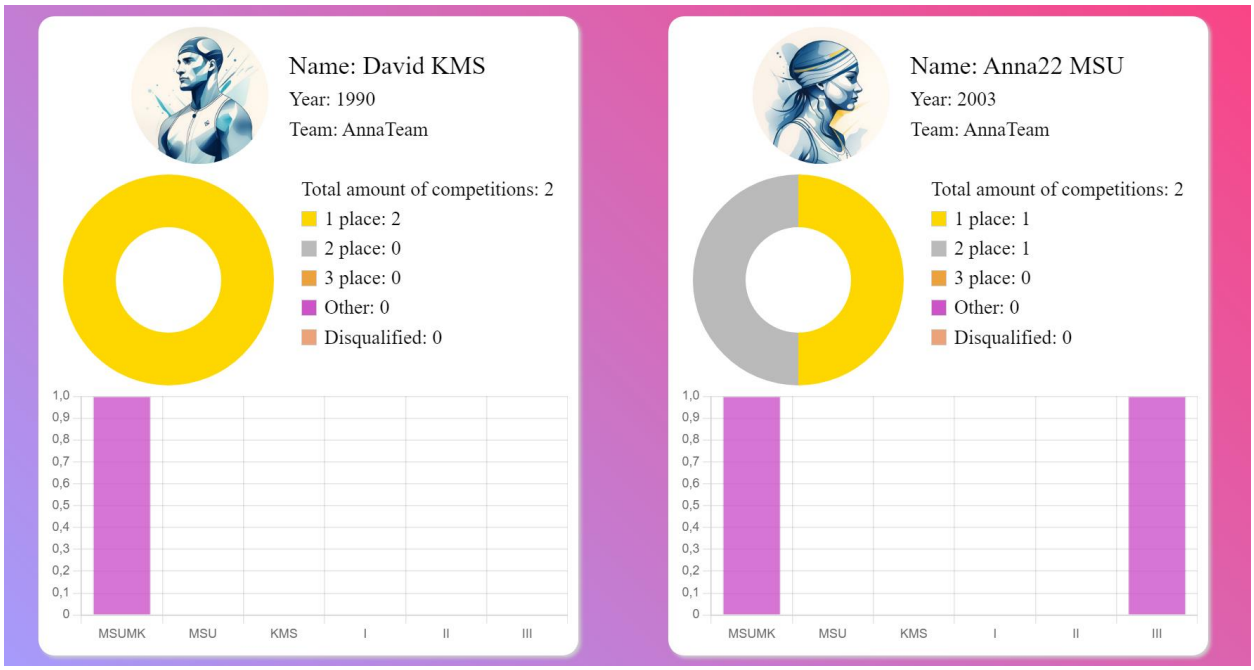


Рисунок 3.11 – Порівняння профілів спортсменів

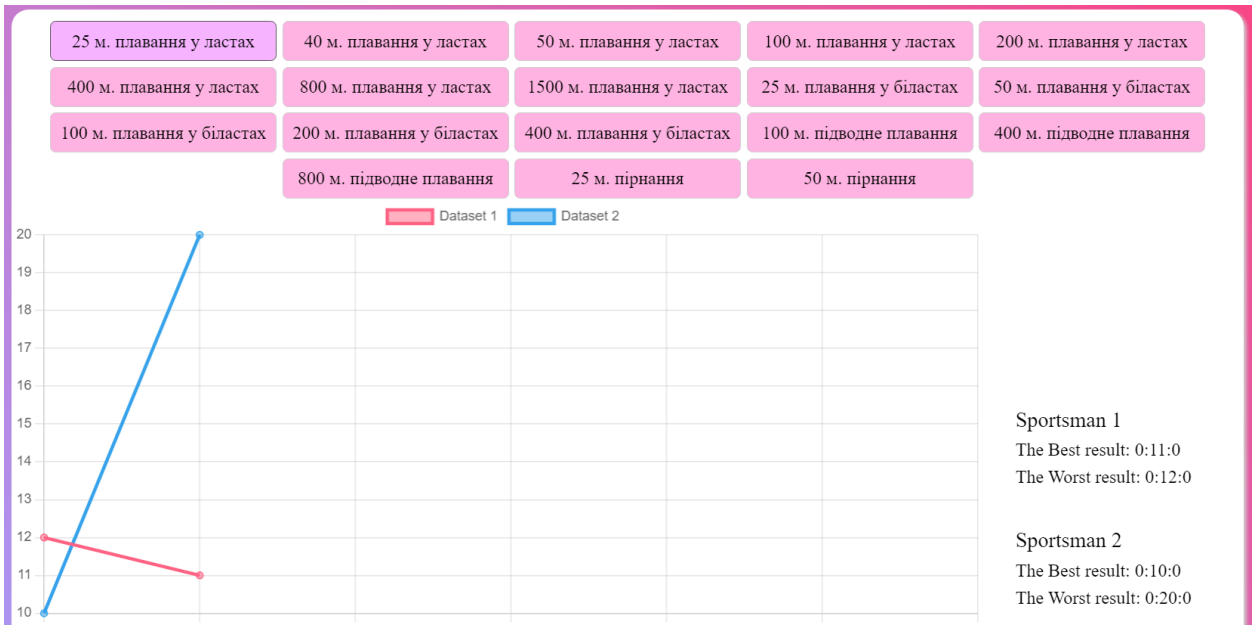


Рисунок 3.12 – Секція порівняння у дисципліні

**Порівняння команд:** Порівняння команд здійснюється аналогічним способом як і порівняння спортсменів.

**Подача заявок:** Після переходу на сторінку з потрібним змаганням, ви можете ознайомитись з усією доступною інформацією про нього, подавати заявки можуть лише користувачі категорії тренери. Для цього натисніть на кнопку «Add participants» (рис.3.13), після чого оберіть потрібного спортсмена (рис.3.14), заповніть інформацію про час та натисніть на кнопку «Confirm» (рис.3.15). Щоб анулювати заявку, вкажіть час, що дорівнює нулю.

