ВІЙСЬКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ ТА ІНФОРМАТИЗАЦІЇ імені ГЕРОЇВ КРУТ

Інформаційних технологій

(назва факультету)

Комп’ютерних інформаційних технологій

(повна назва кафедри)

не таємно гриф

**Пояснювальна записка**

до кваліфікаційної роботи

бакалавр .

(освітньо-кваліфікаційний рівень)

на тему: “Онлайн-інструмент для моніторингу стану техніки”

Виконав: курсант 4 курсу, групи 403 напряму підготовки (спеціальності) 122 Комп'ютерні науки та . інформаційні технології ю

(шифр і назва напряму підготовки, спеціальності)

Лутцев Я.Р. .

(прізвище та ініціали)

Керівник Редзюк Є.В. .

(прізвище та ініціали)

Рецензент Сілко О.В. .

(прізвище та ініціали)

Київ-2025

# АНОТАЦІЯ

кваліфікаційної роботи бакалавра на тему:

“ Програмний модуль автоматизації обліку спортивних заходів ЗВО”

Кваліфікаційна робота бакалавра містить: 72 сторінки, 24 рисунків, 19 джерела.

У роботі розглянуто актуальні завдання та підходи до автоматизації процесів обліку та управління спортивними заходами та фізичною підготовкою у закладах вищої освіти, зокрема у військових вищих навчальних закладах. Проведено аналіз потреби в сучасних цифрових інструментах для підвищення ефективності планування, проведення та моніторингу спортивних заходів та нормативів. Визначено ключові архітектурні рішення для розробки програмного модуля, спроєктовано структуру бази даних для зберігання інформації про користувачів, підрозділи, особовий склад, фізичні вправи, місця проведення, тренувальні заняття та результати здачі нормативів. Обґрунтовано принципи забезпечення інформаційної безпеки в розроблюваному програмному модулі. Розроблено програмний модуль, що забезпечує автоматизований збір, обробку, зберігання та аналіз даних про спортивні заходи та фізичну підготовку. Модуль включає функціонал для ведення обліку тренувальних занять, реєстрації особового складу та їхніх результатів, управління довідниками вправ та місць проведення, а також формування звітності та аналітики.

Ключові слова: автоматизація, облік спортивних заходів, фізична підготовка, програмний модуль, ЗВО, база даних, веб-додаток, Node.js, React.

ABSTRACT

of the qualification paper of bachelor on theme:

“ Software module for automation of accounting of sports events in Higher Education Institutions”

The bachelor's qualification paper contains: 72 pages, 24 pictures, 19 sources.

The work considers current tasks and approaches to the automation of accounting and management of sports events and physical training in higher education institutions, particularly in military higher education institutions (MHEIs). An analysis of the need for modern digital tools to improve the efficiency of planning, conducting, and monitoring sports events and standards has been carried out. Key architectural solutions for the development of the software module were identified, the database structure for storing information about users, units, personnel, exercises, locations, training sessions, and assessment results was designed. The principles of ensuring information security in the developed software module were substantiated. A software module has been developed that provides automated collection, processing, storage, and analysis of data on sports events and physical training. The module includes functionality for managing training sessions, registering personnel and their results, managing directories of exercises and locations, as well as generating reports and analytics.

Keywords: automation, accounting of sports events, physical training, software module, MHEI, database, web application, Node.js, React.

ЗМІСТ

[АНОТАЦІЯ 2](#_Toc199151297)

[ЗМІСТ 4](#_Toc199151298)

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 7](#_Toc199151299)

[ВСТУП 9](#_Toc199151300)

[РОЗДІЛ 1. ОБҐРУНТУВАННЯ НЕОБХІДНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ СПОРТИВНИХ ЗАХОДІВ ЗВО 11](#_Toc199151301)

[1.1. Аналіз нормативно-правового регулювання та існуючої практики моніторингу технічного стану техніки у Збройних Силах України. 12](#_Toc199151302)

[1.2. Аналіз існуючих підходів та засобів автоматизації обліку спортивних заходів та фізичної підготовки в Україні 20](#_Toc199151303)

[1.3. Аналіз світового досвіду щодо автоматизації процесів обліку спортивних заходів у закладах освіти 23](#_Toc199151304)

[1.4. Технічне завдання на проектування 26](#_Toc199151305)

[Висновок до Розділу 1 28](#_Toc199151306)

[РОЗДІЛ 2. ТЕОРЕТИЧНІ ЗАСАДИ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ СПОРТИВНИХ ЗАХОДІВ 30](#_Toc199151307)

[2.1. Обґрунтування вибору архітектури онлайн-інструменту та розробка її узагальненої структурної схеми. 31](#_Toc199151310)

[2.2. Проектування бази даних програмного модулю автоматизації обліку спортивних заходів ЗВО. 36](#_Toc199151311)

[2.3. Принципи та методи забезпечення безпеки інформації в програмному модулі. 39](#_Toc199151312)

[Висновок до Розділу 2 41](#_Toc199151313)

[РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ СПОРТИВНИХ ЗАХОДІВ 43](#_Toc199151314)

[3.1. Обґрунтування вибору інструментальних засобів та технологій для розробки програмного модулю 44](#_Toc199151315)

[3.2. Розробка програмного модулю. 49](#_Toc199151316)

[3.2.1. Реалізація серверної логіки та прикладного програмного інтерфейсу. 49](#_Toc199151317)

[3.2.2. Реалізація клієнтської частини та користувацьких інтерфейсів. 51](#_Toc199151318)

[3.2.3. Розробка та інтеграція модуля управління доступом та розмежування прав користувачів. 60](#_Toc199151319)

[3.3. Розгортання онлайн-інструменту. 63](#_Toc199151320)

[Висновок до Розділу 3 66](#_Toc199151321)

[ВИСНОВКИ 69](#_Toc199151322)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 72](#_Toc199151323)

[ДОДАТОК А 74](#_Toc199151324)

[TrainingSession 74](#_Toc199151325)

[StandardAssessment 75](#_Toc199151326)

[SessionExercise 77](#_Toc199151327)

[MilitaryPersonnel 78](#_Toc199151328)

[StandardAssessment 79](#_Toc199151329)

[Exercise 81](#_Toc199151330)

[User.js 82](#_Toc199151331)

[trainingSessionService.js 82](#_Toc199151332)

[standardAssessmentService.js 86](#_Toc199151333)

[authController.js 89](#_Toc199151334)

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АІС – Автоматизована інформаційна система

API – Прикладний програмний інтерфейс (Application Programming Interface)

БД – База даних

ВВНЗ – Військовий вищий навчальний заклад

ЗВО – Заклад вищої освіти

ЗСУ – Збройні Сили України

ІС – Інформаційна система

ІТ – Інформаційні технології

МОН – Міністерство освіти і науки України

МОУ – Міністерство оборони України

НД – Нормативний документ

ПЗ – Програмне забезпечення

РБД – Реляційна база даних

СУБД – Система управління базами даних

UI – Користувацький інтерфейс (User Interface)

UX – Досвід користувача (User Experience)

CRUD – Створення, Читання, Оновлення, Видалення (Create, Read, Update, Delete)

JWT – JSON Web Token

RBAC – Role-Based Access Control (Управління доступом на основі ролей)

SPA – Односторінковий застосунок (Single Page Application)

ВСТУП

Актуальність теми. Ефективна організація та контроль фізичної підготовки є невід'ємною складовою навчального процесу у закладах вищої освіти, особливо у військових вищих навчальних закладах, де рівень фізичної підготовленості особового складу безпосередньо впливає на їхню професійну спроможність та боєздатність. Традиційні методи обліку спортивних заходів, результатів здачі нормативів та планування тренувань часто базуються на паперовому документообігу, що призводить до значних витрат часу, ускладнює аналіз даних та оперативне прийняття управлінських рішень. Впровадження сучасних інформаційних технологій для автоматизації цих процесів є ключовим фактором підвищення їх ефективності. Розробка спеціалізованого програмного модуля дозволить оптимізувати процеси планування, обліку, моніторингу та аналізу даних про спортивні заходи та фізичну підготовку, забезпечуючи швидкий доступ до актуальної інформації для всіх зацікавлених сторін – від курсантів та інструкторів до керівництва навчальних підрозділів та закладу в цілому. Це, в свою чергу, сприятиме покращенню якості фізичної підготовки та більш ефективному використанню наявних ресурсів.

Мета роботи: розробка програмного модуля для автоматизації обліку спортивних заходів та фізичної підготовки у закладі вищої освіти, що передбачає теоретичне обґрунтування його структури та функціоналу, проектування ключових компонентів та практичну реалізацію програмного рішення.

Виходячи з мети роботи, виникають наступні завдання:

1. Обґрунтувати необхідність розробки програмного модуля шляхом аналізу існуючої практики обліку спортивних заходів у ЗВО (ВВНЗ), виявлення недоліків ручного ведення обліку та вивчення потреби в автоматизації.
2. Проаналізувати існуючі підходи до автоматизації подібних процесів та світовий досвід використання інформаційних систем у сфері фізичної підготовки та спорту в освітніх закладах.
3. Розробити концептуальну архітектуру програмного модуля, спроєктувати структуру його бази даних для зберігання інформації про користувачів, особовий склад, підрозділи, тренувальні заняття, вправи, місця проведення та результати оцінювання.
4. Визначити основні принципи та методи забезпечення безпеки інформації в системі, включаючи автентифікацію та авторизацію користувачів.
5. Здійснити програмну реалізацію основних функціональних модулів, включаючи серверну частину з API, клієнтську частину з користувацькими інтерфейсами для управління даними та їх візуалізації.
6. Продемонструвати готовність розробленого програмного модуля до розгортання та використання.

Об’єкт досліджень: процеси планування, організації, проведення, обліку та контролю спортивних заходів та фізичної підготовки у закладі вищої освіти.

Предмет досліджень: моделі, методи та програмні засоби для створення програмного модуля автоматизації обліку спортивних заходів, включаючи його архітектуру, компоненти бази даних, програмні модулі, користувацькі інтерфейси та механізми захисту інформації.

РОЗДІЛ 1. ОБҐРУНТУВАННЯ НЕОБХІДНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ СПОРТИВНИХ ЗАХОДІВ ЗВО

Фізична підготовка та спортивно-масова робота відіграють ключову роль у формуванні професійних якостей майбутніх фахівців у закладах вищої освіти, набуваючи особливої значущості у військових вищих навчальних закладах. Ефективність цього напряму діяльності значною мірою залежить від якості планування, організації та контролю, що, своєю чергою, вимагає сучасних підходів до збору, обробки та аналізу відповідної інформації.

Незважаючи на високі вимоги до рівня фізичної підготовленості, у багатьох ЗВО, включно з ВВНЗ, процеси обліку спортивних заходів, фіксації результатів виконання нормативів та моніторингу індивідуальних показників особового складу часто залишаються недостатньо автоматизованими. Переважне використання паперового документообігу та ручних методів обробки даних не лише збільшує трудомісткість роботи відповідальних осіб, але й обмежує можливості для оперативного аналізу та прийняття обґрунтованих управлінських рішень.

Даний розділ присвячений детальному розгляду передумов, що зумовлюють актуальність створення спеціалізованого програмного модуля. Буде здійснено аналіз чинної нормативно-правової бази, що регулює питання фізичної підготовки, досліджено поточну практику обліку спортивних заходів у ЗВО (ВВНЗ), а також вивчено існуючі вітчизняні та зарубіжні підходи до автоматизації подібних завдань. Результатом цього аналізу стане чітке обґрунтування доцільності розробки програмного інструменту, спрямованого на модернізацію та підвищення ефективності управління фізичною підготовкою.

* 1. Аналіз нормативно-правового регулювання та існуючої практики моніторингу технічного стану техніки у Збройних Силах України.

Основою для організації фізичної підготовки та спортивно-масової роботи у закладах вищої освіти (ЗВО), а надто у військових вищих навчальних закладах (ВВНЗ) та військових навчальних підрозділах ЗВО (ВНП ЗВО), слугує чинне законодавство України та низка спеціалізованих нормативно-правових актів. Ці документи встановлюють не лише загальні рамки освітньої діяльності, а й конкретні вимоги до фізичного стану здобувачів освіти, що є критично важливим для майбутніх офіцерів та фахівців сектору безпеки і оборони. Загальні принципи освітньої діяльності, включаючи фізичне виховання як її невід'ємну частину, закріплені в Законах України "Про освіту", "Про вищу освіту" та "Про фізичну культуру і спорт". Для ВВНЗ та ВНП ЗВО додатково визначальними є положення Закону України "Про військовий обов'язок і військову службу" та статути Збройних Сил України, які підкреслюють важливість високого рівня фізичної підготовленості для виконання службових обов'язків.

Ключовим документом, що визначає специфіку освітнього процесу у військових навчальних закладах, є Наказ Міністерства оборони України № 120 від 15.02.2024 «Про затвердження Положення про особливості організації освітнього процесу у вищих військових навчальних закладах Міністерства оборони України, військових навчальних підрозділах закладів вищої освіти, закладах фахової передвищої військової освіти». Це Положення наголошує на компетентнісному підході до навчання, що передбачає формування у здобувачів освіти не лише теоретичних знань та практичних умінь, а й здатності їх ефективного застосування у професійній діяльності, невід'ємною частиною якої є належна фізична підготовленість.

Основним документом, що безпосередньо регулює всі аспекти фізичної підготовки особового складу Збройних Сил України, включаючи курсантів (слухачів) ВВНЗ, є Інструкція з фізичної підготовки в системі Міністерства оборони України, затверджена Наказом Міністерства оборони України від 05.08.2021 № 225 (далі – Інструкція № 225). Ця Інструкція визначає систему фізичної підготовки, форми, зміст і порядок її організації та спортивної діяльності. Її вимоги поширюються, зокрема, на кандидатів для вступу у ВВНЗ, ВНП ЗВО, ЗФПВО, а також на студентів цих закладів.

Метою фізичної підготовки є забезпечення фізичної готовності військовослужбовців до бойової діяльності та виконання службових завдань. До загальних завдань належать розвиток фізичних та спеціальних якостей, оволодіння військово-прикладними руховими навичками, виховання морально-вольових якостей, підвищення стійкості організму до несприятливих факторів та зміцнення здоров'я.

