ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка адаптивної інформаційної системи планування мікроциклів рухової активності як засіб превенції гіподинамічних станів»

на здобуття освітнього ступеня бакалавра

зі спеціальності 121 Інженерія програмного забезпечення

освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Віктор КУЦИК
(підпис)

Виконав: здобувач вищої освіти групи ПД-44
Віктор КУЦИК

Керівник: Юрій БАЖАН

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного забезпечення

_____ Ірина ЗАМРІЙ

« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Куцику Віктору Миколайовичу

1. Тема кваліфікаційної роботи: «Розробка адаптивної інформаційної системи планування мікроциклів рухової активності як засіб превенції гіподинамічних станів»

керівник кваліфікаційної роботи викладач кафедри ІПЗ Юрій БАЖАН, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. No 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 р.

3. Вихідні дані до кваліфікаційної роботи**:** теоретичні відомості про вплив малорухливого способу життя (гіподинамії) на організм людини, фізіологічні засади планування мікроциклів рухової активності, методика оцінки

суб'єктивного сприйняття навантаження за шкалою RPE (Rate of Perceived Exertion), документація до фреймворків React та ASP.NET Core, опис побудови клієнт-серверної архітектури та проєктування реляційних баз даних, відомості про методи архітектурного захисту інформації (зокрема, використання об'єктів передачі даних DTO та ORM).

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1. Аналіз предметної області та обґрунтування вимог до інформаційної системи превенції гіподинамічних станів.
2. Обґрунтування вибору засобів реалізації та проєктування архітектури системи, бази даних і алгоритму поведінкової адаптації.
3. Опис програмної реалізації клієнт-серверного Web-застосунку для адаптивного планування мікроциклів.
4. Забезпечення безпеки інформації та тестування системи.

5. Перелік графічного матеріалу: *презентація*

1. Актуальність теми та аналіз існуючих аналогів.
2. Вимоги до програмного забезпечення та інформаційної безпеки.
3. Програмні засоби реалізації.
4. Архітектура системи та логіка взаємодії компонентів (діаграма архітектури, діаграма послідовності).
5. Проєктування бази даних (ER-діаграма).
6. Блок-схема алгоритму адаптивного планування (на основі шкали RPE).
7. Схема архітектури захисту даних (DTO, Entity Framework Core).
8. Екранні форми та програмна реалізація інтерфейсу.
9. Апробація результатів дослідження.

6. Дата видачі завдання «23» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02–28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03–07.03.2026	
3	Аналіз предметної області та обґрунтування вимог до інформаційної системи превенції гіподинамічних станів	08.03–15.03.2026	
4	Проєктування архітектури Web-застосунку, бази даних та алгоритму поведінкової адаптації	16.03–30.03.2026	
5	Програмна реалізація клієнт-серверного Web-застосунку	31.03–20.04.2026	
6	Забезпечення безпеки інформації та тестування застосунку	21.04–28.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	29.04–05.05.2026	
8	Розробка демонстраційних матеріалів	06.05–10.05.2026	
9	Попередній захист роботи	11.05-22.05.2026	

Здобувач вищої освіти _____ Віктор КУЦИК
(підпис)

Керівник
кваліфікаційної роботи _____ Юрій БАЖАН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 74 стор., 2 табл., 14 рис., 25 джерел.

Мета роботи – зниження ризиків розвитку професійних захворювань та покращення загального стану здоров'я осіб із малорухливим характером праці шляхом розробки адаптивного web-застосунку для планування рухових мікроциклів.

Об'єкт дослідження – процес автоматизації планування індивідуальної рухової активності для осіб, що ведуть малорухливий спосіб життя.

Предмет дослідження – методи, алгоритми адаптації та програмні засоби розробки інформаційної системи для генерації персоналізованих мікроциклів на основі поведінкової аналітики.

*Короткий зміст роботи***:** В роботі проаналізовано проблему гіподинамії серед працівників IT-сфери та фізіологічні закономірності побудови ефективних рухових мікроциклів. Проаналізовано існуючі програмні рішення (Stretchly, Wakeout, Nike Training Club тощо) та виявлено відсутність гнучких механізмів підлаштування під стан користувача. Розроблено алгоритм поведінкової адаптації на основі 10-бальної шкали суб'єктивного сприйняття навантаження (RPE), що дозволяє динамічно замінювати вправи на важчі або легші аналоги в межах однієї категорії.

Спроєктовано клієнт-серверну архітектуру та реляційну базу даних. Програмно реалізовані ключові функціональні можливості: генерація мікроциклів, збір зворотного зв'язку, візуалізація статистики. Забезпечено захист даних та бізнес-логіки системи від некоректного введення та SQL-ін'єкцій за допомогою патерну DTO та інструментів ORM. Проведено функціональне та інтеграційне тестування системи. В роботі використано бібліотеку React для створення інтерактивного користувацького інтерфейсу, фреймворк ASP.NET Core Web API для реалізації бізнес-логіки та СУБД MS SQL Server за допомогою Entity Framework Core для збереження даних.

Сферою використання застосунку є організація здорового робочого процесу, самостійна фізична реабілітація та профілактика гіподинамічних станів для фахівців ІТ-галузі та офісних працівників.

КЛЮЧОВІ СЛОВА: ГІПОДИНАМІЯ, РУХОВІ МІКРОЦИКЛИ, АДАПТИВНИЙ АЛГОРИТМ, RPE, REACT, ASP.NET CORE, DTO, SWAGGER.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

ІТ – Інформаційні технології

ІС – Інформаційна система

ПЗ – Програмне забезпечення

СУБД – Система управління базами даних

API – Application Programming Interface (Інтерфейс прикладного програмування)

CSS – Cascading Style Sheets (Каскадні таблиці стилів)

EF Core – Entity Framework Core

HTML – HyperText Markup Language (Мова розмітки гіпертексту)

HTTP – HyperText Transfer Protocol (Протокол передачі гіпертексту)

HTTPS – HyperText Transfer Protocol Secure (Захищений протокол передачі гіпертексту)

IDE – Integrated Development Environment (Інтегроване середовище розробки)

JS – JavaScript JSON – JavaScript Object Notation

ORM – Object-Relational Mapping (Об'єктно-реляційне відображення)

REST – Representational State Transfer (Передача репрезентативного стану)

RPE – Rate of Perceived Exertion (Оцінка суб'єктивного сприйняття навантаження)

SPA – Single Page Application (Односторінковий вебзастосунок)

SQL – Structured Query Language (Мова структурованих запитів)

XSS – Cross-Site Scripting (Міжсайтовий скриптинг)

ВСТУП

Актуальність: цифровізація сучасного суспільства та перехід значної кількості працівників на віддалений або офісний формат роботи спровокували глобальну проблему — епідемію гіподинамії. Особливо гостро це стосується фахівців ІТ-галузі, чий робочий день характеризується тривалим перебуванням у статичному сидячому положенні. Це призводить до порушень опорно-рухового апарату, зниження когнітивної продуктивності, погіршення зору та розвитку хронічної втоми. Зростання усвідомленості щодо ергономіки праці створює попит на ефективні інструменти профілактики. Використання інтелектуальних вебзастосунків дозволяє не просто нагадувати про необхідність перерви, а й організувати процес виконання рухових мікроциклів у зручному, персоналізованому форматі безпосередньо на робочому місці.

Вебзастосунок виступає як універсальний інструмент для підтримки фізичного здоров'я користувачів під час робочого процесу. На відміну від статичних таймерів, розроблювана система об’єднує в собі базу спеціалізованих вправ (для шиї, спини, очей) та алгоритм поведінкової адаптації. Цифрові інструменти дозволяють системі аналізувати суб'єктивний стан користувача після кожної рухової паузи та динамічно коригувати подальший план. Завдяки кросплатформності, гнучкості архітектури та персоналізованому підходу до генерації мікроциклів, процес профілактики гіподинамії стає науково обґрунтованим, непомітним для робочого процесу та максимально ефективним.

Об’єкт дослідження є процес профілактики гіподинамічних станів шляхом автоматизації планування рухової активності.

Предмет дослідження є методи, алгоритми адаптації та програмні засоби розробки інформаційної системи для генерації персоналізованих мікроциклів на основі поведінкової аналітики.

Метою роботи є зниження ризиків розвитку професійних захворювань та підвищення працездатності осіб із малорухливим характером праці шляхом

розробки адаптивного web-застосунку, здатного динамічно планувати рухові паузи.

Методи дослідження: першим етапом дослідження стало системне вивчення проблеми гіподинамії та фізіологічних засад побудови мікроциклів, а також аналіз існуючих на ринку рішень (Stretchly, Wakeout тощо). Далі було проведено огляд сучасного стеку технологій. Для реалізації серверної частини (бізнес-логіки та алгоритмів) було обрано фреймворк ASP.NET Core мовою C#. Для клієнтської частини (фронтенду) використано бібліотеку React. Як об’єктно-реляційну систему управління базами даних обрано MS SQL Server із застосуванням ORM-технології Entity Framework Core. Всі компоненти об'єднано за допомогою REST API. Архітектуру програмного забезпечення та логіку бази даних спроєктовано з використанням UML- та ER-діаграм. Для забезпечення цілісності даних та безпеки API реалізовано архітектурний захист на базі об'єктів передачі даних (DTO) та автоматичної параметризації запитів.