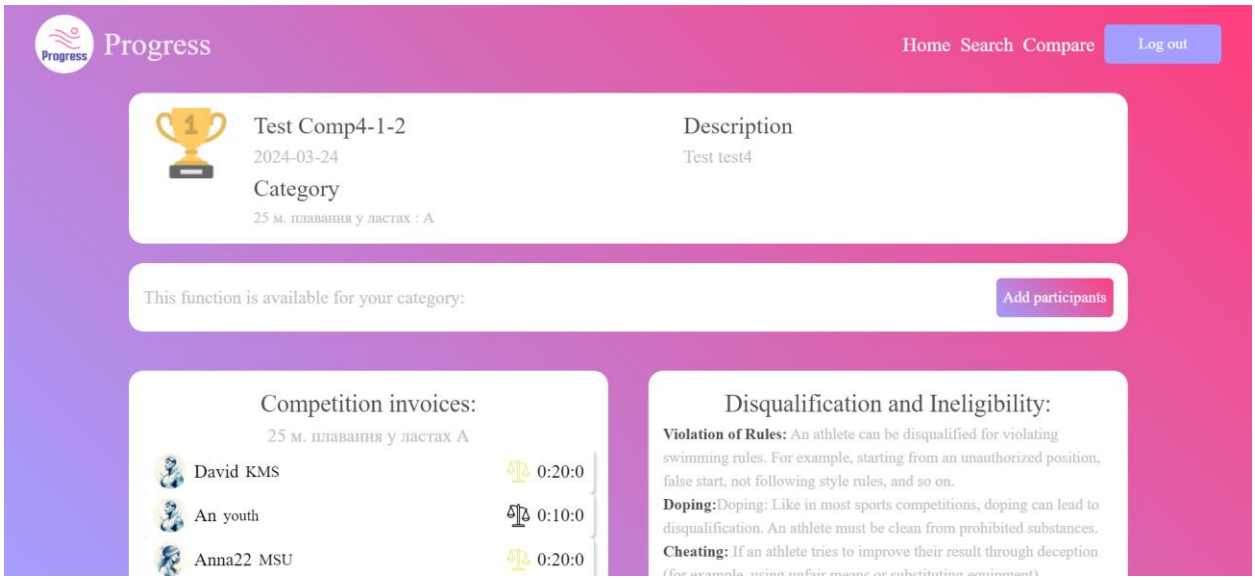


Рисунок 3.13 – Створення заявки

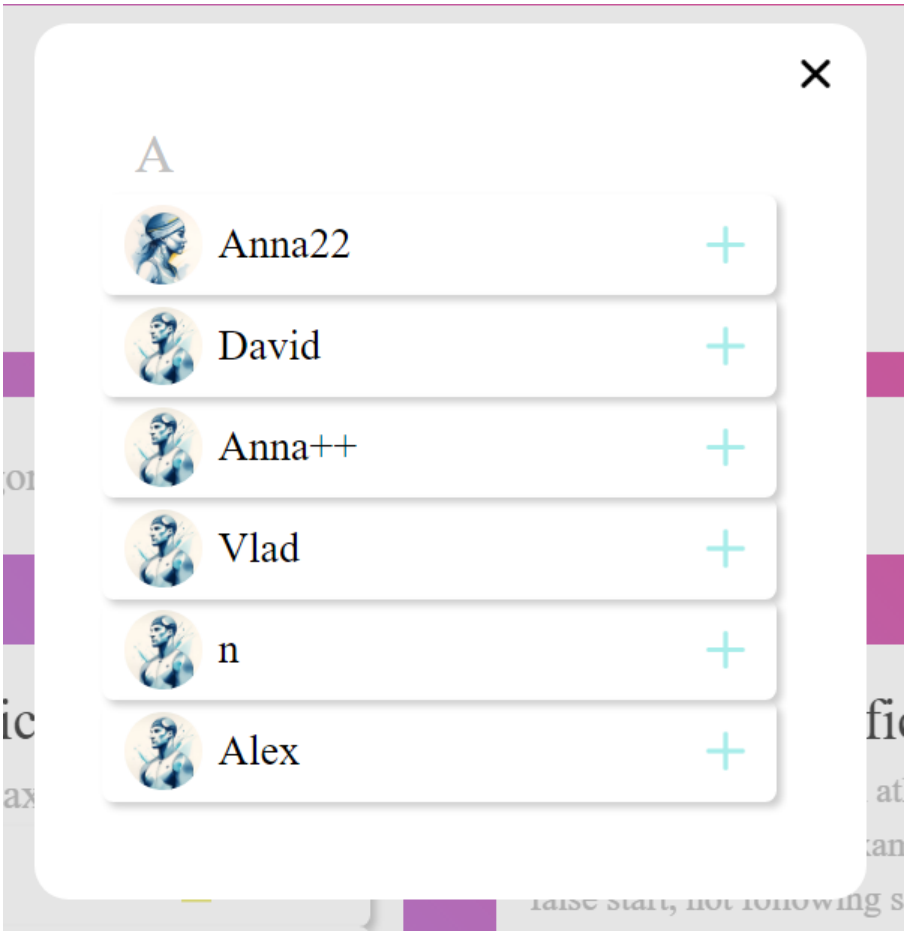


Рисунок 3.14 – Обирання спортсмена

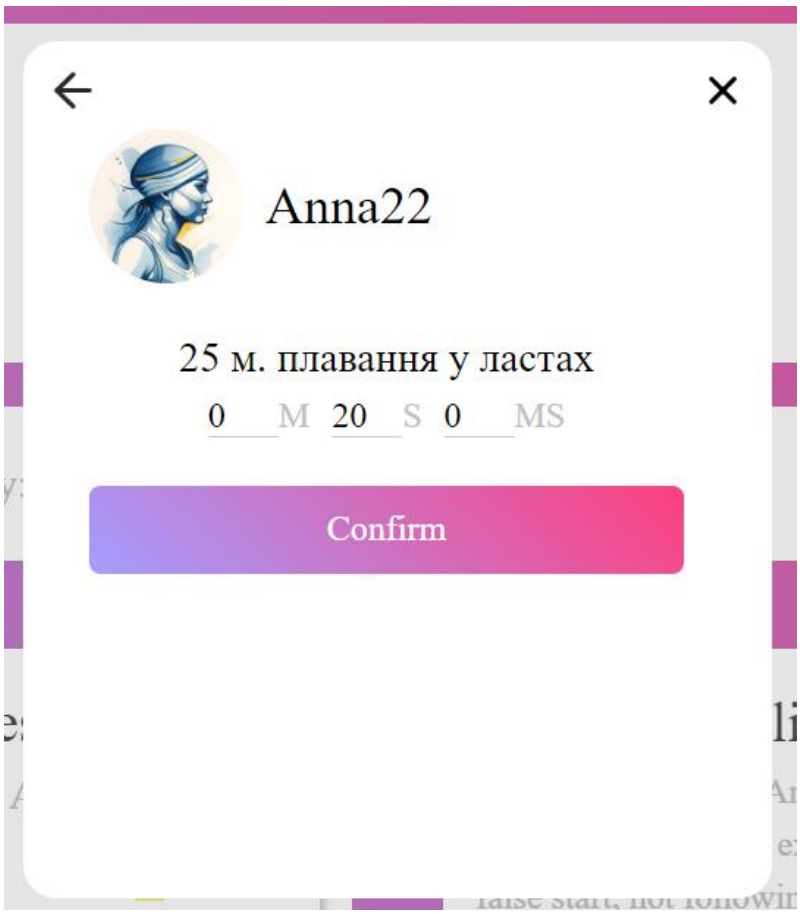


Рисунок 3.15 – Внесення інформації про заявку

**Внесення результатів:** Після переходу на сторінку з потрібним змаганням, для внесення результатів потрібно натиснути на кнопку «Add Results» (рис.3.16), заповнити інформацію про результати (рис. 3.17). Для дискваліфікації учасника, вкажіть час, що дорівнює нулю, натисніть на кнопку «Confirm results». Додавати результати можуть лише користувачі категорії судді.