Інструкція встановлює поділ військовослужбовців на категорії та вікові групи для організації навчального процесу та перевірки фізичної підготовки. Визначено три категорії військовослужбовців залежно від специфіки їхньої служби. Курсанти ВВНЗ, ВНП ЗВО, ЗФПВО, які готують фахівців для розвідувальних частин, Десантно-штурмових військ, Сил спеціальних операцій, Морської піхоти, належать до першої категорії. Курсанти інших спеціальностей ВВНЗ, ВНП ЗВО, ЗФПВО належать переважно до другої категорії. Для військовослужбовців встановлено дев'ять вікових груп: перша – до 25 років, друга – 25-29 років, третя – 30-34 роки, і так далі до дев'ятої – 60 років і старші.

Організація фізичної підготовки включає планування, підготовку керівників, контроль та облік. Плануванням визначається час на навчальні заняття, спортивно-масову роботу, ранкову фізичну зарядку та самостійну підготовку. Наприклад, для курсантів ВВНЗ, ВНП ЗВО, ЗФПВО передбачено не менше 4 годин навчальних занять на тиждень у розкладі, не менше 5 годин на спортивно-масову роботу та щоденну ранкову фізичну зарядку тривалістю 30-50 хвилин. Тривалість навчальних занять з фізичного виховання та спеціальної фізичної підготовки для курсантів становить 2 академічні години (90 хв).

Навчальні заняття з фізичної підготовки є основною формою і поділяються на теоретичні та практичні. Практичні заняття проводяться за розділами (рис.1.1), наприклад, прискорене пересування та легка атлетика; гімнастика і атлетична підготовка; подолання перешкод і метання гранат; військово-прикладне плавання та веслування; рукопашний бій; лижна підготовка; спортивні та рухливі ігри або комплексно. Навчально-тренувальні заняття мають три частини: підготовчу (7-10 хв для одногодинного, 10-25 хв для двогодинного), основну (30-35 хв або 65-85 хв відповідно) та заключну (3-5 хв або 5-10 хв). Особливості проведення занять з кожного розділу фізичної підготовки детально описані в керівних документах.

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним. Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним. Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним.

Рисунок 1.1 Приклад структури Плана комплексного навчально-тренувального заняття з фізичної підготовки

Перевірка та оцінка фізичної підготовки є важливим елементом управління. Вона проводиться для визначення рівня фізичної готовності військовослужбовців та стану фізичної підготовки у військовій частині або ВВНЗ. Перевірки здійснюються за планами, затвердженими командирами, і включають виконання вправ, що характеризують основні фізичні та спеціальні якості, військово-прикладні навички. Інструкцією № 225 визначено конкретні комплекси контрольних вправ для різних категорій та вікових груп (рис. 1.2), а також порядок нарахування балів та визначення оцінки (рис. 1.3). Наприклад, кандидати на вступ до ВВНЗ (чоловіки до 40 років) перевіряються з підтягування на перекладині, бігу на 100 м та бігу на 3 км. Оцінювання виконання фізичних вправ проводиться за сумою балів, що нараховуються з урахуванням статі та віку.

Зображення, що містить текст, знімок екрана, Шрифт, квитанція

Вміст, створений ШІ, може бути неправильним.

Зображення, що містить текст, знімок екрана, число, Шрифт

Вміст, створений ШІ, може бути неправильним.

Рисунок 1.2 Таблиця нормативів для різних вікових груп

Зображення, що містить текст, Паралель, квитанція, схема

Вміст, створений ШІ, може бути неправильним.

Рисунок 1.3 Таблиця нарахування балів за виконання фізичних вправ для вступу у ВВНЗ, ВНП ЗВО, ЗФПВО

Облік фізичної підготовки передбачає збирання, систематизацію, узагальнення та збереження інформації про функціонування системи фізичної підготовки та її результатів. У військовій частині (ВВНЗ) облік ведеться відповідними посадовими особами. За результатами навчального року складається довідка-доповідь про стан фізичної підготовки і спорту (рис.1.4). Документація, що розробляється у військовій частині, включає програму фізичної підготовки, наказ про організацію фізичної підготовки та спортивно-масової роботи, плани спортивних заходів, розклади занять тощо.

Зображення, що містить текст, знімок екрана, Шрифт, число

Вміст, створений ШІ, може бути неправильним.

Рисунок 1.4 Довідка-доповідь про стан фізичної підготовки і спорту у військовій частині за навчальний рік

Незважаючи на детальну регламентацію процесів планування, проведення, контролю та обліку фізичної підготовки в Інструкції № 225, на практиці у багатьох ЗВО та ВВНЗ ці процеси все ще значною мірою покладаються на паперові форми. Ведення журналів обліку занять (наприклад, журнал обліку результатів виконання вправ ВСК – форма № 13 додатка 3 ), відомостей здачі нормативів, протоколів змагань та ручне складання звітів є трудомістким процесом, що не позбавлений ризику помилок та ускладнює оперативний аналіз великих масивів даних. Це створює значні перешкоди для ефективного управління фізичним вдосконаленням особового складу та підкреслює актуальність розробки сучасних автоматизованих систем обліку спортивних заходів.

* 1. Аналіз існуючих підходів та засобів автоматизації обліку спортивних заходів та фізичної підготовки в Україні

Незважаючи на чітко визначені нормативними документами вимоги щодо організації, контролю та обліку фізичної підготовки, реальна практика у багатьох закладах вищої освіти України, включаючи військові вищі навчальні заклади та підрозділи Збройних Сил України, демонструє значне відставання у впровадженні сучасних автоматизованих систем для цих потреб. Існуючі підходи часто є застарілими та характеризуються високою часткою ручної праці.

Поширеною практикою залишається використання загальнодоступних офісних програмних продуктів. Насамперед, це стосується табличних процесорів, таких як Microsoft Excel. На кафедрах фізичного виховання, у спортивних клубах та у відповідних службах військових частин Excel часто використовується для ведення різноманітних списків: особового складу, розподілу по навчальним групам чи спортивним секціям, обліку відвідуваності занять (рис. 1.5). Також у таблицях фіксуються результати здачі нормативів, підсумки змагань, складаються графіки тренувань та розподілу навантаження. Безумовною перевагою є широка доступність цього інструменту та відносна простота його освоєння персоналом. Однак, такий підхід має критичні недоліки при спробі побудувати комплексну систему обліку. Дані в окремих файлах Excel є розрізненими, схильними до дублювання та виникнення неузгодженостей. Контроль версій файлів, забезпечення цілісності та конфіденційності інформації, а також надання розмежованого доступу різним категоріям користувачів (наприклад, інструкторам, курсантам, керівникам підрозділів) стає вкрай ускладненим. Формування складних аналітичних звітів, що вимагають агрегації даних з багатьох джерел, перетворюється на тривалий та трудомісткий процес, схильний до помилок.

Зображення, що містить текст, Шрифт, ряд, число

Вміст, створений ШІ, може бути неправильним.

Рисунок 1.5 Практичне використання Microsoft Excel для обліку здачі нормативів у ЗВО

Подекуди для більш структурованого зберігання даних використовуються локальні системи управління базами даних, наприклад, Microsoft Access (рис. 1.6). Розробка таких баз даних часто є ініціативою окремих викладачів чи співробітників, які мають відповідні навички. Ці системи можуть пропонувати більш зручні форми для введення даних та генерації простих запитів порівняно з Excel. Проте, вони здебільшого залишаються локальними, не інтегрованими в загальну інформаційну інфраструктуру навчального закладу чи військової частини. Відсутність централізованої підтримки, оновлення та масштабування робить такі рішення вразливими та недовговічними. Крім того, функціонал таких саморобних систем рідко виходить за межі базових операцій і не покриває всіх потреб сучасного управління фізичною підготовкою, таких як гнучке планування, автоматизований розрахунок рейтингів, візуалізація прогресу тощо.

Зображення, що містить текст, Шрифт, число, знімок екрана

Вміст, створений ШІ, може бути неправильним.

Рисунок 1.6 Практичне використання Microsoft Access для обліку здачі нормативів у ЗВО

В контексті Збройних Сил України, процеси цифровізації активно розвиваються, проте фокус часто спрямований на системи оперативного управління, логістики чи кадрового обліку. Спеціалізовані автоматизовані системи для детального моніторингу та аналізу саме фізичної підготовленості та спортивних досягнень на рівні окремих ВВНЗ чи військових частин є радше винятком, ніж правилом. Загальновійськові системи обліку можуть містити лише узагальнені дані про стан фізичної підготовки, але не надають інструментів для оперативного щоденного обліку тренувальних навантажень, індивідуальних результатів виконання вправ, планування занять з урахуванням специфіки підрозділу чи індивідуальних потреб військовослужбовців. Відсутність таких систем призводить до того, що інформація, зібрана на місцях у паперовому чи табличному вигляді, важко піддається узагальненню та аналізу на вищих рівнях управління.

Проблема поглиблюється тим, що потреби різних категорій користувачів – від індивідуального курсанта, який бажає відстежувати свій прогрес, інструктора, який планує заняття для групи, до керівника кафедри чи начальника фізичної підготовки, якому потрібні узагальнені дані для звітності та прийняття рішень – суттєво відрізняються. Задовольнити ці різноманітні потреби за допомогою універсальних офісних інструментів або розрізнених локальних баз даних практично неможливо.

Отже, аналіз існуючих підходів до автоматизації обліку спортивних заходів та фізичної підготовки в Україні свідчить про значний розрив між наявними інструментами та реальними потребами ЗВО, ВВНЗ та ЗСУ. Відсутність комплексних, інтегрованих та спеціалізованих програмних рішень стримує підвищення ефективності управління фізичною підготовкою, ускладнює аналітичну роботу та не дозволяє повною мірою використовувати потенціал сучасних інформаційних технологій для покращення цього важливого аспекту підготовки фахівців.

* 1. Аналіз світового досвіду щодо автоматизації процесів обліку спортивних заходів у закладах освіти

Автоматизація управлінських та освітніх процесів є глобальною тенденцією, що активно впроваджується у закладах вищої освіти по всьому світу. Сфера фізичного виховання та спорту не є виключенням, адже сучасні інформаційні технології надають широкі можливості для підвищення ефективності планування тренувального процесу, моніторингу фізичної підготовленості та управління спортивними заходами. Передові університети та військові академії різних країн активно використовують спеціалізовані програмні рішення для цих потреб.

У багатьох закордонних університетах функціонують комплексні системи управління студентською інформацією (Student Information Systems - SIS) або системи управління навчанням (Learning Management Systems - LMS), які часто включають модулі для адміністрування спортивної діяльності. Такі модулі дозволяють студентам реєструватися на спортивні курси та секції, переглядати розклад занять та змагань, а викладачам та тренерам – вести облік відвідуваності, фіксувати результати виконання нормативів та спортивні досягнення. Прикладом можуть слугувати рішення, що інтегруються з популярними платформами як Canvas (рис. 1.7) або Moodle, де створюються окремі курси або розділи для фізичного виховання, що дозволяє централізовано управляти навчальним контентом, завданнями та оцінюванням. Візуалізація прогресу студентів через особисті кабінети та можливість генерувати аналітичні звіти для адміністрації є стандартними функціями таких систем.

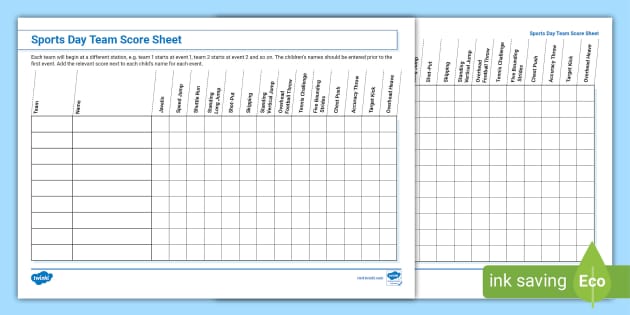


Рисунок 1.7 Приклад використання системи Canvas

Окремий напрямок складають спеціалізовані платформи для управління спортивними командами та клубами при університетах. Системи, такі як TeamSnap, SportsEngine (рис. 1.8), TeamSideline, орієнтовані на потреби тренерів та менеджерів команд. Вони надають інструменти для складання розкладу тренувань та ігор, управління складом команди, комунікації з гравцями та батьками, збору внесків, а також відстеження статистики виступів. Хоча такі системи більше сфокусовані на змагальній діяльності, їхні принципи організації даних та користувацьких інтерфейсів можуть бути корисними при розробці модуля для обліку фізичної підготовки.

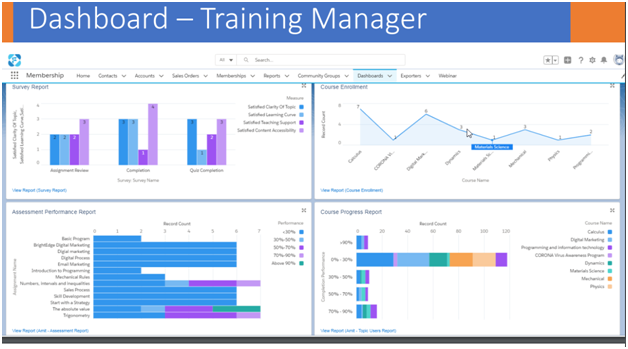


Рисунок 1.8 Приклад використання SportsEngine

У військових навчальних закладах країн-членів НАТО та інших розвинених держав приділяється значна увага фізичній готовності особового складу, і, відповідно, системам її моніторингу. Наприклад, в армії США використовуються елементи, що дозволяють відстежувати показники фізичної підготовленості військовослужбовців в рамках їхніх загальних систем управління персоналом та бойовою підготовкою. Так, Army Physical Fitness Test, а згодом Army Combat Fitness Test, вимагають ретельної фіксації результатів. Хоча не завжди існують єдині всеохоплюючі системи, доступні для публічного огляду, тенденція йде до створення інтегрованих баз даних, де показники фізичної підготовленості пов'язуються з іншими даними про військовослужбовця. Такі системи дозволяють аналізувати загальний рівень підготовки підрозділів, виявляти індивідуальні потреби в додаткових тренуваннях та планувати програми фізичного вдосконалення. Наприклад, платформи, подібні до тих, що використовуються для управління тренувальним процесом у професійному спорті, адаптуються для потреб військових, дозволяючи створювати індивідуалізовані плани тренувань та відстежувати прогрес.