Наукова новизна роботи полягає у розробці унікального алгоритму поведінкової адаптації рухових мікроциклів. Особливістю системи є використання шкали суб'єктивного сприйняття навантаження (RPE від 1 до 10) як основного метричного показника. На відміну від існуючих аналогів, які обмежуються зміною тривалості таймера, запропонований алгоритм здійснює автоматичну заміну вправи на більш складний або легший аналог того самого типу (наприклад, для шийного відділу), базуючись на зворотному зв'язку користувача.

Практична значущість результатів полягає у можливості використання створеного вебзастосунку IT-компаніями або індивідуальними користувачами для організації здорового робочого процесу. Розроблена система є готовим інструментом, що допомагає інтегрувати безпечну та адаптивну фізичну активність у щоденну рутину без використання додаткового спортивного інвентарю.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ОБҐРУНТУВАННЯ ВИМОГ ДО СИСТЕМИ

1.1 Проблема гіподинамії в ІТ-сфері та фізіологічні засади превенції

Перехід значної частки фахівців, зокрема працівників ІТ-сфери (програмістів, тестувальників, системних адміністраторів, аналітиків), на дистанційний або виключно офісний формат роботи спровокував безпрецедентне зниження рівня щоденної фізичної активності.

Гіподинамія (від грец. hypo — під, знизу і dynamis — сила) — це патологічний стан, який виникає внаслідок обмеження рухової активності та характеризується порушенням функцій опорно-рухового апарату, кровообігу, дихання та травлення.

Для спеціалістів галузі інформаційних технологій, чий робочий графік передбачає безперервне перебування у сидячому положенні від 8 до 12 годин на добу, ця проблема набула характеру професійної епідемії.

З точки зору біомеханіки, еволюційно людське тіло не пристосоване до тривалого статичного сидіння. Під час роботи за комп'ютером основне навантаження припадає на шийний та поперековий відділи хребта. Тривала статична напруга м'язів призводить до їх спазмування, порушення мікроциркуляції крові та лімфи, що з часом ініціює дегенеративно-дистрофічні зміни у міжхребцевих дисках (остеохондроз, протрузії, грижі).

Крім того, специфічне положення рук під час роботи з клавіатурою та мишею створює передумови для розвитку тунельного синдрому зап'ястя, а постійна концентрація погляду на моніторі викликає спазм акомодації та синдром сухого ока.

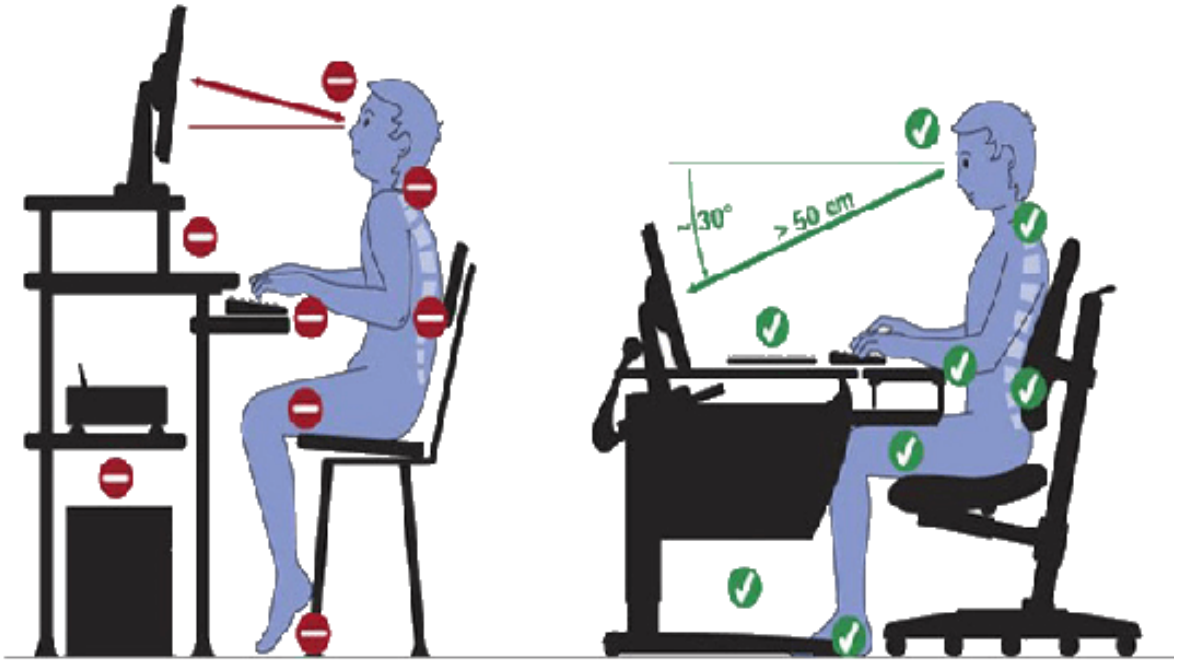


Рис. 1.1 Біомеханічні зони ризику при тривалій роботі за комп'ютером

Окрім суто фізіологічних наслідків, гіподинамія має прямий негативний вплив на когнітивні функції, які є критично важливими для працівників розумової праці. Через тривале сидіння та спазм шийних м'язів погіршується кровопостачання головного мозку, що призводить до зниження рівня кисню (гіпоксії тканин).

Як наслідок, у фахівця спостерігається зниження концентрації уваги, погіршення пам'яті, швидка втомлюваність та загальне падіння продуктивності праці. Таким чином, фізичний стан ІТ-спеціаліста безпосередньо корелює з якістю виконання його професійних обов'язків.

Для кращого розуміння комплексності проблеми, основні негативні наслідки малорухливого способу життя для ІТ-фахівців систематизовано у таблиці 1.1.

Таблиця 1.1

Вплив гіподинамічних станів на системи організму та професійну діяльність

Система організму	**Фізіологічний наслідок**	**Вплив на продуктивність**
Опорно-руховий апарат	Дегенеративно-дистрофічні зміни хребта (остеохондроз), спазмування м'язів спини та шиї, розвиток тунельного синдрому.	Виникнення больових відчуттів відволікає від виконання складних алгоритмічних завдань; потреба у частих змінах пози.
Серцево-судинна система	Зниження тонусу судин, застій крові у нижніх кінцівках та органах малого таза, порушення венозного відтоку.	Швидка втомлюваність, відчуття важкості; зниження загальної витривалості під час тривалих робочих сесій.
Нервова система	Зниження кровопостачання головного мозку (гіпоксія), порушення нейронних зв'язків через монотонність статичного положення.	Сповільнення швидкості прийняття рішень, погіршення оперативної пам'яті, зниження концентрації уваги на коді.
Зоровий апарат	Спазм акомодації (синдром «комп’ютерного зору»), сухість слизової оболонки, зниження гостроти зору.	Труднощі при читанні технічної документації та коду; підвищена кількість помилок через зорову втому.
Обмін речовин	Сповільнення метаболізму, підвищення рівня глюкози в крові, розвиток інсулінорезистентності та зайвої ваги.	Загальна млявість, сонливість після прийому їжі, зниження мотивації до ініціативної професійної діяльності.
Психоемоційна сфера	Підвищення рівня стресу, дратівливість, розвиток синдрому хронічної втоми та апатії.	Погіршення комунікації всередині команди, зниження творчого потенціалу та

У сучасній ергономіці праці існує поширена, проте хибна думка, що відвідування тренажерного залу кілька разів на тиждень здатне повністю нівелювати шкоду від щоденного багатогодинного сидіння. Сучасні медичні дослідження виділяють феномен «активного лежня» (Active Couch Potato) — стану, коли людина виконує норму інтенсивних тренувань, але залишається малорухливою протягом робочого дня.

Доведено, що тривале безперервне сидіння (понад 2 години поспіль) запускає патологічні метаболічні процеси, які неможливо компенсувати вечірнім тренуванням. Єдиним ефективним методом профілактики є регулярне переривання статичної пози за допомогою короткочасних рухових сесій — мікроциклів.

Руховий мікроцикл — це збалансований комплекс легких фізичних вправ тривалістю від 3 до 5 хвилин, який виконується безпосередньо на робочому місці кожні 45–60 хвилин. Метою такого мікроциклу є не тренування м'язів на гіпертрофію чи витривалість, а зняття статичного спазму, відновлення кровообігу, розтягнення скорочених м'язових груп (наприклад, грудних м'язів) та активізація м'язів-антагоністів (наприклад, м'язів спини).