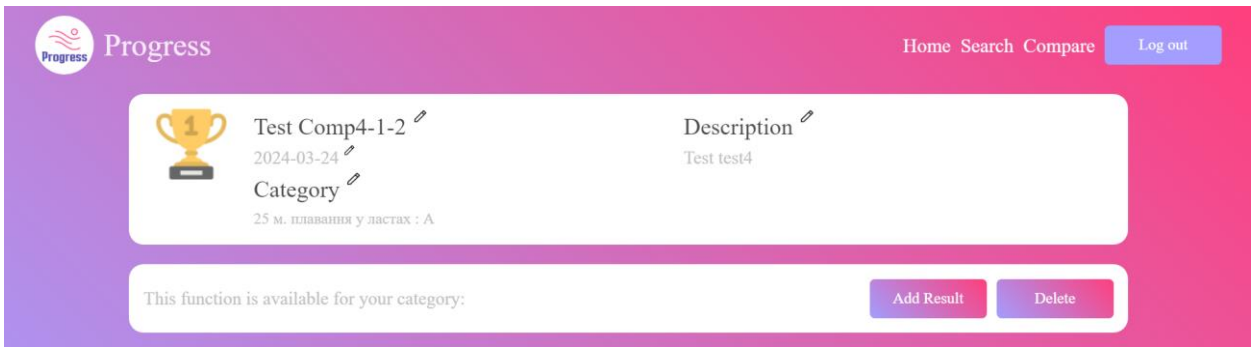


Рисунок 3.16 – Внесення результатів

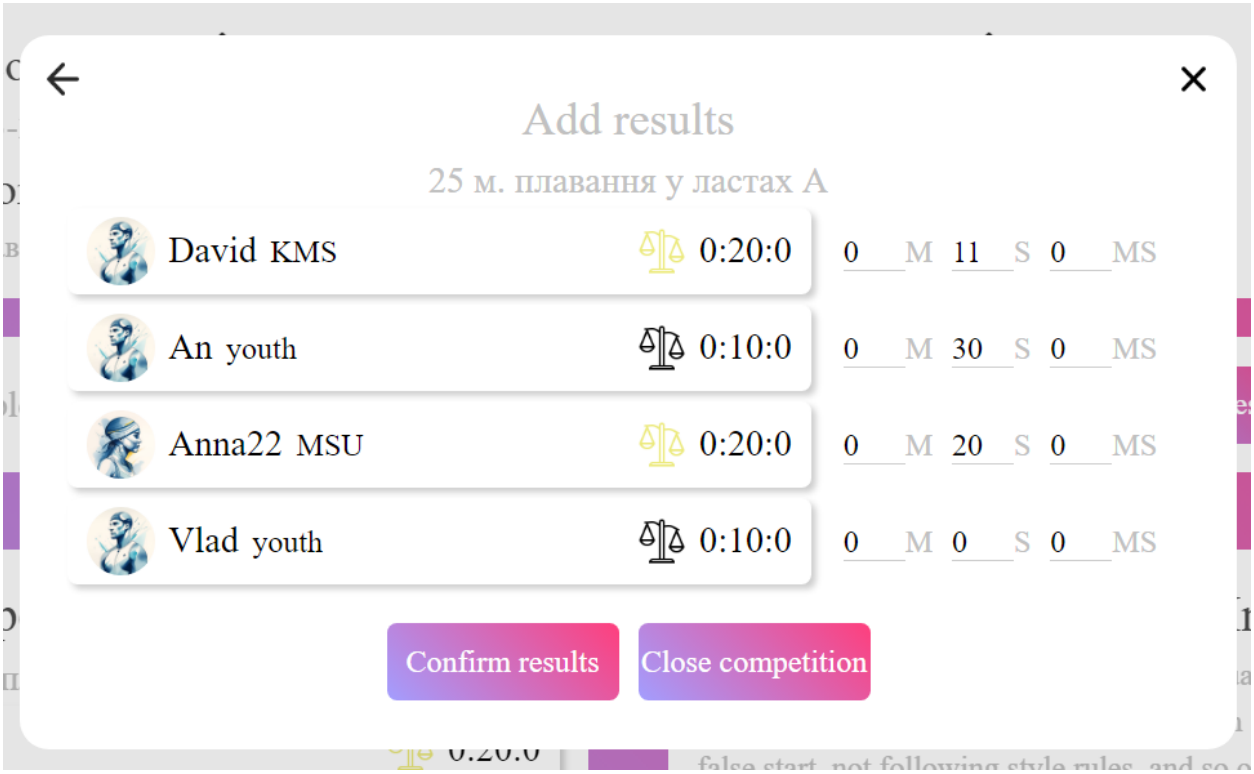


Рисунок 3.17 – Внесення інформації про час

**Закриття змагань:** Після закриття змагань редагування інформації про змагання заборонено. Щоб закрити змагання натисніть на кнопку «Close competition», після чого з’явиться кінцева залікова таблиця, після ознайомлення з нею натисніть на кнопку «Confirm results» (рис. 3.18).