Загальною рисою для закордонних систем є їхня спрямованість на надання зручних веб-інтерфейсів та мобільних додатків для всіх категорій користувачів. Це забезпечує оперативний доступ до інформації, спрощує введення даних та покращує комунікацію між студентами, викладачами та адміністрацією. Важливим аспектом є також забезпечення аналітичних можливостей: формування звітів, візуалізація даних у вигляді графіків та діаграм, що допомагає приймати обґрунтовані рішення щодо вдосконалення навчально-тренувального процесу.

Досвід цих країн свідчить про те, що ефективна автоматизація обліку спортивних заходів та фізичної підготовки вимагає розробки спеціалізованих програмних продуктів, які враховують специфіку освітнього процесу та потреби користувачів. Такі системи сприяють не лише оптимізації адміністративних завдань, але й підвищенню мотивації здобувачів освіти та покращенню загальних показників фізичної підготовленості.

* 1. Технічне завдання на проектування

Виходячи з аналізу актуальності автоматизації обліку спортивних заходів, потреби у підвищенні ефективності управління фізичною підготовкою у ЗВО, а також враховуючи недоліки існуючих підходів, було прийняте рішення розробити спеціалізований програмний модуль. Це програмне забезпечення призначене для організації та автоматизації процесів планування, обліку, контролю та аналізу даних, що стосуються спортивних заходів та фізичної підготовленості особового складу, відповідно до вимог керівних документів та сучасних потреб освітніх закладів.

Програмний модуль повинен забезпечувати виконання наступних функцій:

* Управління інформацією про підрозділи (навчальні групи) ЗВО (ВВНЗ).
* Ведення реєстру особового складу (курсантів, студентів) із зазначенням необхідних атрибутів (ПІБ, звання, дата народження, підрозділ).
* Формування та актуалізація довідника фізичних вправ та нормативів.
* Облік місць проведення занять (спортивних залів, майданчиків, стадіонів).
* Створення, редагування та перегляд розкладу тренувальних занять та інших спортивних заходів.
* Призначення відповідальних осіб (інструкторів, викладачів) та учасників (підрозділів, окремих осіб) на заняття.
* Візуалізація розкладу у форматі календаря.
* Фіксація результатів виконання нормативів та участі у спортивних заходах для кожного учасника.
* Можливість внесення оцінок, балів та приміток до результатів.
* Генерація зведеної інформації та статистичних даних щодо стану фізичної підготовленості особового складу та ефективності спортивних заходів.
* Візуалізація аналітичних даних за допомогою діаграм та графіків.
* Автентифікація користувачів системи.
* Розмежування прав доступу до функціоналу та даних на основі ролей користувачів (Адміністратор, Працівник навчального відділу, Командир підрозділу, Інструктор).
* Забезпечення надійності та цілісності даних, що зберігаються в системі, на рівні програмної логіки та комунікаційних з’єднань.

Кінцевими користувачами системи повинні бути адміністратор системи, працівники навчального відділу або кафедри фізичного виховання, командири підрозділів та викладачі з фізичної підготовки.

Система повинна структурно складатись з наступних частин:

* серверна частина, що призначена для централізованої обробки та зберігання даних, що необхідні для роботи програмного модуля;
* клієнтська частина для зручного використання системи користувачами;
* база даних для зберігання всієї інформації системи.

Висновок до Розділу 1

У даному розділі було проведено всебічне обґрунтування необхідності розробки програмного модуля для автоматизації обліку спортивних заходів та фізичної підготовки у закладах вищої освіти, з особливим акцентом на специфіку військових вищих навчальних закладів.

Розглянуто нормативно-правове підґрунтя, що регулює організацію фізичної підготовки, зокрема Наказ Міністерства оборони України № 120 від 15.02.2024 та Інструкцію з фізичної підготовки в системі Міністерства оборони України. Проаналізовано ключові вимоги цих документів щодо планування, проведення занять, контролю фізичного стану, ведення облікової документації та звітності. Виявлено, що існуюча практика обліку, яка переважно спирається на паперовий документообіг та ручну обробку даних, є трудомісткою, схильною до помилок та не відповідає сучасним вимогам оперативності та аналітичних можливостей.

Проаналізовано наявні в Україні підходи до автоматизації обліку спортивних заходів, які здебільшого обмежуються використанням неспеціалізованих офісних програмних продуктів або локальними розробками, що не забезпечують комплексності та інтеграції даних.

Дослідження світового досвіду автоматизації подібних процесів у закордонних цивільних та військових навчальних закладах підтвердило доцільність створення спеціалізованих інформаційних систем, що забезпечують ефективне управління фізичною підготовкою, моніторинг результатів та зручний доступ до інформації для всіх учасників освітнього процесу.

На основі проведеного аналізу було сформульовано детальне технічне завдання на проектування програмного модуля. Воно визначає основні функціональні вимоги до системи, включаючи управління довідниками, облік особового складу, планування та облік тренувальних занять, фіксацію результатів виконання нормативів, формування аналітичної звітності, а також вимоги до архітектури, безпеки та користувачів системи.

Таким чином, у першому розділі обґрунтовано актуальність та практичну значущість розробки програмного модуля автоматизації обліку спортивних заходів, що заклало підґрунтя для подальшого проектування та реалізації системи.

РОЗДІЛ 2. ТЕОРЕТИЧНІ ЗАСАДИ ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ СПОРТИВНИХ ЗАХОДІВ

Після всебічного обґрунтування актуальності та нагальної потреби у розробці програмного модуля для автоматизації обліку спортивних заходів та фізичної підготовки у закладах вищої освіти, а також детального формування технічного завдання на його проектування, що було викладено у попередньому розділі, даний розділ присвячений закладенню теоретичних основ та безпосередньому процесу проектування зазначеної системи. Успішна реалізація поставлених функціональних вимог та забезпечення надійної роботи програмного модуля неможливі без ретельного підходу до вибору архітектурних рішень, створення логічно обґрунтованої та ефективної структури зберігання даних, а також запровадження дієвих механізмів забезпечення інформаційної безпеки.

У цьому розділі буде послідовно розглянуто ключові етапи проектування програмного модуля. По-перше, буде здійснено обґрунтування вибору оптимальної архітектури системи, що визначить основні компоненти програмного модуля та характер взаємозв'язків між ними, а також розроблено її узагальнену структурну схему. По-друге, буде детально представлено процес проектування бази даних, що включає розробку концептуальної та логічної моделей даних, необхідних для ефективного обліку, зберігання та обробки інформації про користувачів, особовий склад, підрозділи, тренувальні заняття, фізичні вправи, місця проведення занять та результати оцінювання. Нарешті, особливу увагу буде приділено розгляду теоретичних принципів та практичних методів забезпечення безпеки інформації в розроблюваному програмному модулі, що є критично важливим аспектом для систем, призначених для використання в освітніх, зокрема військових, закладах.

2. 1. Обґрунтування вибору архітектури онлайн-інструменту та розробка її узагальненої структурної схеми.

Вибір правильної архітектури системи є наріжним каменем у процесі створення будь-якої програмної системи, оскільки архітектурне рішення безпосередньо впливає на її загальну структуру, механізми взаємодії окремих компонентів, потенціал до масштабування, рівень надійності та можливості для подальшого вдосконалення та розвитку. Для розроблюваного програмного модуля, призначеного для автоматизації обліку спортивних заходів у ЗВО, враховуючи специфіку поставлених завдань, окреслених у технічному завданні, було обрано клієнт-серверну архітектуру. При цьому, реалізація серверної частини системи передбачається у вигляді монолітного застосунку.

Концептуально, клієнт-серверна архітектура являє собою модель інформаційної мережі, в якій ключові обчислювальні потужності, масиви даних та основна бізнес-логіка концентруються на серверних вузлах. Ці сервери відповідають за обробку запитів, що надходять від клієнтських програм. Така модель традиційно складається з трьох фундаментальних типів елементів (рис. 2.1):

* сервери: це програмно-апаратні комплекси, що забезпечують доступ до баз даних, реалізують виконання закладеної бізнес-логіки та обробляють запити від інших програмних компонентів.
* клієнти: це програми або пристрої, що виступають ініціаторами запитів до серверної частини та використовують надані нею сервіси для реалізації завдань кінцевого користувача.
* мережа: являє собою комунікаційне середовище, яке забезпечує можливість взаємодії між клієнтськими та серверними компонентами, використовуючи для цього стандартизовані протоколи передачі даних, такі як HTTP/HTTPS.

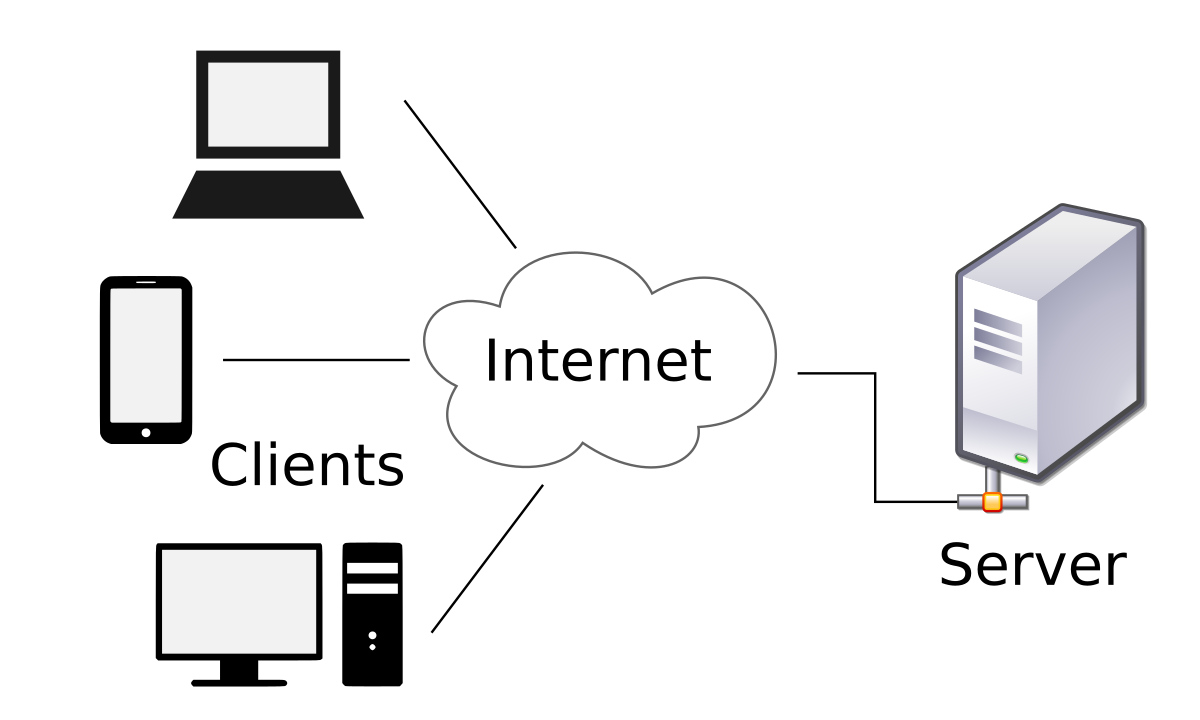


Рисунок 1.8 Типова схема клієнт-серверної архітектури

З точки зору логічної структури, у клієнт-серверній системі прийнято виділяти три взаємопов'язані рівні обробки операцій (рис. 2.2):

* рівень представлення даних (Presentation Tier): Даний рівень несе відповідальність за безпосередню взаємодію з кінцевим користувачем, забезпечуючи відображення інформації та отримання команд через користувацький інтерфейс.
* прикладний рівень (Application Tier / Business Logic Tier): На цьому рівні реалізується основна функціональна логіка програмного модуля, здійснюється обробка даних відповідно до встановлених бізнес-правил та алгоритмів.
* рівень управління даними (Data Tier): Цей рівень відповідає за надійне зберігання, забезпечення доступу, підтримку цілісності та ефективне управління даними. Як правило, він реалізується за допомогою відповідної системи управління базами даних.

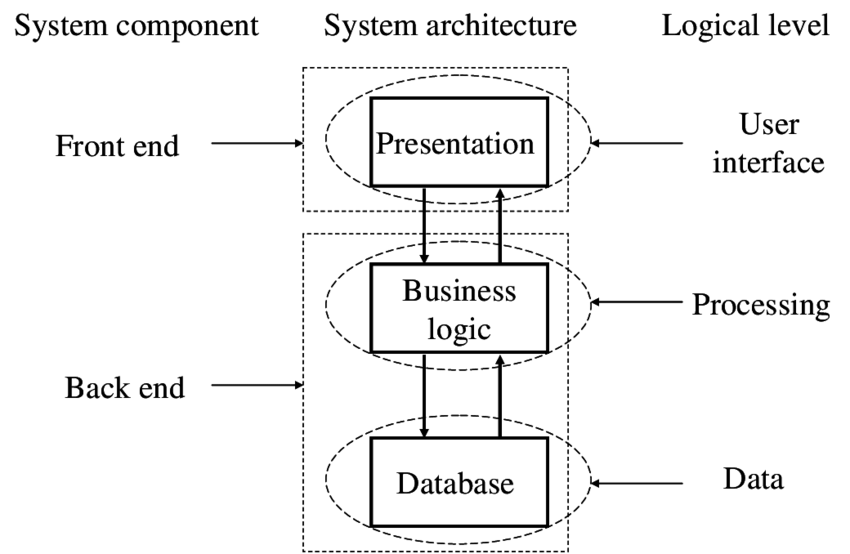


Рисунок 2.2 Схема рівнів обробки операцій клієнт-серверної архітектури

Обґрунтування вибору клієнт-серверної архітектури з монолітною реалізацією серверної компоненти для програмного модуля автоматизації обліку спортивних заходів спирається на низку вагомих переваг, які найкращим чином відповідають поставленим у технічному завданні вимогам. Централізоване зберігання даних та бізнес-логіки на сервері забезпечує їх цілісність, актуальність та можливість контрольованого доступу для різних категорій користувачів, таких як адміністратори, інструктори та керівники підрозділів.