Попри очевидну користь мікроциклів, впровадження їх у щоденну рутину стикається з проблемою швидкої втрати мотивації у користувачів. Більшість існуючих на ринку програмних рішень виконують лише функцію жорсткого таймера. Вони сигналізують про необхідність перерви або пропонують статичний, незмінний набір вправ. Такий підхід ігнорує індивідуальні особливості користувача: рівень його втоми, наявність хронічного болю у конкретних зонах (наприклад, гострий біль у шиї) або адаптацію до навантаження з часом.

Для розв'язання цієї проблеми інформаційна система повинна бути адаптивною та орієнтуватися на постійний збір поведінкової аналітики. Найбільш релевантним інструментом для оцінки поточного фізичного стану є адаптована шкала суб'єктивного сприйняття навантаження та дискомфорту (Rate of Perceived Exertion — RPE), розширена до діапазону від 1 до 10 балів. У

контексті офісної гіподинамії ця шкала дозволяє користувачеві кількісно оцінити рівень скутості або болю у певній м'язовій зоні після виконання запропонованого мікроциклу.

Якщо користувач оцінює виконання вправи високим балом (наприклад, 8–10, що свідчить про значний дискомфорт або занадто легке виконання, яке не приносить полегшення), система не повинна просто збільшувати час виконання тієї ж вправи, як це реалізовано у примітивних таймерах.

Натомість інтелектуальний алгоритм має здійснити пошук у базі даних та автоматично замінити поточну вправу на її структурний аналог у межах тієї ж категорії (наприклад, «Розтяжка шиї»), але з іншим рівнем фізичної складності. Це вимагає розробки складної реляційної бази даних, де рівень складності є незмінним атрибутом кожної окремої вправи.

Отже, аналіз предметної галузі доводить, що гіподинамія в IT-сфері є комплексною проблемою, яка потребує високотехнологічного рішення. Створення сучасного вебзастосунку, здатного не лише нагадувати про перерви, а й динамічно формувати та коригувати плани тренувань на основі зворотного зв'язку за шкалою RPE (1-10) та інтелектуальної заміни вправ за складністю, є необхідним і своєчасним кроком у розвитку систем підтримки здоров'я на робочому місці.

1.2 Аналіз існуючих програмних рішень та обґрунтування необхідності розробки нової системи

Вирішення проблеми гіподинамії в умовах офісної або дистанційної роботи неможливе без використання сучасних інформаційних технологій. На сьогодні ринок програмного забезпечення пропонує широкий спектр продуктів у категоріях «Health & Fitness» (Здоров'я та фітнес) та «Productivity» (Продуктивність). Проте детальний аналіз показує, що більшість існуючих систем тяжіють до однієї з двох крайнощів: це або примітивні таймери перерв, або важкі фітнес-застосунки для повноцінних спортивних тренувань.

Для визначення архітектурних та функціональних вимог до розроблюваної системи було проведено дослідження найбільш популярних представників обох категорій: Stretchly, Wakeout, Nike Training Club (NTC) та Freeletics.

1.2.1 Огляд застосунків-таймерів продуктивності (на прикладі Stretchly)

Stretchly — це кросплатформний застосунок з відкритим вихідним кодом (Open Source), призначений для нагадування користувачам про необхідність робити перерви під час роботи за комп'ютером. Головною перевагою цього програмного продукту є його гнучкість у налаштуванні часових інтервалів: система розрізняє «мікро-перерви» блокуючи екран або виводячи ненав'язливі спливаючі вікна з текстовими підказками щодо виконання легких розтяжок.

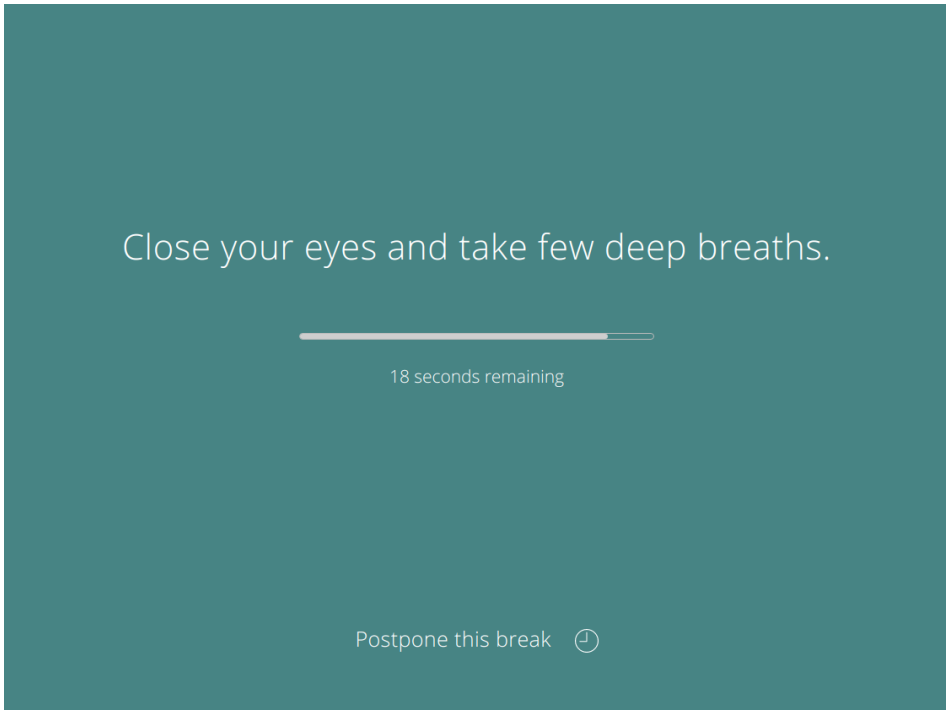


Рис. 1.2 Візуальне оформлення спливаючого вікна таймера Stretchly

Незважаючи на ефективність у дотриманні режиму праці та відпочинку, Stretchly має суттєві функціональні недоліки в контексті персоналізованої превенції гіподинамії. Система є абсолютно статичною: вона пропонує

випадковий набір текстових вправ із заздалегідь закладеної бази, ігноруючи індивідуальні фізичні потреби користувача.

Відсутній механізм збору зворотного зв'язку — програма не фіксує, чи допомогла перерва зняти напругу з шийного відділу чи очей. Відповідно, відсутня будь-яка поведінкова адаптація, що з часом призводить до звикання (так званої «банерної сліпоти») та ігнорування нагадувань користувачем.

1.2.2 Огляд каталогів офісних активностей (на прикладі Wakeout)

Wakeout позиціонується як застосунок для створення креативних і коротких тренувань безпосередньо на робочому місці. Перевагою цього рішення є наявність величезної бази високоякісного інтерактивного відеоконтенту. Вправи спроєктовані таким чином, щоб їх можна було виконувати не встаючи зі стільця, використовуючи як інвентар офісні предмети (кавове горнятко, стіл тощо). Застосунок активно використовує гейміфікацію для підтримки мотивації користувачів (див. Рис. 1.3).

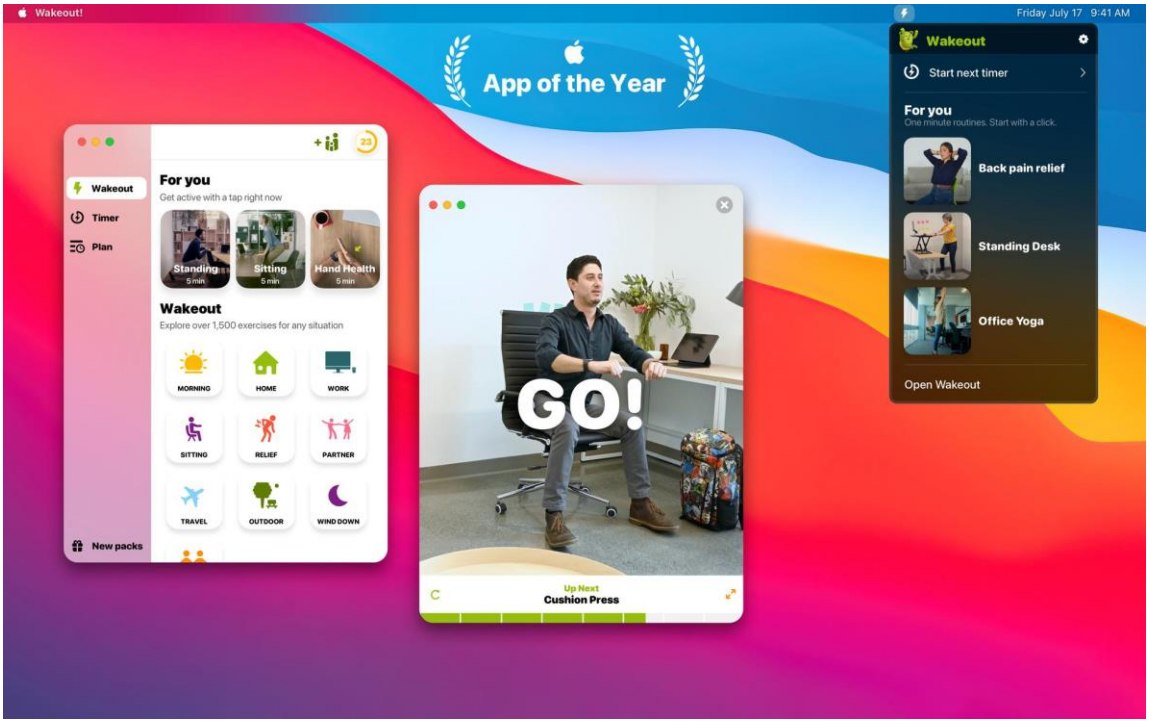


Рис. 1.3 Демонстрація виконання мікроциклу в програмі Wakeout)

Водночас, недоліками Wakeout є його комерційна закритість (доступ лише за дорогою платною підпискою) та орієнтація переважно на екосистему Apple (iOS/macOS/watchOS), що обмежує доступність для користувачів інших платформ.