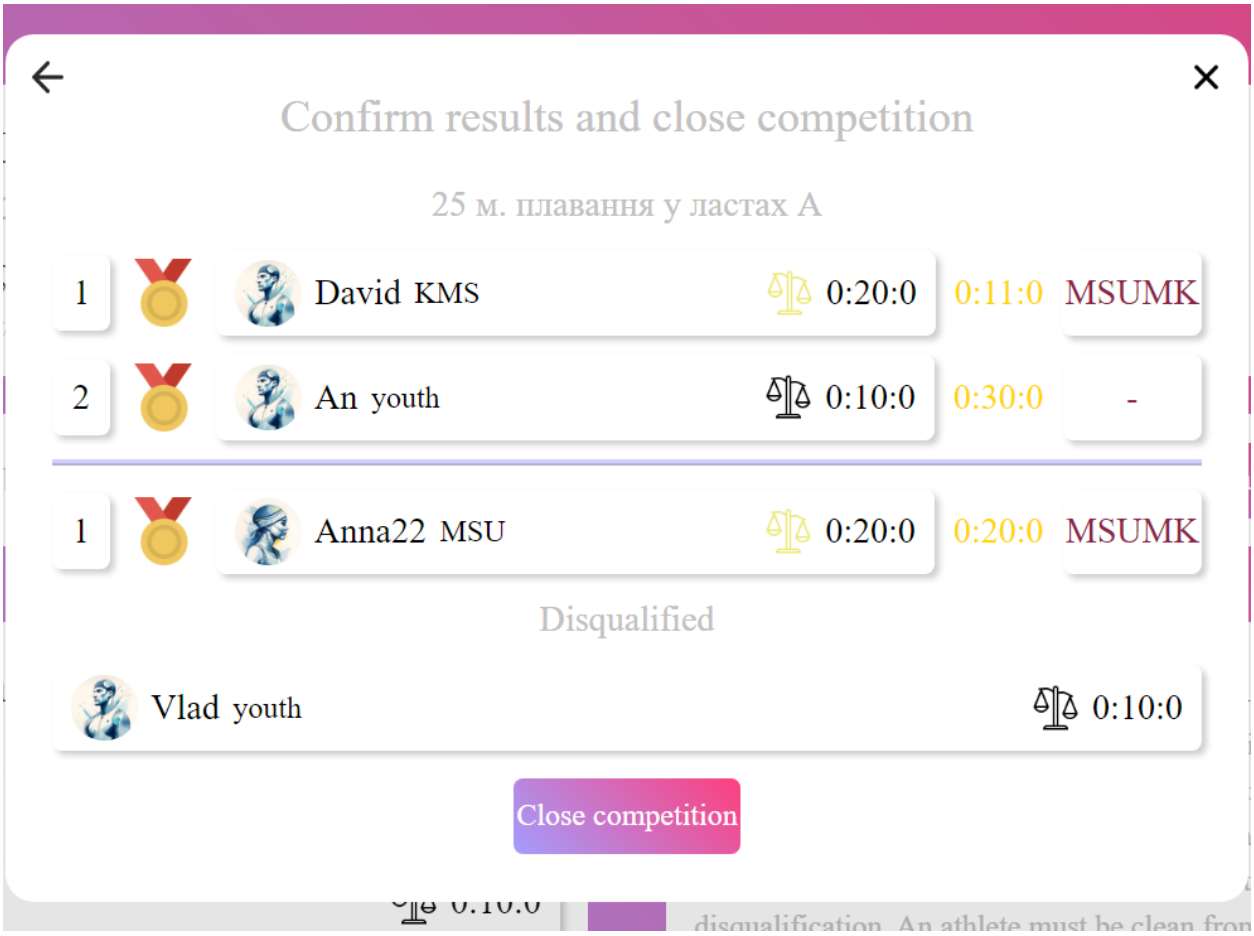


Рисунок 3.18 – Внесення інформації про час

**Перегляд статистики спортсмена:** Для перегляду статистики спортсмена необхідно натиснути на профіль, тоді програма перенаправить користувача на сторінку спортсмена, де він зможе ознайомитись із такими статистичними даними: виконані розряди та здобуті місця, також користувач може обрати рік, за який хоче отримати усю інформацію про результати та змагання, у яких приймав участь спортсмен (рис. 3.19). Користувач має змогу також провести оцінку прогресу спортсмену, скориставшись функцією порівняльного аналізу, обравши дисципліну змагань, для більш детального дослідження користувач може обрати роки, за які він хоче провести аналіз (рис. 3.20).