Використання стандартного веб-браузера як клієнта робить систему легкодоступною з будь-якого пристрою, підключеного до мережі, без необхідності інсталяції спеціального програмного забезпечення, що значно полегшує розгортання та подальше обслуговування. Також, такий підхід дозволяє чітко розмежувати відповідальність: клієнт відповідає за інтерфейс, а сервер – за обробку даних та логіку, що спрощує процес розробки та тестування.

Рішення про використання монолітної архітектури для серверної частини на поточному етапі є доцільним з огляду на чітко визначений функціонал системи: управління довідниками, облік занять та результатів, генерація статистичних звітів, розмежування доступу. Моноліт дозволяє швидше розробити та розгорнути початкову версію продукту, оскільки всі серверні компоненти знаходяться в одному програмному просторі, що спрощує їх взаємодію та розгортання. Для передбачуваного обсягу даних та кількості користувачів у рамках одного ЗВО, монолітна структура є достатньо продуктивною та керованою.

Забезпечення належного рівня надійності та безпеки даних, що є критичною вимогою для системи, яка оперує персональними даними та результатами оцінювання, також ефективніше реалізується та контролюється в межах єдиного серверного застосунку. Хоча монолітна архітектура може мати певні обмеження щодо гнучкості та масштабованості для надзвичайно великих систем, для поточного проекту вона пропонує оптимальний баланс між швидкістю розробки, простотою підтримки та відповідністю поставленим завданням. У майбутньому, за умови значного зростання вимог до системи, можливий поступовий перехід до більш розподілених архітектур, наприклад, мікросервісів, для окремих її частин.

Базуючись на обраній клієнт-серверній архітектурі з монолітною серверною частиною, була розроблена узагальнена структурна схема програмного модуля (рис. 2.3). Ця схема візуально представляє ключові компоненти системи, включаючи клієнтську частину, серверну частину, базу даних, а також основні потоки даних та взаємодії між ними.

Зображення, що містить текст, знімок екрана, схема, дизайн

Вміст, створений ШІ, може бути неправильним.

Рисунок 2.3 Структурна схема програмного модулю автоматизації обліку спортивних заходів ЗВО

* 1. Проектування бази даних програмного модулю автоматизації обліку спортивних заходів ЗВО.

Проектування бази даних є одним із критично важливих етапів створення будь-якої інформаційної системи, оскільки від її структури, логічної організації та фізичної реалізації залежать такі ключові характеристики, як цілісність збережених даних, швидкість доступу до них, загальна продуктивність системи та її гнучкість щодо можливих майбутніх модифікацій. База даних слугує централізованим сховищем для всієї інформації, з якою оперує програмний модуль автоматизації обліку спортивних заходів ЗВО.

Процес проектування баз даних традиційно включає декілька послідовних рівнів моделювання:

1. Концептуальне проектування спрямоване на ідентифікацію та опис основних сутностей предметної області, їхніх атрибутів та взаємозв'язків між ними, незалежно від специфіки конкретної системи управління базами даних. На цьому етапі часто використовуються ER-діаграми для візуалізації моделі.
2. Логічне проектування полягає у трансформації концептуальної моделі у конкретну структуру даних, що відповідає обраній моделі даних, наприклад, реляційній, об'єктно-орієнтованій, NoSQL. На цьому етапі визначаються таблиці, їхні поля, типи даних для кожного поля, первинні та зовнішні ключі, а також обмеження цілісності, що забезпечують коректність даних. Важливим процесом на цьому етапі є нормалізація даних, метою якої є усунення надлишковості даних та запобігання потенційним аномаліям при їх оновленні, додаванні чи видаленні.
3. Фізичне проектування визначає, яким чином логічна структура даних буде реалізована на конкретній обраній СУБД з урахуванням її технічних особливостей. Це включає визначення індексів для прискорення пошуку, способів фізичного зберігання даних, оптимізацію запитів та інші аспекти, пов'язані з продуктивністю та ефективністю роботи бази даних.

Для розроблюваного програмного модуля автоматизації обліку спортивних заходів було обрано реляційну модель даних. Реляційні бази даних представляють дані у вигляді набору взаємопов'язаних таблиць, кожна з яких складається з рядків та стовпців. Така модель забезпечує високий рівень структурованості даних, гнучкість у формуванні запитів, надійні механізми забезпечення цілісності даних завдяки використанню первинних та зовнішніх ключів, а також має потужну теоретичну базу та широке розповсюдження. Для взаємодії з реляційними базами даних використовується стандартизована мова запитів SQL.

На основі аналізу вимог технічного завдання та функціоналу системи, визначеного на етапі проектування архітектури, була розроблена логічна схема бази даних (рис. 2.4). Вона включає набір взаємопов'язаних таблиць, що відображають ключові сутності предметної області "Облік спортивних заходів ЗВО".

Основними таблицями в розробленій базі даних є:

* Users (Користувачі системи): Зберігає інформацію про зареєстрованих користувачів програмного модуля. Основні атрибути включають унікальний ідентифікатор, ім'я, прізвище, електронну пошту, хеш пароля (password\_hash), роль користувача в системі: ADMIN, INSTRUCTOR, COMMANDER, DEPARTMENT\_EMPLOYEE, ідентифікатор підрозділу, до якого може належати користувач.
* Units (Підрозділи): Описує навчальні підрозділи або групи ЗВО. Містить унікальний ідентифікатор та назву підрозділу.
* MilitaryPersonnel (Особовий склад): Зберігає дані про курсантів/студентів, які беруть участь у спортивних заходах та фізичній підготовці. Включає ідентифікатор, ім'я, прізвище, звання, дату народження та ідентифікатор підрозділу, до якого приписаний військовослужбовець.
* Exercises (Вправи): Довідник фізичних вправ та нормативів. Атрибути: ідентифікатор, назва вправи та її опис.
* Locations (Місця проведення): Довідник місць, де проводяться заняття або спортивні заходи. Включає ідентифікатор, назву та опис.
* TrainingSessions (Спортивні заходи): Центральна таблиця для обліку запланованих та проведених занять. Містить ідентифікатор, тип заняття: TRAINING, STANDARDS\_ASSESSMENT, UNIT\_TRAINING, дату та час початку і закінчення, ідентифікатор місця проведення, ідентифікатор користувача, який проводить заняття, та ідентифікатор підрозділу, для якого проводиться заняття.
* SessionExercises (Вправи на занятті): Проміжна таблиця для реалізації зв'язку "багато-до-багатьох" між тренувальними заняттями та вправами, що виконуються на цих заняттях. Включає ідентифікатор зв'язку, ідентифікатор заняття, ідентифікатор вправи та порядковий номер вправи у занятті.
* StandardAssessments (Результати оцінювання): Таблиця для фіксації результатів виконання нормативів особовим складом. Містить ідентифікатор оцінки, посилання на заняття, на військовослужбовця, на вправу, отриману оцінку, наприклад, PASSED, EXCELLENT, GOOD, SATISFACTORY, FAILED, дату та час оцінювання та примітки.

Зв'язки між таблицями реалізовані за допомогою первинних ключів та відповідних зовнішніх ключів в залежних таблицях, наприклад, unit\_id в таблиці Users та MilitaryPersonnel вказує на запис у таблиці Units; session\_id, military\_person\_id, exercise\_id в таблиці StandardAssessments вказують на відповідні записи в таблицях TrainingSessions, MilitaryPersonnel та Exercises. Ці зв'язки забезпечують посилальну цілісність даних та дозволяють виконувати складні запити для отримання агрегованої та пов'язаної інформації, що є необхідним для реалізації функціоналу програмного модуля.

Зображення, що містить текст, число, знімок екрана

Вміст, створений ШІ, може бути неправильним.

Рисунок 2.4 Логічна схема бази даних програмного модулю автоматизації обліку спортивних заходів ЗВО

* 1. Принципи та методи забезпечення безпеки інформації в програмному модулі.

Ключовим аспектом при розробці програмного модуля для автоматизації обліку спортивних заходів у ЗВО є забезпечення належного рівня інформаційної безпеки. Система повинна гарантувати захист даних від несанкціонованого доступу, модифікації та знищення, а також забезпечувати доступність інформації лише для авторизованих користувачів відповідно до їхніх повноважень. Реалізація цих вимог досягається шляхом впровадження комплексу програмно-технічних заходів.

Для підтвердження особи користувача та надання йому доступу до системи використовується механізм автентифікації на основі JWT. Коли користувач здійснює вхід, надаючи свої облікові дані (логін та пароль), серверна частина перевіряє їхню відповідність. Важливо, що паролі в базі даних зберігаються у вигляді криптографічних хеш-сум, створених за допомогою бібліотеки bcryptjs, що унеможливлює їх відновлення у відкритому вигляді.

У разі успішної автентифікації, сервер генерує JWT, що містить інформацію про користувача, його ідентифікатор та роль, і підписується секретним ключем process.env.JWT\_SECRET. Цей токен передається на клієнтську сторону, де зберігається, наприклад, в localStorage і використовується для автентифікації наступних запитів до API, додаючись до заголовка Authorization. Перевірка валідності токена на сервері здійснюється за допомогою проміжного програмного забезпечення authMiddleware.js.

Для забезпечення гранульованого контролю над діями користувачів в системі впроваджено механізм розмежування доступу на основі ролей. Кожному користувачеві присвоюється певна роль, яка визначає набір його повноважень. У системі визначені наступні ролі: ADMIN, DEPARTMENT\_EMPLOYEE, COMMANDER, INSTRUCTOR. Права доступу для кожної ролі до конкретних API-маршрутів та HTTP-методів конфігуруються на сервері та перевіряються за допомогою проміжного програмного забезпечення roleMiddleware.js. На клієнтській частині також здійснюється контроль доступу до елементів інтерфейсу та маршрутів через компонент ProtectedRoute.jsx та логіку в authStore.js.

Конфіденційність даних під час передачі між клієнтом і сервером забезпечується шляхом використання протоколу HTTPS. Цілісність даних на рівні бази даних підтримується завдяки використанню реляційної СУБД PostgreSQL та механізмів ORM Sequelize, що включають валідацію даних та параметризовані запити для запобігання SQL-ін'єкціям.

Обрана комбінація механізмів автентифікації на основі JWT та авторизації за ролями надає системі ряд суттєвих переваг. По-перше, це гнучкість управління правами доступу: адміністратор може легко налаштовувати повноваження для різних категорій користувачів, не вносячи змін до коду програми. По-друге, забезпечується дотримання принципу мінімальних привілеїв, коли кожен користувач отримує лише ті права, які необхідні для виконання його функціональних обов'язків, що знижує ризик несанкціонованих дій або випадкового пошкодження даних. По-третє, використання JWT токенів сприяє масштабованості системи, оскільки сервер не потребує зберігання інформації про сесії користувачів. Крім того, хешування паролів та захист каналів передачі даних підвищують загальний рівень захищеності персональної та службової інформації, що обробляється в програмному модулі. Централізований контроль доступу через проміжне програмне забезпечення на сервері є основним рубежем захисту, доповненим відповідними перевірками на клієнтській стороні для покращення користувацького досвіду.

Висновок до Розділу 2

У другому розділі кваліфікаційної роботи було закладено теоретичне підґрунтя та розроблено ключові проектні рішення для створення програмного модуля автоматизації обліку спортивних заходів та фізичної підготовки у закладах вищої освіти.

На основі аналізу функціональних та нефункціональних вимог, викладених у технічному завданні, було обґрунтовано вибір клієнт-серверної архітектури з монолітною реалізацією серверної частини на платформі Node.js з фреймворком Express.js та клієнтською частиною у вигляді односторінкового веб-додатку (SPA) на React. Така архітектурна модель була визнана оптимальною для даного етапу розробки, оскільки вона забезпечує централізоване управління даними, гнучкий доступ для користувачів через веб-браузер, чітке розмежування відповідальності між клієнтською та серверною логікою, а також дозволяє досягти відносної простоти розробки та розгортання для визначеного обсягу функціоналу. Було представлено узагальнену структурну схему системи, що візуалізує основні компоненти та потоки даних між ними.

Важливою складовою проектних робіт стала розробка логічної структури бази даних. На основі обраної реляційної моделі даних та СУБД PostgreSQL було визначено ключові сутності предметної області, необхідні для всебічного обліку спортивних заходів. До них належать: "Користувачі", "Підрозділи", "Особовий склад", "Вправи", "Місця проведення", "Тренувальні заняття", "Вправи на занятті" та "Результати оцінювання". Для кожної сутності було визначено атрибути та встановлено взаємозв'язки, що забезпечують цілісність та ефективне управління інформацією.

Значну увагу було приділено питанням забезпечення інформаційної безпеки розроблюваного програмного модуля. Було детально описано архітектуру безпеки, що включає механізми автентифікації користувачів на основі JSON Web Tokens та авторизації за допомогою рольової моделі доступу. Визначено основні ролі користувачів та їхні передбачувані повноваження в системі. Також розглянуто питання захисту паролів шляхом хешування та забезпечення безпеки каналів передачі даних. Обґрунтовано переваги обраного підходу до безпеки, зокрема його гнучкість, дотримання принципу мінімальних привілеїв та сприяння масштабованості системи.

Таким чином, у другому розділі було виконано всі необхідні теоретичні та проектні роботи, що створюють міцну основу для подальшої практичної реалізації програмного модуля автоматизації обліку спортивних заходів. Розроблені архітектурні рішення, деталізована структура бази даних та продумані принципи забезпечення безпеки спрямовані на створення надійного, функціонального та захищеного програмного продукту, здатного задовольнити потреби користувачів у автоматизації обліку фізичної підготовки та спортивних заходів у ЗВО.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО МОДУЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ СПОРТИВНИХ ЗАХОДІВ

Після ретельного аналізу предметної області, обґрунтування доцільності та необхідності створення програмного модуля для автоматизації обліку спортивних заходів та фізичної підготовки у закладах вищої освіти, а також розробки його концептуальних засад, включаючи архітектуру, структуру бази даних та принципи забезпечення інформаційної безпеки, що були викладені у попередніх розділах, даний розділ присвячений безпосередній практичній реалізації спроєктованої системи. На цьому етапі теоретичні напрацювання та проектні рішення знаходять своє втілення у конкретному програмному продукті, готовому до тестування, розгортання та подальшого використання.