З медичної точки зору, застосунок має розважальний характер і не є інструментом глибокої фізіологічної адаптації. У ньому не реалізовано механізм оцінки складності вправ на основі метрик дискомфорту (таких як RPE), тому він не здатний цілеспрямовано працювати з хронічним болем у попереку чи іншими специфічними проявами гіподинамії.

1.2.3 Аналіз систем спортивного моделювання (Nike Training Club, Freeletics)

Класичні фітнес-застосунки, такі як Nike Training Club та Freeletics, використовують складні алгоритми штучного інтелекту для підбору інтенсивності тренувань, що є їхньою беззаперечною перевагою. Однак вони орієнтовані на важкі спортивні сесії тривалістю від 15 до 60 хвилин.

Концепція цих застосунків передбачає наявність спортивної форми, розігрів м'язів, інтенсивне потовиділення та використання специфічного інвентарю. У контексті проблеми, що розглядається (потреба у 3-хвилинній руховій паузі в офісному одязі для зняття спазму з очей та шиї), використання таких систем є неможливим та недоцільним.

1.2.4 Порівняльний аналіз та висновки

Для систематизації отриманих результатів дослідження, порівняльний аналіз наявних програмних засобів та розроблюваної адаптивної системи мікроциклів (АСМ) наведено у таблиці 1.2.

Таблиця 1.2

Порівняльний аналіз функціональних можливостей існуючих програмних рішень

Критерій	**Nike Training Club**	**Stretchly**	**Wakeout**	**Freeletics**	**ACM**
Тип системи	База фітнес-тренувань	Таймер перерв для ПК	Каталог офісних вправ	ШІ-фітнес тренер	Адаптивна система мікроциклів
Адаптація за рівнем скутості (RPE)	–	–	–	–	+
Фокус на превенції гіподинамії	–	+	+	–	+
Динамічний розрахунок часу вправи	–	–	–	–	+
Врахування наявного інвентарю/локації	+	–	+	+	+
Аналітичні можливості стану	+	–	–	+	+
Автоматична генерація мікроциклів	–	–	–	+	+

Зважаючи на проведений аналіз, можна зробити висновок, що на сучасному ринку інформаційних технологій відсутня комплексна кросплатформна Web-система, яка б ефективно поєднувала планувальник робочих перерв із розумною адаптацією контенту. Існуючі рішення або не збирають дані про стан користувача, або не вміють інтерпретувати їх для заміни вправ за рівнем фізичної складності.

Саме тому розробка власного програмного забезпечення, ядром якого є алгоритм оцінювання дискомфорту за 10-бальною шкалою RPE та динамічна заміна вправ у межах цільової м'язової групи, є науково та практично обґрунтованою. Розроблюваний застосунок повинен забезпечити кросплатформний доступ через браузер (SPA архітектура), мати надійний серверний бекенд для розрахунку адаптивних моделей та гарантувати безпеку персональних метрик користувача.

1.3 Обґрунтування функціональних та нефункціональних вимог до програмного забезпечення

На основі проведеного аналізу предметної області та виявлених недоліків існуючих програмних рішень, першочерговим етапом проєктування нової інформаційної системи є чітке визначення та формалізація вимог до неї. Оскільки розроблюваний програмний продукт знаходиться на стику технологій продуктивності та медично-профілактичних рекомендацій, він повинен відповідати суворим критеріям як щодо логіки роботи алгоритмів, так і щодо захисту персональних даних користувачів. Процес інженерії вимог у контексті даного дослідження передбачає формування вичерпного переліку функціональних та нефункціональних характеристик, які стануть базисом для подальшого вибору архітектурних патернів та засобів програмної реалізації.

Функціональні вимоги визначають безпосередню поведінку системи та спектр задач, які вона здатна вирішувати у відповідь на дії користувача. Головним завданням застосунку є автоматизація процесу планування рухових

мікроциклів з урахуванням індивідуальної реакції організму на фізичне навантаження. Ядром системи має стати інтелектуальний алгоритм, здатний динамічно змінювати контент тренування. Для забезпечення такої адаптивності система повинна оперувати структурованою базою даних вправ, де кожна одиниця активності має не лише категорію (наприклад, цільову м'язову групу), але й чітко визначений та незмінний індекс фізичної складності.

З огляду на специфіку превенції гіподинамії в ІТ-сфері, базовий набір функціональних вимог до розроблюваної системи включає наступні пункти:

1. Збір та збереження базових метрик профілю користувача, включаючи визначення цільових або проблемних зон опорно-рухового апарату (наприклад, шийний відділ, поперек, зоровий апарат).

2. Автоматична генерація первинного рухового мікроциклу на основі обраних проблемних зон із залученням вправ базового рівня складності.

3. Забезпечення інтерактивного інтерфейсу для супроводу виконання мікроциклу (таймер виконання, візуалізація або текстовий опис вправи).

4. Обов'язковий збір поведінкового зворотного зв'язку після завершення кожної сесії шляхом оцінювання рівня дискомфорту чи зусиль за 10-бальною шкалою суб'єктивного сприйняття навантаження (RPE).

5. Реалізація механізму алгоритмічної адаптації: при отриманні високих або занадто низьких балів RPE система повинна здійснювати автоматичний пошук у базі даних та замінювати поточну вправу на структурний аналог тієї ж категорії, але з вищим або нижчим рівнем складності для наступного мікроциклу.

6. Ведення історії виконаних мікроциклів та збереження оцінок RPE для можливості подальшої візуалізації прогресу та загального зниження рівня тілесної скутості користувача.

Для наочного представлення функціональних меж системи та взаємодії кінцевого користувача з основними модулями програми, доцільно використати методологію об’єктно-орієнтованого проєктування.

Рис. 1.4 UML Діаграма прецедентів (Use Case Diagram)]

Не менш важливою складовою є нефункціональні вимоги, які визначають атрибути якості системи: її продуктивність, надійність, масштабованість та рівень безпеки. Зважаючи на те, що цільовою аудиторією застосунку є офісні працівники, система повинна бути максимально доступною та не вимагати встановлення додаткового або важкого програмного забезпечення на робочі комп'ютери. Тому вимогою є реалізація продукту у вигляді вебзастосунку з архітектурою Single Page Application (SPA), що дозволить забезпечити кросплатформність та миттєвий відгук інтерфейсу прямо у вікні браузера.

Крім того, оскільки система обробляє дані, що стосуються фізичного стану здоров'я людини, критичною вимогою є забезпечення інформаційної безпеки та цілісності даних на рівні серверної частини та бази даних.

З огляду на обраний архітектурний підхід, нефункціональні вимоги до програмного забезпечення сформульовано таким чином:

1. Клієнтська частина (Frontend) повинна бути реалізована за допомогою сучасної бібліотеки побудови інтерфейсів (наприклад, React), що забезпечить асинхронне оновлення компонентів сторінки без її повного перезавантаження.

2. Серверна частина (Backend) має бути побудована на високопродуктивному кросплатформному фреймворку (ASP.NET Core), який здатен ефективно обробляти REST API запити та виконувати обчислювальні операції алгоритму адаптації.

3. Зберігання даних повинно здійснюватися у надійній реляційній системі управління базами даних (наприклад, MS SQL Server) для збереження складної ієрархії між категоріями вправ, їх складністю та логами користувачів.

4. Взаємодія між клієнтом та сервером повинна здійснюватися виключно через спеціалізовані об'єкти передачі даних (DTO) для приховування внутрішньої структури бази даних. Взаємодія програмного коду з БД повинна бути повністю захищена від атак типу SQL-ін'єкцій шляхом використання сучасних засобів ORM (Entity Framework Core).

5. Взаємодія програмного коду з базою даних повинна бути повністю захищена від атак типу SQL-ін'єкцій шляхом використання сучасних засобів об'єктно-реляційного відображення (ORM), таких як Entity Framework Core, які автоматично параметризують усі запити до СУБД.