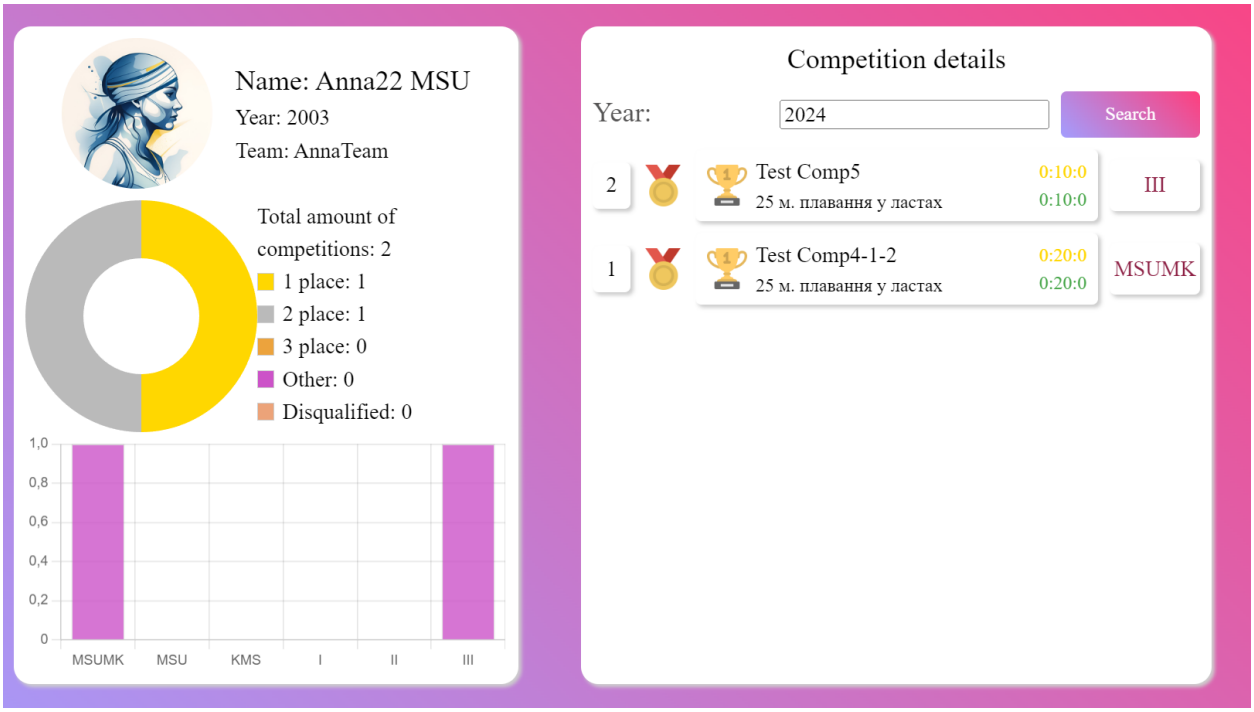


Рисунок 3.19 – Сторінка дослідження результатів спортсмена

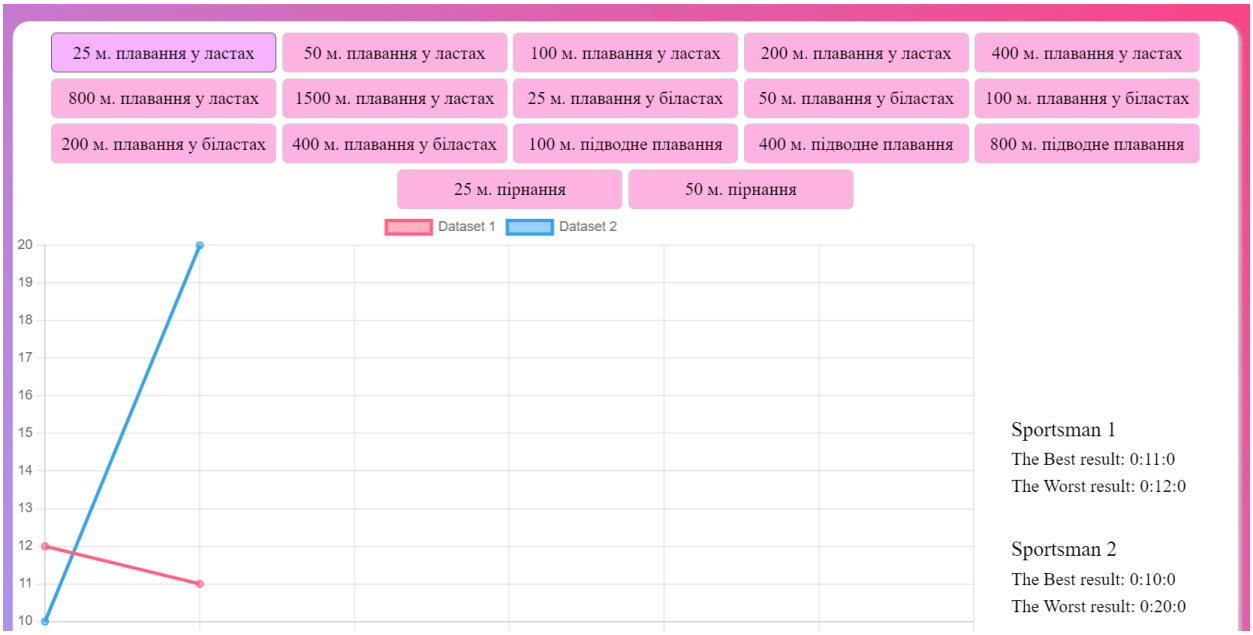


Рисунок 3.20 – Дослідження прогресу за обраною дисципліною

**Перегляд статистики команди:** Для перегляду статистики команди необхідно натиснути на профіль, тоді програма перенаправить користувача на сторінку з інформацією про команду, де він зможе ознайомитись із такими статистичними даними: розряди спортсменів та здобуті місця (рис. 3.21), також користувач може обрати рік, за який хоче отримати усю інформацію

про результати та змагання, у яких приймала участь команда (рис. 3.22), натиснувши на кнопку «More Details», користувач отримає більш розширену інформацію про обране змагання та результати команди у ньому (рис. 3.23).