Метою цього розділу є послідовний опис ключових аспектів процесу розробки програмного модуля. Буде детально розглянуто обґрунтування вибору конкретних інструментальних засобів та технологій, які були використані для створення серверної та клієнтської частин системи, а також системи управління базами даних. Далі буде висвітлено процес розробки основних програмних компонентів, включаючи реалізацію серверної логіки та прикладного програмного інтерфейсу, створення клієнтської частини з користувацькими інтерфейсами для взаємодії з системою, а також розробку та інтеграцію модуля управління доступом користувачів. Також буде розглянуто питання, пов'язані з підготовкою програмного модуля до розгортання у робочому середовищі.

3.1. Обґрунтування вибору інструментальних засобів та технологій для розробки програмного модулю

Вибір технологічного стеку є визначальним етапом, що закладає фундамент для успішної розробки, подальшої підтримки та потенційного масштабування програмного продукту. Для програмного модуля автоматизації обліку спортивних заходів ЗВО цей вибір здійснювався з урахуванням специфіки предметної області, функціональних вимог, визначених у технічному завданні, обраної клієнт-серверної архітектури, а також таких критеріїв, як швидкість розробки, надійність, безпека, доступність ресурсів та активність спільноти розробників. Розглянемо детальніше обґрунтування для кожної з ключових компонент системи.

Для реалізації серверної частини програмного модуля було обрано платформу Node.js та веб-фреймворк Express.js. Node.js – це середовище виконання JavaScript, що працює на високопродуктивному рушії V8 від Google. Його ключовою перевагою є неблокуюча, подієво-орієнтована архітектура вводу-виводу (I/O). Це означає, що Node.js може ефективно обробляти велику кількість одночасних підключень та запитів, не блокуючи основний потік виконання, що є критично важливим для веб-застосунків, де очікується швидка реакція на дії користувачів. Для системи обліку спортивних заходів, яка передбачає взаємодію з багатьма користувачами, така архітектура забезпечує високу пропускну здатність та низьку затримку.

Використання JavaScript як єдиної мови програмування і для серверної, і для клієнтської частини є ще однією суттєвою перевагою. Це дозволяє уніфікувати процес розробки, спростити обмін даними та логікою між клієнтом і сервером, а також потенційно скоротити час на навчання команди розробників, якщо вони вже знайомі з JavaScript.

Express.js, у свою чергу, є мінімалістичним, гнучким та одним з найпопулярніших веб-фреймворків для Node.js. Він надає потужний набір інструментів для створення RESTful API, що є основою взаємодії між клієнтською та серверною частинами даного програмного модуля. Його переваги включають простоту освоєння, велику та активну спільноту, що забезпечує наявність значної кількості готових модулів та рішень для типових завдань. Альтернативами могли б бути інші Node.js фреймворки, такі як NestJS або Koa. Однак, для даного проєкту з монолітною архітектурою та чітко визначеним функціоналом, гнучкість та простота Express.js є оптимальними, дозволяючи швидше реалізувати необхідний API. Порівняно з платформами на інших мовах, такими як Python (Django, Flask) чи Java (Spring), Node.js з Express.js часто демонструє кращу продуктивність саме в I/O-залежних операціях, що типово для веб-серверів, які взаємодіють з базою даних та численними клієнтами.



Рисунок 3.1 Лого Node.js та Express.js

Для роботи з базою даних PostgreSQL на серверній стороні було обрано Object-Relational Mapper Sequelize. Використання ORM, такого як Sequelize, надає низку переваг порівняно з написанням SQL-запитів. По-перше, це абстракція від конкретної СУБД: хоча проєкт використовує PostgreSQL, Sequelize теоретично дозволяє легше мігрувати на іншу реляційну СУБД у майбутньому наприклад, MySQL, SQLite, MSSQL, з мінімальними змінами в коді. По-друге, ORM дозволяє працювати з даними як з об'єктами JavaScript, що робить код більш читабельним, структурованим та легшим для розуміння, особливо для розробників, які більше зосереджені на об'єктно-орієнтованому програмуванні. По-третє, Sequelize бере на себе багато рутинних завдань, таких як генерація SQL-запитів, управління з'єднаннями, міграції схеми бази даних та валідація даних на рівні моделей. Це значно прискорює процес розробки та зменшує ймовірність помилок, зокрема, пов'язаних з SQL-ін'єкціями, оскільки ORM використовує параметризовані запити. Альтернативами могли б бути інші ORM для Node.js, такі як TypeORM або Knex.js. Однак, Sequelize є зрілим, добре документованим рішенням з великою спільнотою, що робить його надійним вибором для даного проєкту.



Рисунок 3.2 Лого Sequelize.js та PostgreSQL

Для розробки користувацького інтерфейсу програмного модуля було обрано бібліотеку React.js. React є однією з найпопулярніших JavaScript-бібліотек для створення динамічних та інтерактивних користувацьких інтерфейсів. Її ключовими перевагами є:

* Компонентний підхід: React дозволяє розбивати складний інтерфейс на невеликі, незалежні та перевикористовувані компоненти. Це значно спрощує процес розробки, тестування та подальшої підтримки коду.
* Віртуальний DOM (Virtual DOM): React використовує віртуальну репрезентацію DOM, що дозволяє оптимізувати процес оновлення реального DOM, оновлюючи лише ті частини, які дійсно змінилися. Це призводить до кращої продуктивності, особливо у складних застосунках з частими оновленнями даних.
* Велика екосистема та спільнота: Навколо React існує величезна екосистема бібліотек, інструментів та активна спільнота розробників, що забезпечує доступ до готових рішень, навчальних матеріалів та швидку підтримку.

Альтернативами React могли б бути інші популярні JavaScript-фреймворки, такі як Angular або Vue.js. Angular є більш комплексним та "жорстким" фреймворком, що пропонує комплексне рішення "з коробки", але може бути надлишковим та мати вищий поріг входження для проєктів середнього розміру. Vue.js, відомий своєю простотою та гнучкістю, є гарною альтернативою, але React наразі має дещо більшу екосистему та поширеність, що може бути перевагою при пошуку розробників або готових компонентів.

Для управління станом клієнтського застосунку було обрано бібліотеку MobX. MobX пропонує простий та інтуїтивно зрозумілий підхід до управління станом, заснований на принципах реактивного програмування. Він дозволяє легко визначати спостережувані дані та реакції на їх зміни, що автоматично оновлюють відповідні компоненти інтерфейсу. Порівняно з більш популярним Redux, MobX часто вимагає менше шаблонного коду та пропонує більш прямий шлях до синхронізації стану та UI, що може прискорити розробку, особливо для проєктів, де не потрібна надлишкова складність управління потоками даних.

Для створення візуально привабливого та консистентного інтерфейсу було використано бібліотеку UI-компонентів Material-UI. MUI надає великий набір готових, добре протестованих та кастомізованих React-компонентів, що реалізують принципи Material Design. Це дозволяє значно прискорити процес розробки інтерфейсу, забезпечити його адаптивність для різних розмірів екранів та надати користувачам знайомий та інтуїтивно зрозумілий досвід взаємодії.

Для зберігання даних програмного модуля було обрано об'єктно-реляційну СУБД PostgreSQL. PostgreSQL є потужною, надійною та функціональною СУБД з відкритим вихідним кодом. Її перевагами є сувора відповідність стандартам SQL, висока розширюваність (можливість додавання користувацьких типів даних, функцій, операторів), підтримка транзакцій ACID, що гарантує цілісність даних, та розвинені механізми для реплікації та відмовостійкості. Для проєкту, що передбачає зберігання структурованої інформації про особовий склад, їхні результати, розклад занять та інші пов'язані дані, реляційна модель PostgreSQL є оптимальним вибором. Альтернативою могла б бути MySQL, яка також є популярною СУБД з відкритим кодом, проте PostgreSQL часто вважається більш потужною у роботі зі складними запитами та має кращу підтримку розширень. NoSQL бази даних розглядалися, але для даного проєкту з чітко визначеною схемою даних та потребою у транзакційності реляційна модель є більш виправданою.

Для забезпечення ефективного процесу розробки клієнтської частини було обрано інструмент збірки Vite. Vite забезпечує надзвичайно швидкий запуск сервера для розробки та миттєве "гаряче" оновлення модулів (HMR), що значно прискорює ітераційний процес розробки інтерфейсів. Порівняно з більш традиційними збирачами, такими як Webpack, Vite пропонує кращу продуктивність на етапі розробки завдяки використанню нативних ES-модулів браузера.

Для розгортання програмного модуля та управління його компонентами в різних середовищах було прийнято рішення використовувати технологію контейнеризації Docker та інструмент оркестрації docker-compose. Docker дозволяє "запакувати" серверний застосунок, базу даних та клієнтську частину разом з усіма їхніми залежностями в ізольовані контейнери. Це гарантує, що програмне забезпечення буде працювати однаково незалежно від оточення, усуває проблеми сумісності та спрощує процес розгортання. Docker Compose дозволяє легко визначати та запускати багатоконтейнерні Docker-застосунки за допомогою одного конфігураційного файлу. Такий підхід забезпечує портативність, відтворюваність та спрощує управління життєвим циклом програмного модуля.

Сукупність обраних технологій та інструментів формує сучасний, продуктивний та надійний стек, що дозволяє ефективно реалізувати поставлені завдання з розробки програмного модуля автоматизації обліку спортивних заходів та забезпечує хорошу основу для його подальшої підтримки та розвитку.

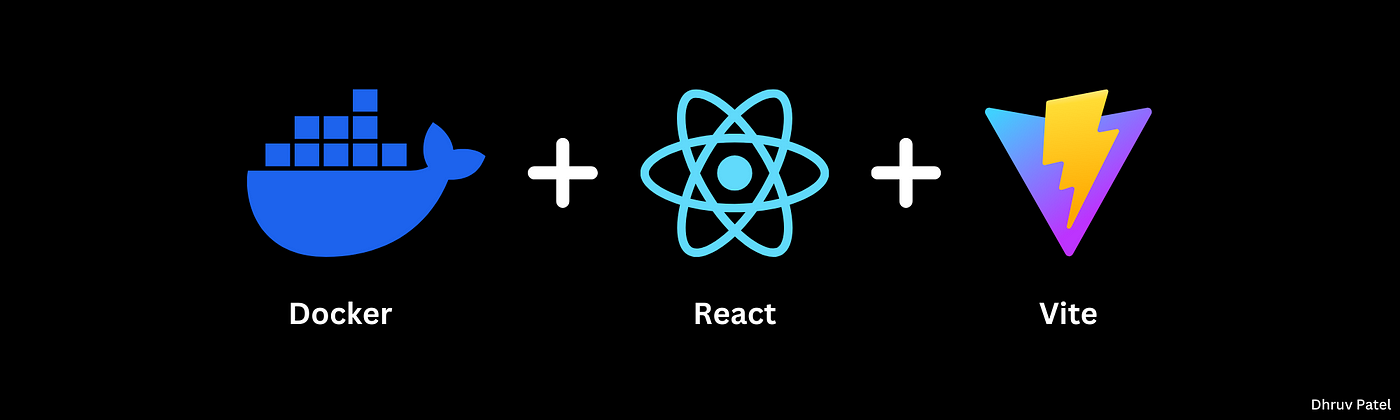


Рисунок 3.3 Лого Docker, React.js та Vite

3.2. Розробка програмного модулю.

3.2.1. Реалізація серверної логіки та прикладного програмного інтерфейсу.

Центральним компонентом розробленого програмного модуля є його серверна частина, що функціонує як ядро системи. Вона відповідає за управління даними, реалізацію основних бізнес-процесів та забезпечення взаємодії з клієнтським застосунком через прикладний програмний інтерфейс. Для побудови серверної частини було обрано платформу Node.js та фреймворк Express.js, що дозволило створити продуктивне та масштабоване рішення, здатне ефективно обробляти запити від користувачів.

Архітектура серверного застосунку спроєктована за модульним принципом, що передбачає логічний поділ коду на функціональні блоки для підвищення його читабельності, керованості та можливості подальшого розширення. Ключовими елементами цієї структури є моделі даних, маршрутизатори, контролери та сервіси. Моделі даних, визначені за допомогою ORM Sequelize, точно відображають структуру таблиць бази даних PostgreSQL та інкапсулюють логіку взаємодії з ними на низькому рівні. Кожна модель, наприклад User.model.js або TrainingSession.model.js, описує атрибути відповідної сутності та її зв'язки з іншими сутностями, забезпечуючи об'єктно-орієнтований підхід до роботи з даними.

Маршрутизація запитів API реалізована за допомогою Express Router. Для кожної основної сутності системи створено окремий файл маршрутів, який визначає конкретні URL-адреси та відповідні їм HTTP-методи. Ці маршрутизатори підключаються до основного файлу застосунку app.js, формуючи єдиний API системи. Обробку конкретних запитів, що надходять на ці ендпоінти, здійснюють контролери, які отримують дані із запиту, валідують їх та викликають відповідні сервісні методи для виконання основної бізнес-логіки. Для забезпечення стабільної роботи та коректної обробки можливих помилок під час виконання асинхронних операцій у контролерах використовується спеціальна функція-обгортка catchErrorsAsync.

Шар сервісів інкапсулює основну бізнес-логіку програмного модуля. Сервіси, такі як userService.js або trainingSessionService.js, відповідають за виконання операцій CRUD з даними, реалізацію специфічних правил обробки інформації, що стосується обліку спортивних заходів, та взаємодію з моделями Sequelize. Наприклад, сервіс для роботи з тренувальними заняттями забезпечує не лише створення самого заняття, але й обробку пов'язаних із ним вправ через проміжну модель SessionExercise, використовуючи для цього транзакції бази даних для забезпечення атомарності операцій.

Прикладний програмний інтерфейс розроблено з дотриманням принципів REST. Кожна сутність системи розглядається як ресурс, доступ до якого здійснюється через стандартизовані HTTP-методи. Наприклад, для отримання списку всіх тренувальних занять клієнтська частина надсилає GET-запит на ендпоінт /api/training-sessions, а для створення нового заняття – POST-запит на той самий ендпоінт з тілом запиту, що містить необхідні дані. Форматом обміну даними є JSON. Система відповідей API також стандартизована: успішні операції супроводжуються відповідними HTTP-статусами, наприклад, 200 OK або 201 Created, а у випадку помилок сервер повертає інформативний статус-код, наприклад, 400 Bad Request, та JSON-об'єкт з описом помилки, що полегшує діагностику та налагодження на клієнтській стороні.