Сформований комплекс функціональних та нефункціональних вимог повністю описує поведінку майбутньої інформаційної системи. Їх виконання гарантує створення надійного, адаптивного та безпечного програмного продукту, який технічно здатен вирішити описану раніше проблему профілактики гіподинамії в умовах офісної праці.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ ТА АЛГОРИТМУ ПОВЕДІНКОВОЇ АДАПТАЦІЇ

2.1 Проєктування клієнт-серверної архітектури та обґрунтування вибору програмних платформ

Реалізація сучасного програмного забезпечення, здатного ефективно вирішувати завдання персоналізованого планування рухової активності, вимагає ретельного проєктування його архітектурної основи. Необхідність забезпечення кросплатформного доступу, високої швидкості відгуку інтерфейсу та надійного захисту персональних медико-біологічних даних, для розробки було обрано класичну багаторівневу клієнт-серверну архітектуру. Такий підхід базується на принципі суворого розділення відповідальностей (Separation of Concerns), що дозволяє ізолювати логіку представлення даних (клієнтську частину) від обробки бізнес-правил (серверної частини) та механізмів довгострокового зберігання інформації (бази даних). Це гарантує високу модульність системи, спрощує її подальше масштабування та локалізацію можливих програмних дефектів під час тестування.

Для реалізації клієнтського рівня (Frontend) було прийнято рішення відмовитися від традиційної моделі багатосторінкових вебсайтів на користь архітектури Single Page Application (SPA). У моделі SPA браузер завантажує єдиний HTML-документ, а подальше оновлення контенту (наприклад, перемикання між таймером виконання вправи та формою оцінювання) відбувається динамічно за допомогою JavaScript без повного перезавантаження сторінки. Інструментом для розробки клієнтської частини обрано популярну бібліотеку React. Її ключова перевага полягає у використанні компонентного підходу та алгоритмів віртуального DOM (Document Object Model). Завдяки цьому інтерфейс застосунку миттєво реагує на введення користувачем оцінки за

шкалою RPE, забезпечуючи плавні анімації таймера мікроциклів, що є критично важливим для утримання уваги та мотивації користувача.

Серверна частина (Backend) виступає ядром системи, оскільки саме тут інкапсульовано всю бізнес-логіку програми, включаючи алгоритм обробки оцінок дискомфорту та генерації адаптивних планів. Для розробки серверного рівня обрано високопродуктивний кросплатформний фреймворк ASP.NET Core на базі мови програмування C#. Вибір цієї платформи обґрунтовується її вбудованими механізмами маршрутизації, підтримкою ін'єкції залежностей (Dependency Injection) та суворим контролем типів мови C#, що мінімізує кількість помилок на етапі компіляції. Серверна логіка реалізована у вигляді REST API (Representational State Transfer Application Programming Interface). Клієнтська та серверна частини взаємодіють між собою виключно шляхом обміну легковаговими пакетами даних у форматі JSON (JavaScript Object Notation) через захищений протокол передачі гіпертексту (HTTPS).

Зберігання даних є не менш відповідальним архітектурним рівнем, оскільки адаптивна система потребує ведення складної ієрархії взаємозв'язків. Кожна вправа у базі даних має належати до певної категорії (наприклад, м'язи шиї), мати фіксований рівень складності та бути пов'язаною з історією виконання конкретним користувачем. Для забезпечення цілісності (ACID-транзакцій) такої структури ідеально підходять реляційні системи управління базами даних. Оптимальним вибором стала СУБД Microsoft SQL Server. Вона забезпечує високу надійність зберігання логів RPE та профілів користувачів.

Взаємодія між сервером ASP.NET Core та MS SQL Server здійснюється за допомогою сучасного інструменту об'єктно-реляційного відображення (ORM) — Entity Framework Core. Використання ORM дозволяє оперувати записами бази даних як звичайними об'єктами C#, що суттєво пришвидшує розробку. Крім того, Entity Framework Core автоматично генерує параметризовані SQL-запити. Це архітектурне рішення на базовому рівні унеможливлює проведення атак типу SQL-ін'єкцій, гарантуючи захист системи від компрометації.

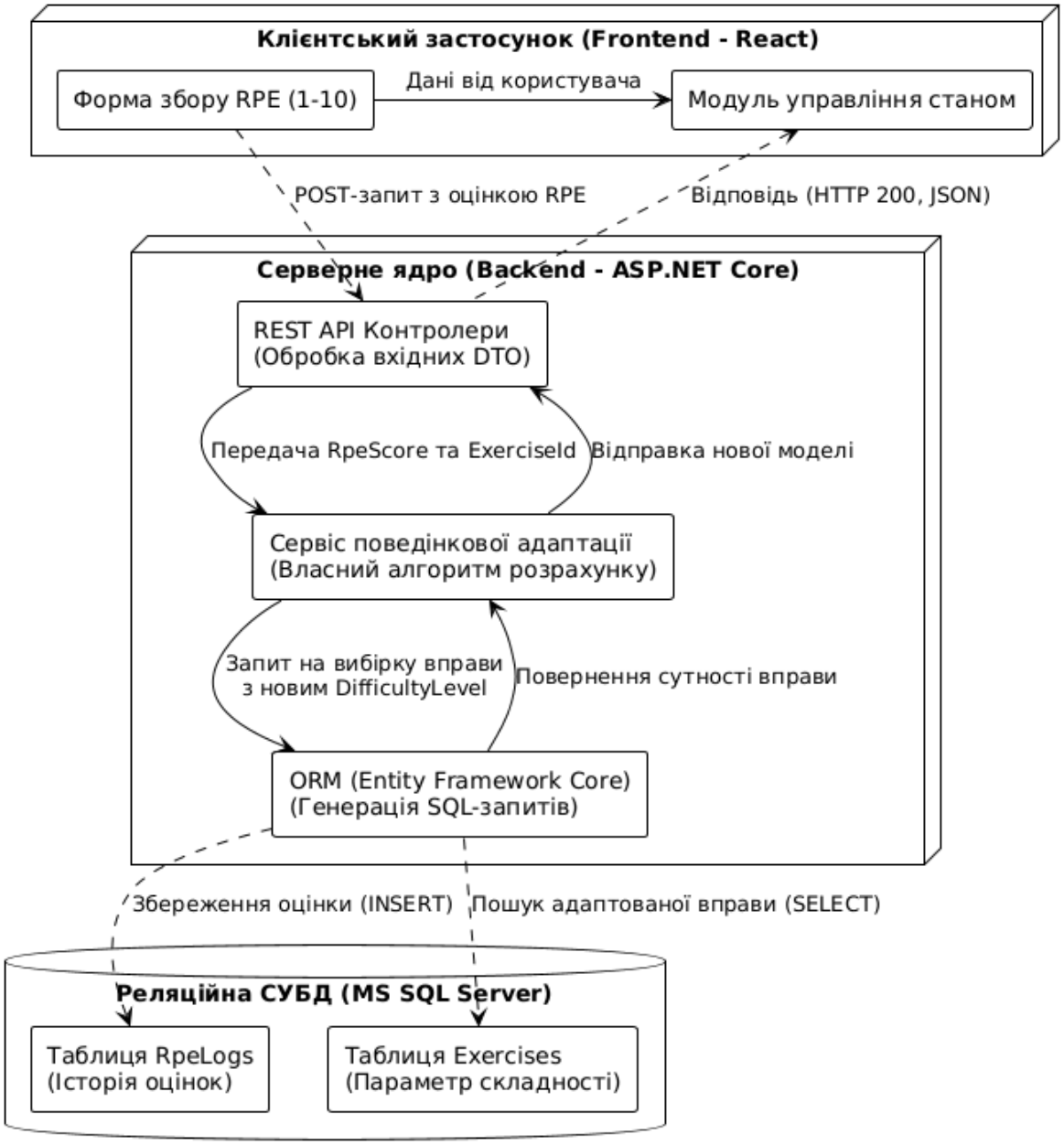


Рис. 2.1 Клієнт-серверна архітектура розроблюваного вебзастосунку

Таким чином, спроєктована трирівнева архітектура із застосуванням сучасних інструментів (React, ASP.NET Core, MS SQL Server) створює надійний, високопродуктивний та безпечний фундамент. Цей стек технологій повністю відповідає поставленим завданням і забезпечує технічну можливість для реалізації наступного, найскладнішого етапу — проєктування реляційної структури бази даних та математичної моделі поведінкового алгоритму адаптації мікроциклів.

2.2 Проєктування логічної та фізичної структури реляційної бази даних

Ефективне функціонування будь-якої адаптивної інформаційної системи залежить від якості спроєктованої моделі збереження даних. Оскільки розроблюваний вебзастосунок оперує чітко структурованими об'єктами зі сталими та прогнозованими ієрархічними зв'язками (користувачі, категорії вправ, каталог рухових активностей, журнали виконання), оптимальним рішенням стало використання реляційної моделі даних. Процес проєктування бази даних було розділено на два етапи: побудову логічної моделі, яка описує сутності та їхні атрибути незалежно від конкретної СУБД, та створення фізичної моделі, адаптованої під специфіку Microsoft SQL Server.