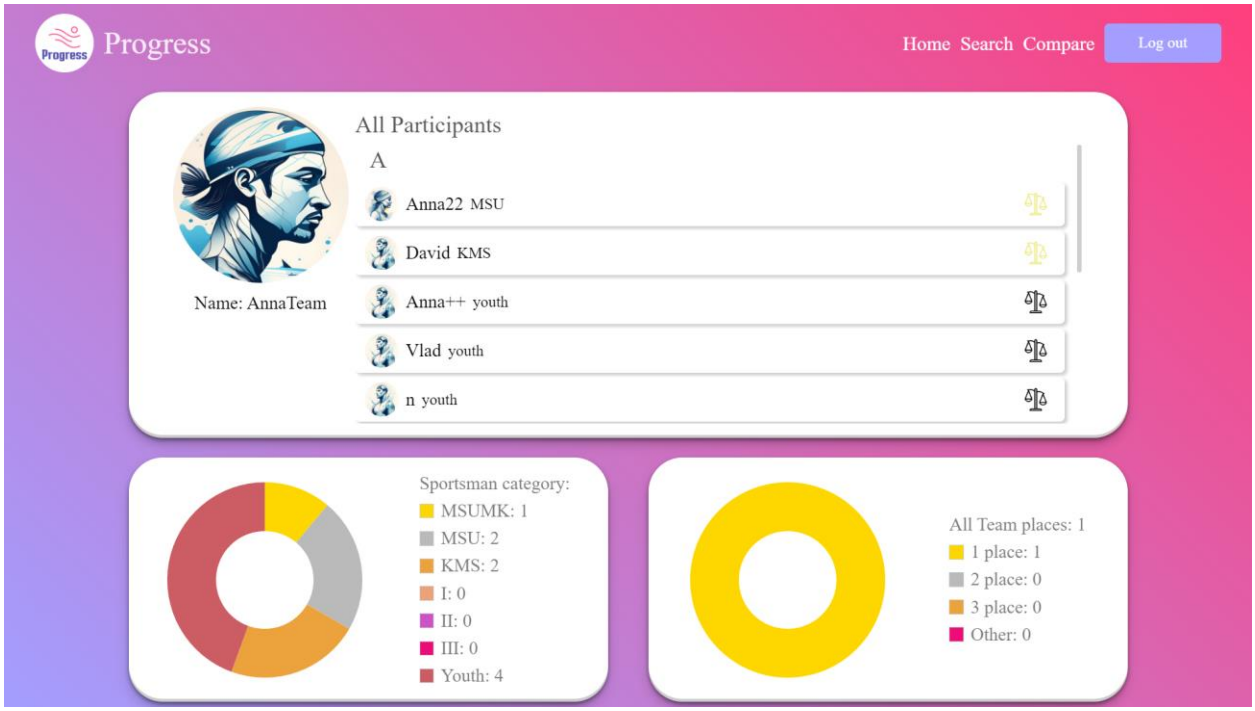


Рисунок 3.21 – Сторінка профілю команди

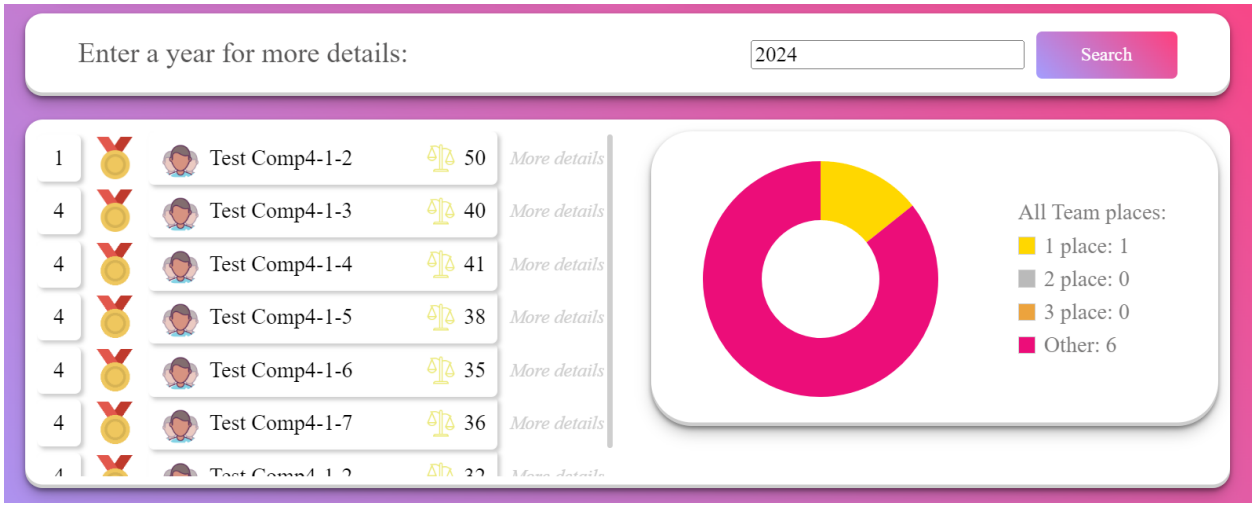


Рисунок 3.22 – Дослідження прогресу команди у обраному році

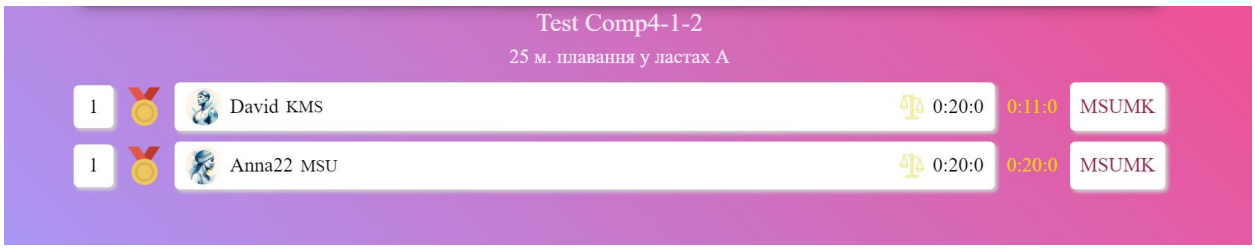


Рисунок 3.23 – Додаткова інформація про результати команди у обраному змаганні

# ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізований застосунок для відстеження прогресу спортсменів швидкісного підводного плавання. Цей застосунок забезпечує усі необхідні функції для різних груп користувачів такі як: створення, керування змаганнями та їх перегляд, внесення результатів, подання заявок та їх редагування, порівняння спортсменів, перегляд статистики команди та спортсмена, керування командою.

Використання сучасних технологій, таких як React та Nest.js, дозволило створити оптимізований та масштабований продукт з легким та інтуїтивно зрозумілим інтерфейсом. Завдяки цьому застосунок є не тільки функціонально насиченим, але й зручним у використанні.

Цінність цієї кваліфікаційної роботи полягає у її практичній користі, оскільки вона надає комплексне рішення для тренерів, спортсменів та суддів швидкісного підводного плавання. Застосунок дозволяє відстежувати досягнення та розвиток спортсменів, що сприяє їхньому професійному росту та підвищенню результативності.