Крім базових операцій з даними, серверна логіка охоплює й більш складні аспекти. Наприклад, при видаленні певних сутностей, таких як підрозділ або користувач, система забезпечує коректне оновлення або видалення пов'язаних записів в інших таблицях для підтримки цілісності даних та уникнення "осиротілих" записів. Так, при видаленні підрозділу, відповідне поле unit\_id у користувачів та військовослужбовців цього підрозділу може бути очищене або, залежно від бізнес-логіки, пов'язані записи можуть бути також видалені, що реалізовано у відповідних сервісах. Аналогічно, видалення тренувального заняття тягне за собою видалення всіх пов'язаних з ним записів про вправи та результати оцінювання. API також надає можливості для фільтрації даних, що дозволяє клієнтській частині запитувати лише необхідну інформацію, наприклад, отримати список занять за певний період або для конкретного підрозділу.

Реалізована серверна частина, таким чином, формує надійну основу для функціонування всього програмного модуля, забезпечуючи логіку обробки даних, безпеку та ефективну взаємодію з клієнтським застосунком, що є передумовою для успішного вирішення поставлених завдань автоматизації обліку спортивних заходів.

3.2.2. Реалізація клієнтської частини та користувацьких інтерфейсів.

Клієнтська частина програмного модуля є інтерактивним веб-додатком, що слугує основним засобом взаємодії користувачів із системою. Вона розроблена як односторінковий застосунок, що забезпечує швидке завантаження та плавну навігацію без перезавантаження сторінки. Основою для розробки клієнтської частини стала бібліотека React.js, яка дозволила ефективно структурувати інтерфейс у вигляді ієрархії незалежних та перевикористовуваних компонентів. Це значно спростило процес розробки, тестування та подальшої підтримки коду.

Візуальне оформлення та інтерактивні елементи користувацького інтерфейсу реалізовані з використанням бібліотеки компонентів Material-UI. MUI надає широкий набір готових до використання та кастомізованих компонентів, таких як кнопки, текстові поля, таблиці, діалогові вікна, навігаційні панелі та інше, що відповідають сучасним принципам дизайну Material Design. Це дозволило створити естетично привабливий, інтуїтивно зрозумілий та узгоджений інтерфейс для всіх розділів програмного модуля. Для забезпечення консистентності візуального стилю та можливості перемикання між світлою та темною темами було розроблено кастомізовану тему у файлі theme.js.

Управління станом даних на клієнтській стороні здійснюється за допомогою бібліотеки MobX. Для кожної ключової сутності системи створено відповідні сховища, наприклад, userStore.js, trainingSessionStore.js, standardAssessmentStore.js тощо. Ці сховища відповідають за завантаження даних з серверного API, їх локальне зберігання у вигляді спостережуваних змінних, а також надають методи для додавання, оновлення та видалення даних. React-компоненти, що відображають ці дані, автоматично оновлюються при будь-яких змінах у відповідних сховищах, забезпечуючи реактивність та актуальність інтерфейсу. Взаємодія з серверним API здійснюється через централізований HTTP-клієнт Axios, налаштований у файлі apiClient.js, який також відповідає за автоматичне додавання JWT-токенів до заголовків запитів для автентифікації.

Навігація між різними сторінками та розділами програмного модуля реалізована за допомогою бібліотеки React Router DOM, що дозволяє створювати чітку та логічну структуру веб-додатку. Доступ до окремих сторінок та функціональних можливостей контролюється компонентом ProtectedRoute.jsx, який перевіряє стан автентифікації користувача та його роль перед наданням доступу до відповідного контенту.

Ключовим елементом навігації є верхня панель (рис. ), яка містить посилання на основні розділи системи, такі як "Головна", "Заняття", "Оцінки", "Військовослужбовці", "Вправи", "Локації", "Підрозділи", "Користувачі" та "Календар". Набір доступних пунктів меню адаптується залежно від ролі поточного користувача, забезпечуючи індивідуалізований доступ до функціоналу.

  
Рисунок 3.4 Глобальний компонент-хеадер сайту

Сторінка входу до системи є першою точкою взаємодії для неавтентифікованих користувачів. Вона містить форму для введення електронної пошти (логіну) та пароля. Після успішної автентифікації на сервері користувач перенаправляється на відповідну головну сторінку.

Зображення, що містить текст, знімок екрана, дизайн

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.5 Сторінка входу до системи

Головна інформаційна панель (Dashboard) слугує центральним інформаційним вузлом для авторизованих користувачів зокрема, для ролей COMMANDER, DEPARTMENT\_EMPLOYEE, ADMIN. Ця сторінка агрегує та візуалізує ключові показники ефективності та статистику щодо фізичної підготовки та спортивних заходів. Тут відображаються картки зі зведеними даними, такі як загальна кількість особового складу, кількість тренувальних занять за поточний тиждень та кількість підрозділів, охоплених системою. Крім того, на дашборді представлені різноманітні діаграми, реалізовані за допомогою бібліотеки Recharts.js, для наочного відображення розподілу занять за типами, за місцями проведення, розподілу особового складу за підрозділами, статистики отриманих оцінок та динаміки успішності з часом.

Зображення, що містить текст, знімок екрана, схема, Барвистість

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.6 Головна інформаційна панель з прикладами статистичних блоків та діаграм

Для управління основними довідниками та сутностями системи, такими як користувачі, підрозділи, особовий склад, фізичні вправи, місця проведення занять, тренувальні заняття та результати оцінювання, розроблені відповідні сторінки зі списками, наприклад сторінка керування користувачами (рис. 3.7). Ці сторінки використовують кастомізований компонент CustomDataGrid, побудований на базі @mui/x-data-grid. Цей компонент забезпечує відображення даних у табличному вигляді, підтримує сортування за стовпцями, фільтрацію, пагінацію, а також надає інструменти для виконання CRUD-операцій над записами, доступність яких контролюється залежно від ролі користувача.

Зображення, що містить текст, знімок екрана, число, Шрифт

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.7 - Приклад сторінки управління користувачами

Для додавання нових записів або редагування існуючих використовуються відповідні сторінки з формами вводу даних, наприклад, TrainingSessionFormPage.jsx для занять (рис. 3.8). Ці форми розроблені з використанням широкого спектра компонентів Material-UI, таких як TextField для введення тексту, Select та MenuItem для випадаючих списків, DatePicker та DateTimePicker з пакету @mui/x-date-pickers для зручного вибору дати та часу. У формах реалізована клієнтська валідація введених даних для запобігання помилкам перед відправкою інформації на сервер.

Зображення, що містить текст, знімок екрана, програмне забезпечення, число

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.8 Приклад форми редагування заняття типу “Здача нормативів” з відповідними полями

Окремим важливим функціональним блоком є сторінка Календаря (рис. 3.9), яка інтегрує компонент @fullcalendar/react. Вона надає користувачам наочне візуальне представлення розкладу всіх запланованих та проведених занять та спортивних заходів. Календар підтримує різні режими відображення та дозволяє переглядати детальну інформацію про кожну подію у спливаючому модальному вікні. Для користувачів з відповідними правами передбачена можливість швидкого створення нового заняття шляхом вибору часового слоту безпосередньо на календарі.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Комп’ютерна піктограма

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.9 - Сторінка календаря з відображенням подій різного типу, що мають різне кольорове кодування

Для оперативного внесення результатів виконання нормативів після проведення відповідних занять розроблено спеціалізований інтерфейс на сторінці оцінювання за результатами заняття (рис. 3.10). Цей інтерфейс дозволяє інструктору або іншій відповідальній особі зручно переглядати список особового складу, що брав участь у занятті, та для кожної запланованої вправи вносити отримані оцінки, наприклад, "відмінно", "добре", "зараховано", та короткі примітки.

Зображення, що містить текст, знімок екрана, число, Шрифт

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.10 - Сторінка внесення оцінок, що відображає таблицю з ПІБ курсантів, назвами вправ та полями для введення оцінок за вправу та приміток

Для забезпечення можливості користувачам керувати власними обліковими даними передбачена особистого кабінету (рис. 3.11). Тут автентифікований користувач може переглянути свою поточну інформацію та за потреби оновити деякі дані, наприклад, змінити пароль.

Зображення, що містить текст, знімок екрана, число, ряд

Вміст, створений ШІ, може бути неправильним.

Рисунок 3.11 - Сторінка Особистого кабінету

Вся сукупність розроблених користувацьких інтерфейсів та компонентів клієнтської частини спрямована на створення інтуїтивно зрозумілого, функціонального та ефективного інструменту для автоматизації обліку спортивних заходів, що відповідає потребам різних категорій користувачів у ЗВО.

3.2.3. Розробка та інтеграція модуля управління доступом та розмежування прав користувачів.

Забезпечення контрольованого доступу до функціоналу та даних програмного модуля є однією з фундаментальних вимог, особливо враховуючи потенційне використання системи у військових навчальних закладах, де обробляється службова та персональна інформація. Реалізація надійного модуля управління доступом ґрунтується на двох ключових процесах: автентифікації для підтвердження особи користувача та авторизації для надання йому відповідних прав доступу згідно з призначеною роллю. Ці механізми були спроєктовані на етапі теоретичного обґрунтування та знайшли своє втілення у програмному коді як серверної, так і клієнтської частин розроблюваної системи.

Процес автентифікації користувачів реалізований з використанням технології JSON Web Tokens. На серверній стороні, міститься логіка обробки запиту на вхід до системи. При отриманні від клієнта коректних облікових даних, які попередньо перевіряються на відповідність хешованим паролям, що зберігаються у базі даних та хешується за допомогою бібліотеки bcrypt.js, генерується JWT токен. Цей токен містить зашифровану інформацію про користувача, зокрема його унікальний ідентифікатор, логін, повне ім'я та призначену роль. Токен підписується секретним ключем, відомим лише серверу, що гарантує його цілісність та автентичність.

Зображення, що містить текст, знімок екрана, число, Паралель

Вміст, створений ШІ, може бути неправильним.  
Рисунок 3.12 - Діаграма процесу автентифікації користувача з використанням JWT

Згенерований JWT токен передається на клієнтську частину, де він зберігається, наприклад, у localStorage браузера. Спеціалізований HTTP-клієнт apiClient.js налаштований таким чином, щоб автоматично додавати цей токен до заголовка Authorization кожного наступного запиту до захищених ресурсів API. Серверна частина, зі свого боку, перед обробкою будь-якого запиту до захищених маршрутів використовує проміжне програмне забезпечення authMiddleware.js. Це ПЗ перевіряє наявність та валідність JWT токена. Якщо токен валідний, інформація про користувача, декодована з токена, додається до об'єкта запиту, роблячи її доступною для подальшої обробки, зокрема для механізму розмежування прав. У випадку невалідного або відсутнього токена, доступ до ресурсу блокується.

Після успішного проходження процедури автентифікації в дію вступає механізм авторизації, який базується на ролях користувачів. У системі визначено чотири основні ролі, та реалізовано у переліченні enum\_Users\_role в базі даних та відповідних константах на клієнті: ADMIN - адміністратор, DEPARTMENT\_EMPLOYEE - працівник навчального відділу, COMMANDER - командир підрозділу та INSTRUCTOR - інструктор.

Для реалізації розмежування прав на серверній стороні було розроблено спеціальне проміжне програмне забезпечення roleMiddleware.js. Це ПЗ приймає на вхід конфігурацію дозволів для конкретного API-маршруту або групи маршрутів, де чітко вказано, які ролі та які HTTP-методи є дозволеними для даного ресурсу. Перед виконанням основного обробника запиту, roleMiddleware перевіряє роль поточного автентифікованого користувача та його право на доступ до запитуваного ресурсу з використанням вказаного HTTP-методу. Якщо доступ заборонено, користувач отримує відповідь зі статусом 403 Forbidden. Конфігурація конкретних прав доступу для різних маршрутів API визначена у головному файлі серверного застосунку. Наприклад, адміністратор має повний доступ до управління користувачами та підрозділами, тоді як інструктор може мати права на створення та редагування тренувальних занять та внесення оцінок, але обмежений у доступі до адміністрування користувачів.

На клієнтській стороні розмежування прав проявляється через динамічне відображення елементів інтерфейсу та контроль доступності маршрутів. Компонент ProtectedRoute.jsx, що використовується у файлі App.jsx для обгортання захищених маршрутів, перевіряє, чи автентифікований користувач і чи належить його роль до списку дозволених ролей для конкретного маршруту. Якщо доступ заборонено, користувач перенаправляється на сторінку входу або на спеціальну сторінку "Доступ заборонено" (рис. 3.13). Крім того, доступність окремих елементів інтерфейсу, таких як кнопки "Додати", "Редагувати", "Видалити" у таблицях даних, також контролюється залежно від ролі користувача, інформація про яку зберігається у authStore.js.

Інтеграція модуля управління доступом відбувається на кількох рівнях: на рівні API, де кожен захищений маршрут на сервері обробляється відповідними authMiddleware та roleMiddleware; на рівні клієнтського застосунку, де маршрутизація та відображення елементів UI залежать від ролі користувача; та на рівні бази даних, де ролі користувачів зберігаються в таблиці Users та використовуються для визначення їхніх повноважень. Такий комплексний дворівневий підхід до автентифікації та авторизації дозволяє створити надійний та гнучкий механізм управління доступом, що відповідає вимогам безпеки та функціональним потребам програмного модуля автоматизації обліку спортивних заходів.

3.3. Розгортання онлайн-інструменту.

Завершальним етапом практичної реалізації програмного модуля автоматизації обліку спортивних заходів є його розгортання, що забезпечує доступність системи для кінцевих користувачів та її стабільну роботу в цільовому середовищі. Обраний підхід до розгортання ґрунтується на сучасних практиках контейнеризації з використанням Docker та інструменту оркестрації Docker Compose. Таке рішення надає низку переваг, включаючи портативність, ізоляцію компонентів системи, консистентність середовищ та спрощення процесів адміністрування та оновлення.