Основою логічної моделі розроблюваної системи є чотири головні сутності: «Користувачі» (Users), «Категорії» (Categories), «Вправи» (Exercises) та «Журнал сесій» (RpeLogs). Сутність «Користувачі» призначена для ідентифікації та збереження базового профілю особи. До атрибутів сутності "Користувачі" належать унікальний ідентифікатор, ім'я користувача та дата профілювання. У поточній версії прототипу фокус зміщено з адміністрування облікових записів на персоналізацію рухових налаштувань, тому система прив'язує аналітику RPE до унікального ідентифікатора користувача без необхідності обробки паролів.

Сутності «Категорії» та «Вправи» формують ядро довідкової інформації застосунку. Сутність «Категорії» є класифікатором, що групує рухові активності за цільовими зонами впливу на опорно-руховий апарат (наприклад, «Шийний відділ», «Грудний відділ», «Зорова гімнастика»). Сутність «Вправи» містить безпосередній опис активностей: назву, текстові або медіа-інструкції з виконання та зовнішній ключ, що вказує на відповідну категорію.

Ключовим архітектурним рішенням, яке забезпечує можливість інтелектуальної поведінкової адаптації мікроциклів, є введення до сутності «Вправи» незмінного числового атрибута — індексу фізичної складності (Difficulty Level). На відміну від статичних таймерів, де навантаження

регулюється виключно тривалістю виконання, у розроблюваній базі даних складність є характеристикою кожної вправи. Це означає, що в межах однієї категорії (наприклад, «Шийний відділ») існують вправи базового, середнього та просунутого рівнів. Саме наявність цього атрибута дозволить математичній моделі застосунку здійснювати горизонтальну заміну вправ при формуванні наступного мікроциклу, адаптуючись до поточного фізіологічного стану людини.

Зв'язуючою частиною системи є сутність «Журнал сесій» (RpeLogs), яка виконує роль сховища поведінкової аналітики. Ця таблиця фіксує кожен факт виконання вправи конкретним користувачем. До її атрибутів входять зовнішні ключі на ідентифікатори користувача та вправи, часова мітка виконання (Timestamp) та головна метрика системи — оцінка за шкалою RPE. Атрибут RPE зберігає цілочисельне значення в діапазоні від 1 до 10, яке користувач вводить після завершення рухової паузи. Накопичення цих даних формує профіль стану користувача, який алгоритм аналізує для прийняття рішень щодо підвищення чи зниження складності подальших мікроциклів.

Для забезпечення високої продуктивності та гарантування цілісності даних логічну модель було приведено до Третьої нормальної форми (3NF). Це дозволило усунути дублювання інформації та запобігти виникненню аномалій вставки, оновлення чи видалення даних. Наприклад, зміна опису вправи в таблиці «Exercises» не потребує каскадного оновлення мільйонів записів у таблиці логів «RpeLogs», оскільки вони пов'язані лише через унікальний ідентифікатор (Primary Key — Foreign Key).

Фізична реалізація бази даних у MS SQL Server здійснюється за допомогою підходу Code-First об'єктно-реляційного картографа Entity Framework Core. Усі описані сутності створюються у вигляді класів мови C#, після чого фреймворк автоматично генерує міграції та транслює їх у SQL-запити на створення відповідних таблиць, індексів та обмежень (Constraints). Логічну структуру бази даних, зв'язки між таблицями (один-до-багатьох) та типи даних атрибутів наочно представлено на ER-діаграмі.

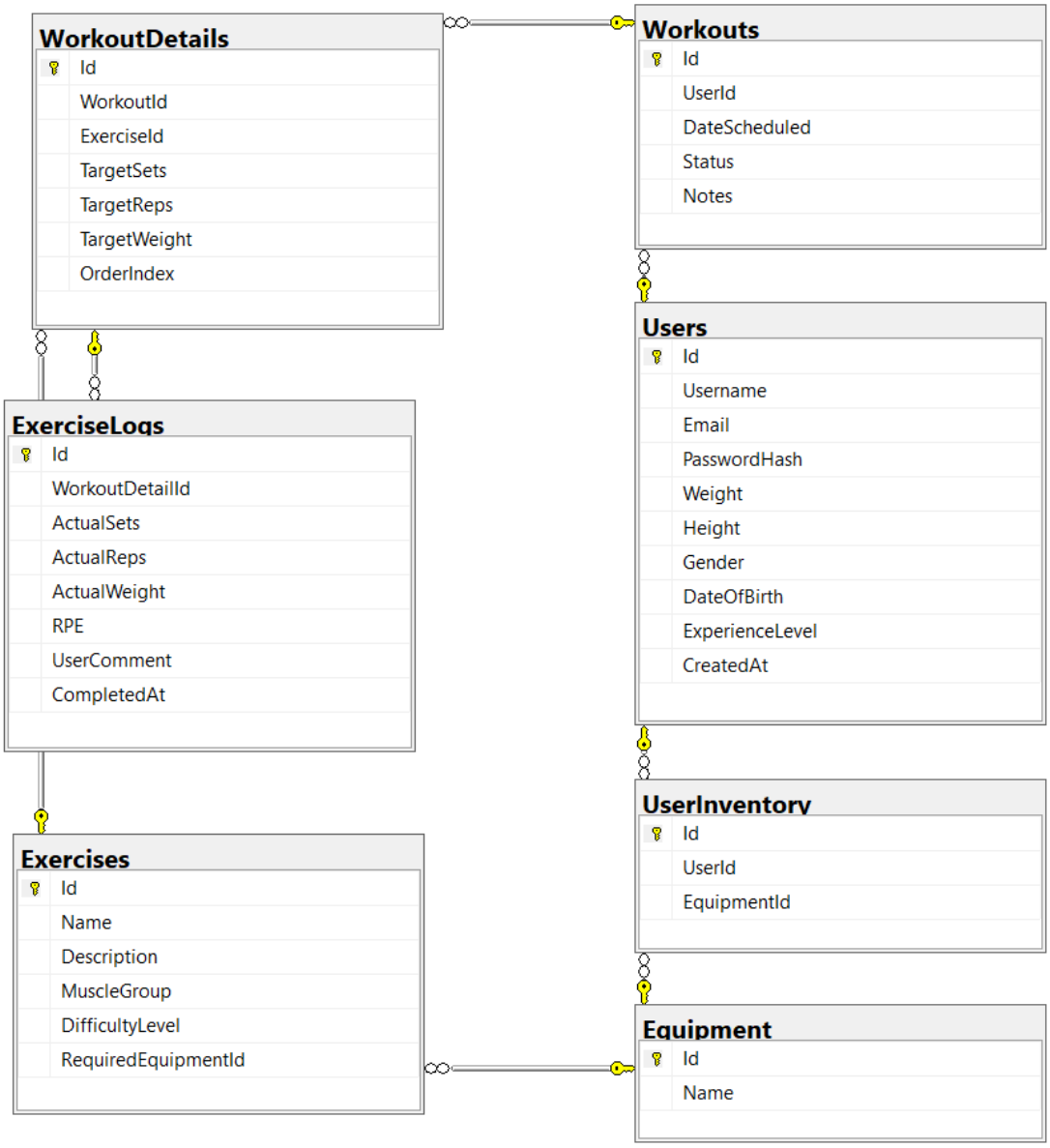


Рис. 2.2 Фізична структура реляційної бази даних

Спроєктована реляційна структура є максимально оптимізованою для швидкого читання та запису даних. Наявність жорстких зв'язків між таблицями та виокремлення параметра складності у самостійний атрибут вправи створюють ідеальні умови для програмної реалізації алгоритму адаптивного вибору мікроциклів, який є основним об'єктом наступного етапу проєктування.

2.3 Розробка математичної моделі та алгоритму поведінкової адаптації мікроциклів

В основі ефективної роботи розроблюваної інформаційної системи лежить програмно реалізована математична модель, яка забезпечує перехід від статичного планування до динамічного управління здоров'ям користувача. Існуючі підходи до формування рухових пауз в умовах офісної праці здебільшого ігнорують принцип фізіологічної індивідуальності: один і той самий набір рухів може бути недостатнім для підготовленої особи і водночас травмонебезпечним для людини з гострим спазмом або больовим синдромом. Для усунення цього недоліку ядром системи було визначено алгоритм поведінкової адаптації, який формує наступний мікроцикл на основі аналізу попереднього досвіду користувача.