Результати роботи апробовано у вигляді тез доповідей під час 28-го Міжнародного молодіжного форуму «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ» [15].

# ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kornienko, D. V., Mishina, S. V., & Melnikov, M. O. (2021, November). The Single Page Application architecture when developing secure Web services. In Journal of Physics: Conference Series (Vol. 2091, No. 1, p. 012065). IOP Publishing.

2. Kinoshenko, D., Mashtalir, V., Yegorova, E., & Vinarsky, V. (2005). Hierarchical partitions for content image retrieval from large-scale database. In Machine Learning and Data Mining in Pattern Recognition: 4th International Conference, MLDM 2005, Leipzig, Germany, July 9-11, 2005. Proceedings 4 (pp. 445-455). Springer Berlin Heidelberg.

3. Nguyen, H. (2021). Front end architecture for a single page web application.

4. Bodyanskiy Y., Vynokurova O., Kobylin, I., Kobylin O. (2016) Adaptive fuzzy clustering of short time series with unevenly distributed observations in Data Stream Mining tasks, Information Technology and Management Science. pp. 23-28.

5. Tvoroshenko I., and Maksimenko H. (2021) Research of regression and modular testing of web applications, Abstracts of IV international science conference «Science, theory and practice» (October 12-15, 2021). Tokyo, Japan, pp. 406-411.

6. Gorokhovatskyi V., Tvoroshenko I., Kobylin O., and Vlasenko N. (2023) Search for visual objects by request in the form of a cluster representation for the structural image description, Advances in Electrical and Electronic Engineering, 21(1), pp. 19-27.

7. V. Lyashenko, D. Rudenko, (2021) Modeling Deformation of Spur Gear. « International Journal of Recent Technology and Applied Science», vol. 3, no. 2, pp. 81-91, Sep. 2021.

8. Творошенко, І.С. (2021). Технології прийняття рішень в інформаційних системах: навч. посібник. Харків: ХНУРЕ.

9. Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

10. Kobylin, O. A., Vyskrebentseva, S. O., & Petrova, R. V. (2019). Обробка даних, що містять пропуски в задачах кластеризації. Системи управління, навігації та зв'язку. Збірник наукових праць, 5(57), 45-50.

11. Bodyanskiy, Y., Vynokurova, O., Szymański, Z., Kobylin, I., & Kobylin, O. (2016, August). Adaptive robust models for identification of nonstationary systems in data stream mining tasks. In 2016 IEEE First International Conference on Data Stream Mining & Processing (DSMP) (pp. 263-268). IEEE.

12. Guide to Web Application Architecture. URL: https://www.intellectsoft.net/blog/web-application-architecture/ (дата звернення 02.05.2024).

13. Tvoroshenko, I. (2020). Information technologies for decision-making on the conditions of spatially distributed objects. In I International Scientific and Practical Conference. Problems and perspectives of modern science and practice, Austria (pp. 45-50).

14. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення 08.05.2024).

15. Кондратова А. Ю. Визначення трендів часового ряду за допомогою Method of moving averages для дослідження прогресу спортсмена у підводному плаванні: Радіоелектроніка та молодь у XXI столітті: 28 міжнародний молодіжний форум. (Харків 16-18 квітня 2024 р.) Харків 2024. С. 58-60.

16. Pham, A. D. (2020). Developing back-end of a web application with NestJS framework: Case: Integrify Oy's student management system.

17. Siahaan, M., & Vianto, V. O. (2022). Comparative Analysis Study of Front-End JavaScript Frameworks Performance Using Lighthouse Tool. Jurnal Mantik, 6 (3), 2462-2468.

18. React. URL: https://react.dev/ (дата звернення 08.05.2024).

19. React Router. URL: https://reactrouter.com/en/main (дата звернення 08.05.2024).

20. Redux. URL https://react-redux.js.org/ (дата звернення 08.05.2024).

21. Redux Toolkit. URL: https://redux-toolkit.js.org/ (дата звернення 08.05.2024).

22. Nest.js. URL https://nestjs.com/ (дата звернення 08.05.2024).

23. Prisma ORM. URL: https://www.prisma.io/ (дата звернення 08.05.2024).

24. Date-fns. URL: https://date-fns.org/v3.2.0/docs/I18n (дата звернення 08.05.2024).

25. Requirement Analysis Techniques. URL: https://www.visual-paradigm.com/guide/requirements-gathering/ (дата звернення 23.04.2024).

26. Best Practices for Developing with TypeScript in 2023. URL: https://medium.com/@worachote/best-practices-for-developing-with-typescript-in-2023-de88fc6f96a0 (дата звернення 18.04.2024).

27. How to Set Up a Scalable React Architecture: Tips and Best Practices to Follow. URL https://geniusee.com/single-blog/react-architecture-best-practices-and-tips (дата звернення 04.05.2024).

28. Gherkin Syntax. URL: https://cucumber.io/docs/gherkin/ (дата звернення 08.05.2024).

29. Non-Functional Requirements: Examples, Definition, Complete Guide. URL: https://testomat.io/blog/non-functional-requirements-examples-definition-complete-guide/ (дата звернення 02.05.2024).

30. Functional vs Non Functional Requirements. URL: https://testomat.io/blog/non-functional-requirements-examples-definition-complete-guide/ (дата звернення 18.04.2024).