Контейнеризація за допомогою Docker дозволяє "упакувати" кожну складову програмного модуля - серверну частину, базу даних та, за потреби, клієнтську частину - в окремі ізольовані контейнери. Кожен контейнер містить усі необхідні для роботи компонента залежності, бібліотеки та конфігураційні файли. Це гарантує, що програмне забезпечення буде функціонувати однаково незалежно від того, на якій машині воно запускається, усуваючи поширені проблеми сумісності між середовищами розробки, тестування та експлуатації.

Оркестрація цих контейнерів здійснюється за допомогою Docker Compose, конфігурація якого описана у файлі docker-compose.yaml. Цей файл визначає сервіси, що становлять систему, їхні образи, налаштування мережі, портів та томів для зберігання даних.

На основі наданого файлу docker-compose.yaml, ключовими сервісами системи є:

* Сервіс бази даних postgres: Цей сервіс відповідає за розгортання системи управління базами даних PostgreSQL, використовуючи офіційний образ postgres:15. Усі дані програмного модуля, що стосуються користувачів, підрозділів, особового складу, занять, вправ, локацій та результатів оцінювання, зберігаються саме тут. Для забезпечення персистентності даних, тобто їх збереження навіть після зупинки чи перезапуску контейнера, використовується іменований том postgres\_data. Конфігураційні параметри, такі як ім'я користувача бази даних (POSTGRES\_USER: admin), пароль (POSTGRES\_PASSWORD: admin123) та назва самої бази даних (POSTGRES\_DB: physical\_sessions\_db), передаються сервісу через змінні середовища. Стандартний порт СУБД PostgreSQL (5432) відображається на аналогічний порт хост-машини для забезпечення підключення до бази даних з боку серверної частини застосунку та, за потреби, для адміністрування.
* Сервіс адміністрування бази даних pgadmin: Для зручного управління базою даних PostgreSQL розгорнуто сервіс pgAdmin, що базується на офіційному образі dpage/pgadmin4. Цей інструмент надає графічний веб-інтерфейс для виконання SQL-запитів, перегляду та модифікації структури даних, управління користувачами бази даних тощо. Сервіс pgadmin залежить від успішного запуску сервісу postgres. Облікові дані для доступу до веб-інтерфейсу pgAdmin також встановлюються через змінні середовища (PGADMIN\_DEFAULT\_EMAIL та PGADMIN\_DEFAULT\_PASSWORD). Порт контейнера pgAdmin відображається на порт 80 (або інший налаштований) хост-машини, забезпечуючи доступ до інструменту через браузер.

Варто зазначити, що у наданому файлі docker-compose.yaml секції для розгортання клієнтської та серверної частин застосунку закоментовані. Для повноцінного розгортання всього програмного модуля ці секції необхідно активувати та забезпечити наявність відповідних Dockerfile у кореневих директоріях frontend та backend. Dockerfile для серверної частини повинен містити інструкції для копіювання коду проєкту, встановлення залежностей та запуску основного серверного процесу. Аналогічно, Dockerfile для клієнтської частини має описувати процес збірки статичних файлів застосунку та подальшу їх роздачу через легко-ваговий веб-сервер, такий як Nginx, або за допомогою вбудованого сервера Node.js.

Усі сервіси, визначені у docker-compose.yaml, об'єднуються в єдину віртуальну мережу з назвою app, що дозволяє їм легко взаємодіяти між собою, використовуючи імена сервісів як мережеві імена хостів. Наприклад, серверна частина зможе підключитися до бази даних, звертаючись до неї за ім'ям postgres.

Процедура розгортання системи на цільовому сервері, де попередньо встановлені Docker та Docker Compose, передбачає виконання таких кроків:

1. Передача коду проєкту на сервер.
2. Підготовка Dockerfile для клієнтської та серверної частин (якщо вони ще не створені).
3. Конфігурація змінних середовища, необхідних для роботи серверної частини (зокрема, параметри підключення до бази даних, секретний ключ для JWT), шляхом створення файлу .env у відповідній директорії (backend/src/config).
4. Актуалізація файлу docker-compose.yaml шляхом розкоментування та, за потреби, налаштування сервісів frontend та backend, включаючи визначення залежностей (наприклад, backend залежить від postgres).
5. Запуск усіх сервісів системи за допомогою команди docker-compose up --build -d, виконаної з кореневої директорії проєкту. Параметр --build ініціює примусову перебудову образів, якщо Dockerfile були змінені, а прапорець -d забезпечує запуск контейнерів у фоновому (detached) режимі.

Після успішного завершення цих дій, програмний модуль буде розгорнуто, і користувачі зможуть отримати доступ до його клієнтської частини через веб-браузер, серверна частина оброблятиме API-запити, а дані будуть надійно зберігатися в базі даних PostgreSQL. Для забезпечення стабільної та безпечної роботи системи в продуктивному середовищі необхідно також буде додатково налаштувати мережеву інфраструктуру, включаючи брандмауери, забезпечити використання HTTPS для шифрування трафіку та організувати регулярне резервне копіювання даних.

Висновок до Розділу 3

Третій розділ кваліфікаційної роботи детально висвітлив процес практичного втілення теоретичних напрацювань та проектних рішень у функціонуючий програмний модуль, призначений для автоматизації обліку спортивних заходів та фізичної підготовки у закладах вищої освіти. Було послідовно розглянуто всі ключові етапи розробки - від вибору технологічного стеку до підготовки системи до розгортання.

На початку розділу було представлено обґрунтування вибору інструментальних засобів та технологій. Для серверної частини було обрано платформу Node.js з фреймворком Express.js та ORM Sequelize для взаємодії з базою даних PostgreSQL. Клієнтська частина реалізована як односторінковий веб-додаток з використанням бібліотеки React, системи управління станом MobX та бібліотеки компонентів Material-UI. Вибір цих технологій зумовлений їхньою сучасністю, продуктивністю, наявністю розвинених екосистем та можливістю ефективної реалізації поставлених завдань.

Далі було детально описано процес розробки основних програмних компонентів модуля. Реалізація серверної логіки включала створення RESTful API для взаємодії з клієнтською частиною, розробку моделей даних, сервісів для інкапсуляції бізнес-логіки (наприклад, управління тренувальними заняттями, обробка результатів оцінювання) та контролерів для обробки запитів. У розробці клієнтської частини основну увагу було приділено створенню інтуїтивно зрозумілих та функціональних користувацьких інтерфейсів для управління довідниками, планування та обліку занять, введення результатів, а також візуалізації аналітичних даних за допомогою діаграм та інформаційної панелі.

Особливу увагу було приділено розробці та інтеграції модуля управління доступом. Цей модуль базується на механізмах автентифікації користувачів за допомогою JWT-токенів та авторизації на основі визначених ролей, що забезпечує надійний захист даних та розмежування прав доступу до функціоналу системи як на серверній, так і на клієнтській стороні.

На завершення розділу було розглянуто підхід до розгортання розробленого програмного модуля з використанням технології контейнеризації Docker та інструменту Docker Compose. Такий підхід забезпечує портативність, ізоляцію компонентів системи та спрощує процес її розгортання у різних середовищах.

Таким чином, у третьому розділі було продемонстровано повний цикл перетворення теоретичних проектних рішень на практично реалізований програмний продукт, що відповідає поставленим у технічному завданні вимогам та готовий до впровадження для автоматизації обліку спортивних заходів у ЗВО.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було успішно вирішено актуальну задачу підвищення ефективності управління фізичною підготовкою та спортивно-масовою роботою у закладах вищої освіти, зокрема у військових вищих навчальних закладах, шляхом розробки спеціалізованого програмного модуля для автоматизації обліку спортивних заходів.

На першому етапі було проведено комплексний аналіз предметної області. Розглянуто нормативно-правове регулювання процесів організації фізичної підготовки у ЗВО та ВВНЗ України, зокрема ключові положення Наказу Міністерства оборони України № 120 від 15.02.2024 та Інструкції з фізичної підготовки в системі Міністерства оборони України. Було виявлено недоліки існуючої практики обліку, пов'язані переважно з ручною обробкою даних та паперовим документообігом, що не відповідає сучасним вимогам до оперативності та аналітичних можливостей. Аналіз наявних в Україні підходів до автоматизації та вивчення світового досвіду підтвердили доцільність та актуальність створення запропонованого програмного модуля. Завершальним етапом стало формування детального технічного завдання на проектування системи.

Другий розділ роботи було присвячено теоретичним засадам та безпосередньому проектуванню програмного модуля. Обґрунтовано вибір клієнт-серверної архітектури з монолітною реалізацією серверної частини на Node.js/Express.js та клієнтською частиною на React, що забезпечує централізоване управління даними, доступність через веб-клієнт та відносну простоту розробки для визначеного функціоналу. Розроблено узагальнену структурну схему системи та детальну реляційну модель бази даних PostgreSQL. Особливу увагу приділено принципам та методам забезпечення інформаційної безпеки, зокрема автентифікації користувачів за допомогою JWT токенів та розмежуванню доступу на основі ролей: "ADMIN", "DEPARTMENT\_EMPLOYEE", "COMMANDER" та "INSTRUCTOR".

У третьому розділі було описано практичну реалізацію спроєктованого програмного модуля. Детально розглянуто вибір технологічного стеку, включаючи Node.js, Express.js, Sequelize для серверної частини, та React з Material-UI, MobX для клієнтської частини. Описано розробку серверної логіки, включаючи API, та реалізацію бізнес-логіки для управління спортивними заходами та оцінками. Представлено реалізацію клієнтських інтерфейсів для управління даними, відображення статистики, календаря подій та введення результатів. Продемонстровано інтеграцію модуля управління доступом та розмежування прав користувачів. Також розглянуто процес розгортання системи за допомогою Docker та Docker Compose.

Результатом кваліфікаційної роботи є розроблений програмний модуль, який дозволяє автоматизувати збір, обробку, зберігання та аналіз даних про спортивні заходи та фізичну підготовленість особового складу ЗВО (ВВНЗ). Система забезпечує управління довідниками, планування та облік тренувальних занять, фіксацію результатів виконання нормативів, формування аналітичної звітності та візуалізацію даних. Впроваджені механізми автентифікації та авторизації гарантують безпеку та контрольований доступ до інформації.

Впровадження розробленого програмного модуля сприятиме підвищенню оперативності отримання актуальної інформації, покращенню якості планування та контролю за процесом фізичної підготовки, оптимізації використання ресурсів навчального закладу та, як наслідок, підтриманню високого рівня фізичної готовності курсантів (студентів), що є важливим для їхньої майбутньої професійної діяльності.

Подальший розвиток системи може включати розширення аналітичних можливостей, інтеграцію з іншими інформаційними системами навчального закладу, розробку мобільного додатку для зручного доступу інструкторів та курсантів, а також впровадження елементів гейміфікації для підвищення мотивації до занять фізичною підготовкою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* + 1. Про затвердження Інструкції з обліку військового майна у Збройних Силах України: Наказ Міністерства оборони України від 17 серпня 2017 р. № 440. URL: <https://zakon.rada.gov.ua/laws/show/z1111-17>.
    2. Про внесення змін до Інструкції про порядок використання автомобільної техніки у Збройних Силах України: Наказ Міністерства оборони України від 05 лютого 2018 р. № 70.
    3. Методичний посібник з планування, організації та проведення заходів щодо переведення озброєння та військової техніки на літню (зимову) експлуатацію у Збройних Силах України / Головне управління логістики ГШ ЗС України. Київ, 2021.
    4. Таненбаум Е., Уэзеролл Д. Компьютерные сети. 5-е изд. СПб.: Питер, 2012. 960 с.
    5. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 4-е изд. СПб.: Питер, 2010. 944 с.
    6. Кравчук С.О., Гайдамашко В.М., Вовкодав Р.В. Проектування баз даних: навч. посіб. Житомир: ЖДТУ, 2017. 200 с.
    7. Дейт К. Дж. Введение в системы баз данных. 8-е изд. : пер. с англ. М. : Издательский дом "Вильямс", 2005. 1328 с.
    8. Гаевский А.Ю. Безопасность сетей на основе Microsoft Windows Server 2003. М.: НТ Пресс, 2005. 528 с.
    9. Гайдук С.В., Корченко О.Г., Козачок Я.В. Інформаційна безпека: інженерно-технічний аспект: Навчальний посібник. Житомир: ЖВІ НАУ, 2011. 400 с.
    10. Фрімен Е., Фрімен Е. Патерни проектування. Head First. O'Reilly, 2004. 650 с.
    11. Мартін Р. Чистий код: створення, аналіз та рефакторинг. Prentice Hall, 2008. 464 с.
    12. Мартін Р. Чиста архітектура: мистецтво розробки програмного забезпечення. Prentice Hall, 2017. 432 с.
    13. Офіційний сайт Node.js. [Електронний ресурс]. – Режим доступу: <https://nodejs.org/> . – Назва з екрана.
    14. Офіційна документація Express.js. [Електронний ресурс]. – Режим доступу: <https://expressjs.com/> . – Назва з екрана.
    15. Офіційна документація React. [Електронний ресурс]. – Режим доступу: <https://reactjs.org/> . – Назва з екрана.
    16. Офіційна документація PostgreSQL. [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/docs/> . – Назва з екрана.
    17. Офіційна документація Sequelize ORM. [Електронний ресурс]. – Режим доступу: <https://sequelize.org/> . – Назва з екрана.
    18. Офіційна документація Docker. [Електронний ресурс]. – Режим доступу: <https://docs.docker.com/> . – Назва з екрана.
    19. JSON Web Tokens - jwt.io. [Електронний ресурс]. – Режим доступу: <https://jwt.io/> . – Назва з екрана.