Головним вхідним параметром для роботи алгоритму є кількісна оцінка за шкалою суб'єктивного сприйняття навантаження (Rate of Perceived Exertion — RPE), яка в даній системі розширена до діапазону від 1 до 10 балів. У контексті профілактики гіподинамії ця шкала інтерпретується як мірило дискомфорту, зусиль та м'язової скутості. Алгоритм здійснює обробку отриманого від користувача бала та ініціює адаптацію, керуючи двома взаємопов'язаними, але незалежними параметрами тренувального процесу: рівнем фізичної складності самої вправи (біомеханічною важкістю рухів) та інтенсивністю її виконання (тривалістю статичної фіксації або кількістю динамічних повторень).

Логіка роботи алгоритму базується на системі продукційних правил (If-Then), які розподіляють отримані бали RPE на три основні зони реагування: зону недостатнього стимулу, зону оптимального навантаження та зону перенапруження.

У випадку, коли після завершення рухової паузи користувач виставляє низький бал (наприклад, RPE від 1 до 3), система розцінює це як сигнал про недостатній фізіологічний стимул. Виконана вправа виявилася занадто легкою і не забезпечила належного розтягнення чи активізації цільової м'язової групи. Як

наслідок, алгоритм ініціює запит до бази даних для пошуку вправи тієї ж категорії (наприклад, «Гімнастика для шиї»), але з індексом складності, підвищеним на один рівень (DifficultyLevel + 1). Окрім структурної заміни самої вправи на важчу, алгоритм також виконує коригування інтенсивності: тривалість виконання наступного мікроциклу пропорційно збільшується (наприклад, з 30 до 45 секунд), що дозволяє посилити терапевтичний ефект без ризику травмування.

Протилежний сценарій активується при отриманні високого бала (RPE від 7 до 10). Такі показники сигналізують про серйозне перенапруження, наявність вираженого больового синдрому, гострого м'язового спазму або загальної фізичної втоми користувача. У такій ситуації продовження виконання аналогічних навантажень є фізіологічно недоцільним і небезпечним. Алгоритм діє превентивно: він автоматично знижує цільовий індекс складності (DifficultyLevel - 1), підбираючи з бази даних найпростіший, найбільш щадний варіант рухової активності для даної зони. Паралельно відбувається різке зниження інтенсивності — час виконання вправи або кількість повторень зменшується до мінімально допустимого значення, що гарантує безпечне відновлення та зняття гострої симптоматики.

Якщо ж оцінка RPE знаходиться в межах оптимальної зони (від 4 до 6 балів), алгоритм робить висновок, що поточне навантаження ідеально відповідає фізичному стану користувача. У цьому випадку індекс складності вправи залишається незмінним, а інтенсивність фіксується на поточному рівні, забезпечуючи стабільну підтримувальну терапію гіподинамічних станів.

Загальний життєвий цикл алгоритму виглядає наступним чином: зчитування останнього логу користувача -> аналіз балу RPE -> розрахунок дельти (зміщення) для складності та інтенсивності -> звернення до СУБД для вибірки нової вправи відповідної категорії з оновленим параметром DifficultyLevel -> формування JSON-відповіді для клієнтського React-застосунку -> оновлення таймера та інтерфейсу. Візуалізацію цієї логіки, яка дозволяє відстежити шляхи прийняття рішень системою на кожному етапі, представлено у вигляді блок-схеми.

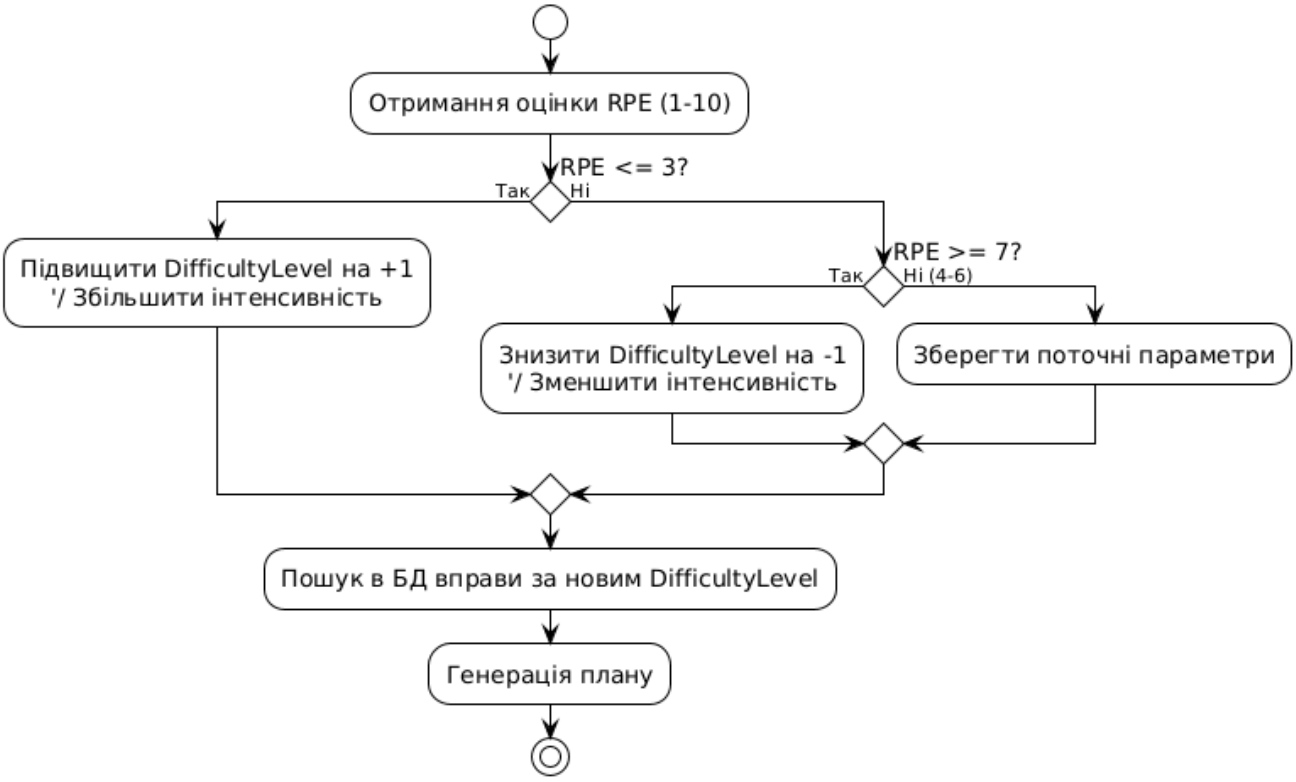


Рис. 2.3 Блок-схема роботи алгоритму поведінкової адаптації мікроциклів

2.4 Проєктування інтерфейсу користувача (UI/UX) та сценаріїв взаємодії

Впровадження будь-якої інформаційної системи, орієнтованої на регулярну взаємодію з користувачем, значною мірою залежить від якості спроєктованого користувацького інтерфейсу (UI) та загального досвіду взаємодії (UX). Для цільової аудиторії розроблюваного застосунку — фахівців IT-сфери та офісних працівників — критично важливо, щоб процес виконання рухових мікроциклів не порушував стан глибокого «робочого потоку» (flow) та не вимагав тривалого відволікання на навігацію всередині програми. З огляду на це, проєктування візуальної частини базується на принципах мінімалізму, когнітивної легкості та забезпечення миттєвого зворотного зв'язку.

Користувач взаємодіє з єдиним динамічним вікном, де зміна станів — від перегляду загальної статистики до запуску активного таймера виконання вправи — відбувається миттєво, без перезавантаження сторінки браузера. Окрему увагу

під час проєктування було приділено ергономіці форм зворотного зв'язку. Оскільки ядром адаптивного алгоритму є оцінка за шкалою RPE, інтерфейс введення цього показника спроєктовано у вигляді набору кнопок з градацією від 1 до 10. Це дозволяє користувачеві за лічені секунди зафіксувати свій фізичний стан після завершення рухової паузи, мінімізуючи психологічний бар'єр для введення даних.

Для детального розуміння процесів, що відбуваються на технічному рівні під час взаємодії користувача з інтерфейсом, розроблено сценарій обробки зворотного зв'язку. Цей сценарій охоплює повний цикл: від моменту підтвердження оцінки RPE на клієнті до обчислення нової адаптивної моделі на сервері та збереження результатів у реляційну базу даних. Для формалізації цього динамічного процесу використано уніфіковану мову моделювання, а саме UML-діаграму послідовності (Sequence Diagram). Вона наочно демонструє синхронні та асинхронні HTTP-виклики, а також порядок передачі управління між основними вузлами системи: React-клієнтом, контролером ASP.NET Core API, сервісом поведінкової адаптації та базою даних MS SQL Server.