ДОДАТОК А

Лістинг програмного коду

TrainingSession

const { DataTypes } = require('sequelize');

const sequelize = require('../config/settingsDB');

const TrainingSession = sequelize.define('TrainingSession', {

session\_id: {

type: DataTypes.BIGINT,

primaryKey: true,

autoIncrement: true,

field: 'session\_id'

},

session\_type: {

type: DataTypes.ENUM('TRAINING', 'STANDARDS\_ASSESSMENT', 'UNIT\_TRAINING'),

allowNull: false,

field: 'session\_type'

},

start\_datetime: {

type: DataTypes.DATE,

allowNull: false,

field: 'start\_datetime'

},

end\_datetime: {

type: DataTypes.DATE,

allowNull: false,

field: 'end\_datetime'

},

location\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'location\_id'

},

conducted\_by\_user\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'conducted\_by\_user\_id'

},

unit\_id: {

type: DataTypes.BIGINT,

allowNull: true,

field: 'unit\_id'

}

}, {

tableName: 'training\_sessions',

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

module.exports = TrainingSession;

StandardAssessment

const { DataTypes } = require('sequelize');

const sequelize = require('../config/settingsDB');

const StandardAssessment = sequelize.define('StandardAssessment', {

assessment\_id: {

type: DataTypes.BIGINT,

primaryKey: true,

autoIncrement: true,

field: 'assessment\_id'

},

session\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'session\_id'

},

military\_person\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'military\_person\_id'

},

exercise\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'exercise\_id'

},

score: {

type: DataTypes.ENUM(

'PASSED', // 'зараховано'

'EXCELLENT', // 'відмінно'

'GOOD', // 'добре'

'SATISFACTORY', // 'задовільно'

'FAILED' // 'не зараховано'

),

allowNull: false,

field: 'score'

},

assessment\_datetime: {

type: DataTypes.DATE,

allowNull: false,

defaultValue: DataTypes.NOW,

field: 'assessment\_datetime'

},

notes: {

type: DataTypes.TEXT,

allowNull: true,

field: 'notes'

}

}, {

tableName: 'standard\_assessments',

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

module.exports = StandardAssessment;

SessionExercise

const { DataTypes } = require('sequelize');

const sequelize = require('../config/settingsDB');

const SessionExercise = sequelize.define('SessionExercise', {

session\_exercise\_id: {

type: DataTypes.BIGINT,

primaryKey: true,

autoIncrement: true,

field: 'session\_exercise\_id'

},

session\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'session\_id'

},

exercise\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'exercise\_id'

},

order\_in\_session: {

type: DataTypes.INTEGER,

allowNull: false,

defaultValue: 1,

field: 'order\_in\_session'

}

}, {

tableName: 'session\_exercises',

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

module.exports = SessionExercise;

MilitaryPersonnel

const { DataTypes } = *require*('sequelize');  
const *sequelize* = *require*('../config/settingsDB');  
  
const *MilitaryPersonnel* = *sequelize*.define('MilitaryPersonnel', {  
 military\_person\_id: {  
 type: DataTypes.*BIGINT*,  
 primaryKey: true,  
 autoIncrement: true,  
 field: 'military\_person\_id'  
 },  
 first\_name: {  
 type: DataTypes.*TEXT*,  
 allowNull: false,  
 field: 'first\_name'  
 },  
 last\_name: {  
 type: DataTypes.*TEXT*,  
 allowNull: false,  
 field: 'last\_name'  
 },  
 rank: {  
 type: DataTypes.*TEXT*,  
 allowNull: true,  
 field: 'rank'  
 },  
 date\_of\_birth: {  
 type: DataTypes.*DATEONLY*,  
 allowNull: false,  
 field: 'date\_of\_birth'  
 },  
 unit\_id: {  
 type: DataTypes.*BIGINT*,  
 allowNull: false,  
 field: 'unit\_id'  
 }  
}, {  
 tableName: 'military\_personnel',  
 timestamps: true,  
 createdAt: 'created\_at',  
 updatedAt: 'updated\_at'  
});  
  
*module*.exports = *MilitaryPersonnel*;

StandardAssessment

const { DataTypes } = require('sequelize');

const sequelize = require('../config/settingsDB');

const StandardAssessment = sequelize.define('StandardAssessment', {

assessment\_id: {

type: DataTypes.BIGINT,

primaryKey: true,

autoIncrement: true,

field: 'assessment\_id'

},

session\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'session\_id'

},

military\_person\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'military\_person\_id'

},

exercise\_id: {

type: DataTypes.BIGINT,

allowNull: false,

field: 'exercise\_id'

},

score: {

type: DataTypes.ENUM(

'PASSED', // 'зараховано'

'EXCELLENT', // 'відмінно'

'GOOD', // 'добре'

'SATISFACTORY', // 'задовільно'

'FAILED' // 'не зараховано'

),

allowNull: false,

field: 'score'

},

assessment\_datetime: {

type: DataTypes.DATE,

allowNull: false,

defaultValue: DataTypes.NOW,

field: 'assessment\_datetime'

},

notes: {

type: DataTypes.TEXT,

allowNull: true,

field: 'notes'

}

}, {

tableName: 'standard\_assessments',

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

module.exports = StandardAssessment;

Exercise

const { DataTypes } = require('sequelize');

const sequelize = require('../config/settingsDB');

const Exercise = sequelize.define('Exercise', {

exercise\_id: {

type: DataTypes.BIGINT,

primaryKey: true,

autoIncrement: true,

field: 'exercise\_id'

},

exercise\_name: {

type: DataTypes.TEXT,

allowNull: false,

unique: true,

field: 'exercise\_name'

},

description: {

type: DataTypes.TEXT,

allowNull: true,

field: 'description'

}

}, {

tableName: 'exercises',

timestamps: true,

createdAt: 'created\_at',

updatedAt: 'updated\_at'

});

module.exports = Exercise;

User.js

const {DataTypes} = require('sequelize');

const sequelize = require('../config/settingsDB');

const User = sequelize.define('User', {

name: {type: DataTypes.STRING, allowNull: false},

username: {type: DataTypes.STRING, allowNull: false},

passwordHash: {type: DataTypes.STRING, allowNull: false},

role: {type: DataTypes.ENUM('UNIT\_COMMANDER', 'COMMANDER', 'ADMIN', 'DUTY\_STAFF'), allowNull: false}

});

module.exports = User;

trainingSessionService.js

const sequelize = require('../config/settingsDB');

const TrainingSession = require('../models/trainingSession.model');

const SessionExercise = require('../models/sessionExercise.model');

const AppError = require("../errors/AppError");

const sessionExerciseService = require("./sessionExerciseService");

const standardAssessmentService = require("./standardAssessmentService");

module.exports = {

async createTrainingSession(sessionData) {

const { exercises, ...sessionDetails } = sessionData;

const transaction = await sequelize.transaction();

try {

const newTrainingSession = await TrainingSession.create(sessionDetails, { transaction });

if (exercises && exercises.length > 0) {

const sessionExercisesData = exercises.map(ex => ({

session\_id: newTrainingSession.session\_id,

exercise\_id: ex.exercise\_id,

order\_in\_session: ex.order\_in\_session || 1

}));

await SessionExercise.bulkCreate(sessionExercisesData, { transaction });

}

await transaction.commit();

return await this.getTrainingSessionById(newTrainingSession.session\_id);

} catch (error) {

await transaction.rollback();

if (error instanceof AppError) throw error;

throw new AppError(`Could not create training session: ${error.message}`, 500);

}

},

async getAllTrainingSessions(filters = {}) {

const trainingSessions = await TrainingSession.findAll({

where: filters,

include: [{

model: sequelize.models.Exercise,

as: 'exercises',

through: { attributes: ['order\_in\_session'] }

}]

});

if (!trainingSessions || trainingSessions.length === 0) {

return null;

}

return trainingSessions;

},

async getTrainingSessionById(id) {

const trainingSession = await TrainingSession.findByPk(id, {

include: [{

model: sequelize.models.Exercise,

as: 'exercises',

through: { attributes: ['order\_in\_session'] }

}]

});

if (!trainingSession) {

throw new AppError(`Training Session with ID ${id} not found`, 404);

}

return trainingSession;

},

async updateTrainingSession(id, updateData) {

const { exercises, ...sessionDetails } = updateData;

const transaction = await sequelize.transaction();

try {

const trainingSession = await TrainingSession.findByPk(id, { transaction });

if (!trainingSession) {

await transaction.rollback();

throw new AppError(`Training Session with ID ${id} not found`, 404);

}

await trainingSession.update(sessionDetails, { transaction });

if (exercises !== undefined) {

await SessionExercise.destroy({ where: { session\_id: id }, transaction });

if (exercises.length > 0) {

const sessionExercisesData = exercises.map(ex => ({

session\_id: id,

exercise\_id: ex.exercise\_id,

order\_in\_session: ex.order\_in\_session || 1

}));

await SessionExercise.bulkCreate(sessionExercisesData, { transaction });

}

}

await transaction.commit();

return await this.getTrainingSessionById(id);

} catch (error) {

await transaction.rollback();

if (error instanceof AppError) throw error;

throw new AppError(`Could not update training session: ${error.message}`, 500);

}

},

async deleteTrainingSession(id, options = {}) {

const transaction = options.transaction || await sequelize.transaction();

try {

const trainingSession = await TrainingSession.findByPk(id, { transaction });

if (!trainingSession) {

if (!options.transaction) await transaction.rollback();

throw new AppError(`Training Session with ID ${id} not found`, 404);

}

await sessionExerciseService.deleteSessionExercisesBySessionId(id, { transaction });

await standardAssessmentService.deleteAssessmentsBySessionId(id, { transaction });

await trainingSession.destroy({ transaction });

if (!options.transaction) await transaction.commit();

return { message: `Training Session with ID ${id} and its associated exercises/assessments deleted successfully` };

} catch (error) {

if (!options.transaction) await transaction.rollback();

if (error instanceof AppError) throw error;

throw new AppError(`Could not delete training session: ${error.message}`, 500);

}

}

};

standardAssessmentService.js

const StandardAssessment = require('../models/standardAssessment.model');

const AppError = require("../errors/AppError");

const sequelize = require('../config/settingsDB');

module.exports = {

async createStandardAssessment(assessmentData) {

const existingAssessment = await StandardAssessment.findOne({

where: {

session\_id: assessmentData.session\_id,

military\_person\_id: assessmentData.military\_person\_id,

exercise\_id: assessmentData.exercise\_id

}

});

if (existingAssessment) {

throw new AppError(`Assessment for this exercise (ID: ${assessmentData.exercise\_id}), person (ID: ${assessmentData.military\_person\_id}), and session (ID: ${assessmentData.session\_id}) already exists. Consider updating.`, 400);

}

return await StandardAssessment.create(assessmentData);

},

async getAllStandardAssessments(filters = {}) {

const includeOptions = [

{ model: sequelize.models.TrainingSession, as: 'trainingSession' },

{ model: sequelize.models.MilitaryPersonnel, as: 'militaryPersonnel' },

{ model: sequelize.models.Exercise, as: 'exerciseDetails' }

];

const assessments = await StandardAssessment.findAll({

where: filters,

});

if (!assessments || assessments.length === 0) {

return null;

}

return assessments;

},

async getStandardAssessmentById(id) {

const assessment = await StandardAssessment.findByPk(id, {

});

if (!assessment) {

throw new AppError(`Standard Assessment with ID ${id} not found`, 404);

}

return assessment;

},

async updateStandardAssessment(id, updateData) {

const assessment = await StandardAssessment.findByPk(id);

if (!assessment) {

throw new AppError(`Standard Assessment with ID ${id} not found`, 404);

}

if (updateData.session\_id && updateData.session\_id !== assessment.session\_id) {

throw new AppError("Cannot change session\_id for an existing assessment. Recreate instead.", 400);

}

await assessment.update(updateData);

return assessment;

},

async deleteStandardAssessment(id, options = {}) {

const transaction = options.transaction || await sequelize.transaction();

try {

const assessment = await StandardAssessment.findByPk(id, { transaction });

if (!assessment) {

if (!options.transaction) await transaction.rollback();

throw new AppError(`Standard Assessment with ID ${id} not found`, 404);

}

await assessment.destroy({ transaction });

if (!options.transaction) await transaction.commit();

return { message: `Standard Assessment with ID ${id} deleted successfully` };

} catch (error) {

if (!options.transaction) await transaction.rollback();

if (error instanceof AppError) throw error;

throw new AppError(`Could not delete Standard Assessment with ID ${id}: ${error.message}`, 500);

}

},

async deleteAssessmentsBySessionId(sessionId, options = {}) {

const transaction = options.transaction;

if (!transaction) {

throw new AppError("Transaction is required for deleting assessments by session ID", 500);

}

const result = await StandardAssessment.destroy({

where: { session\_id: sessionId },

transaction

});

return { message: `${result} assessments for session ID ${sessionId} deleted successfully` };

},

async deleteAssessmentsByMilitaryPersonnelId(militaryPersonId, options = {}) {

const transaction = options.transaction;

if (!transaction) {

throw new AppError("Transaction is required for deleting assessments by military person ID", 500);

}

const result = await StandardAssessment.destroy({

where: { military\_person\_id: militaryPersonId },

transaction

});

return { message: `${result} assessments for military person ID ${militaryPersonId} deleted successfully` };

},

async deleteAssessmentsByExerciseId(exerciseId, options = {}) {

const transaction = options.transaction;

if (!transaction) {

throw new AppError("Transaction is required for deleting assessments by exercise ID", 500);

}

const result = await StandardAssessment.destroy({

where: { exercise\_id: exerciseId },

transaction

});

return { message: `${result} assessments for exercise ID ${exerciseId} deleted successfully` };

}

};

authController.js

const User = require('../models/User');

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const {userToDto} = require("../dtos/user.dto");

require('dotenv').config();

exports.login = async (req, res) => {

const {username, password} = req.body;

try {

const user = await User.findOne({where: {username}});

if (!user) {

return res.status(400).json({message: 'Invalid Credentials'});

}

const isMatch = await bcrypt.compare(password, user.passwordHash);

if (!isMatch) {

return res.status(400).json({message: 'Invalid Credentials'});

}

const payload = {user: userToDto(user)};

jwt.sign(payload, process.env.JWT\_SECRET, {expiresIn: 36000}, (err, token) => {

if (err) throw err;

console.log(`Created token: ${token}`);

res.json({token});

});

} catch (err) {

console.error(err.message);

res.status(500).send('Server error');

}

};

exports.getMe = async (req, res) => {

try {

if (!req.user || !req.user.user || !req.user.user.id) {

return res.status(400).json({ message: 'User ID not found in token payload' });

}

const userId = req.user.user.id;

const user = await User.findByPk(userId);

if (!user) {

return res.status(404).json({ message: 'User not found' });

}

res.json(userToDto(user));

} catch (err) {

console.error("GetMe error:", err.message);

res.status(500).send('Server error');

}

};