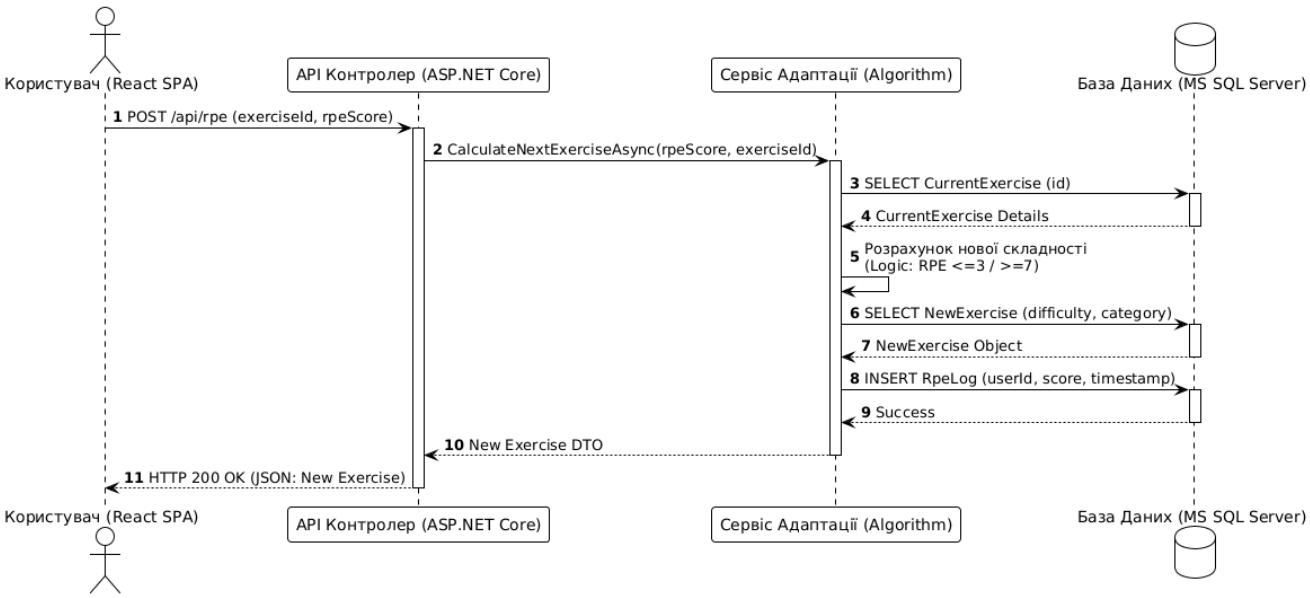


Рис. 2.4 UML-діаграма послідовності обробки оцінки RPE та генерації нового мікроциклу

Як відображено на діаграмі послідовності, ініціатором транзакції є клієнтський застосунок, який після отримання оцінки RPE формує асинхронний запит до серверного API. Серверна частина виконує валідацію вхідних даних та звертається до сховища для отримання поточного індексу складності щойно виконаної вправи. Далі в роботу вступає сервіс поведінкової адаптації, який, згідно із закладеною математичною моделлю продукційних правил, обчислює дельту для наступного навантаження. Після оновлення записів у базі даних (збереження логу виконання та вибірки нової вправи відповідної складності), сервер повертає клієнту оновлений JSON-об'єкт. Цей об'єкт миттєво обробляється React-компонентами та відображається в інтерфейсі, готуючи систему до наступної рухової сесії.

Такий оптимізований цикл обміну даними гарантує, що система залишається високопродуктивною і не створює затримок у робочому процесі користувача. Спроєктована клієнт-серверна архітектура, структура реляційної бази даних та деталізована логіка взаємодії компонентів утворюють цілісний технічний фундамент. Виконання всіх етапів проєктування, описаних у даному розділі, дозволяє обґрунтовано перейти до наступної фази життєвого циклу розробки — безпосередньої програмної реалізації інформаційної системи.

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ АДАПТИВНОГО ВЕБ-ЗАСТОСУНКУ

3.1 Програмна реалізація серверної частини та забезпечення взаємодії з базою даних

Перехід від етапу проєктування до програмної реалізації розпочався зі створення серверного ядра розроблюваної інформаційної системи. Серверна частина (Backend) була реалізована на базі кросплатформного фреймворку ASP.NET Core (на платформі .NET) із використанням мови програмування C#. Архітектура проєкту побудована за принципом REST API, що передбачає створення набору кінцевих точок (ендпоінтів) для обробки HTTP-запитів від клієнтського застосунку. Структура програмного проєкту була розділена на кілька логічних шарів: шар доступу до даних (Models та Data), шар бізнес-логіки (Services) та шар контролерів (Controllers). Для забезпечення слабкого зв'язування між цими компонентами широко використовувався вбудований у .NET механізм ін'єкції залежностей (Dependency Injection), що дозволило реєструвати сервіси та контекст бази даних у головному конфігураційному файлі Program.cs.

Реалізація шару доступу до даних базується на використанні технології об'єктно-реляційного відображення (ORM). Для взаємодії з реляційною системою управління базами даних Microsoft SQL Server було інтегровано бібліотеку Entity Framework Core (пакет Microsoft.EntityFrameworkCore.SqlServer). Застосовано підхід Code-First, згідно з яким структура бази даних генерується автоматично на основі написаних класів C#. Було створено базові класи-сутності: User (користувач), Category (категорія вправи), Exercise (вправа) та RpeLog (журнал виконання). Відповідно до спроєктованої раніше логічної моделі, клас Exercise обов'язково включає цілочисельну властивість DifficultyLevel, яка відповідає за рівень фізичної

складності. Для об'єднання цих сутностей у єдину систему було створено клас ApplicationDbContext, який успадковується від базового класу DbContext і містить набори даних DbSet для кожної таблиці. Після конфігурації рядка підключення (Connection String) було виконано генерацію міграцій та їх застосування до фізичної бази даних, що забезпечило повну відповідність програмних моделей SQL-таблицям.

У даному підрозділі представлено реалізацію взаємодії з реляційною базою даних MS SQL Server. Завдяки використанню технології Entity Framework Core, центральним вузлом доступу до даних виступає клас контексту, який наслідується від базового класу DbContext. Нижче наведено структуру контексту бази даних та модель сутності вправи, де ключовою для роботи алгоритму адаптації є властивість DifficultyLevel.

```
using Microsoft.EntityFrameworkCore;
using SmartWorkoutPlanner.API.Models;

namespace SmartWorkoutPlanner.API.Data
{
    /// <summary>
    /// Контекст бази даних для взаємодії з MS SQL Server через
Entity Framework Core
    /// </summary>
    public class ApplicationDbContext : DbContext
    {
        public
ApplicationDbContext(DbContextOptions<ApplicationDbContext>
options)
            : base(options)
        {
        }

        // Реєстрація сутності Exercise у контексті бази даних
        public DbSet<Exercise> Exercises { get; set; }

        // Інші сутності системи
        public DbSet<Workout> Workouts { get; set; }
        public DbSet<WorkoutDetail> WorkoutDetails { get; set;
}
        public DbSet<ExerciseLog> ExerciseLogs { get; set; }
    }
}
```

```
 namespace SmartWorkoutPlanner.API.Models
{
    /// <summary>
    /// Модель даних профілактичної вправи
    /// </summary>
    public class Exercise
    {
        public int Id { get; set; }

        public string Name { get; set; } = string.Empty;

        public string? MuscleGroup { get; set; }

        public string? Description { get; set; }

        /// <summary>
        /// Рівень складності (1 - Базовий, 2 - Середній, 3 - Просунутий).
        ///   Використовується алгоритмом для стартового призначення мікроциклу.
        /// </summary>
        public int DifficultyLevel { get; set; }
    }
}
```

Зважаючи на специфіку системи як адаптивного прототипу, захист інформації реалізовано через багаторівневу ізоляцію бізнес-логіки від зовнішнього середовища. Основним інструментом гарантування безпеки на рівні обміну даними стало впровадження об'єктів передачі даних (Data Transfer Objects — DTO). Такий підхід дозволив повністю виключити можливість прямого доступу клієнтського застосунку до моделей бази даних, що нівелює ризики атак типу Mass Assignment.

Кожен вхідний запит проходить сувору перевірку на відповідність типам даних, а використання вбудованих механізмів валідації ASP.NET Core гарантує, що до обчислювальних сервісів потрапляють лише коректно сформовані параметри.

Додатковим рівнем захисту виступає використання ORM-фреймворку Entity Framework Core, який автоматично реалізує механізм параметризованих запитів. Це забезпечує надійну ізоляцію бази даних MS SQL Server від

потенційних SQL-ін'єкцій, оскільки будь-яке введення користувача інтерпретується виключно як дані, а не як частина виконуваного коду.

Обробивши питання архітектурної безпеки та ізоляції шарів, було реалізовано шар контролерів для організації мережевої взаємодії через REST API. Центральними вузлами маршрутизації стали контролери WorkoutsController та AnalyticsController.

Перший відповідає за інтелектуальну генерацію мікроциклів, тоді як другий забезпечує формування персоналізованих медичних рекомендацій та візуалізацію профілю дискомфорту. Для забезпечення високої пропускної здатності та відгуку сервера всі методи контролерів були реалізовані з використанням парадигми асинхронного програмування за допомогою ключових слів async та await. Це дозволяє серверу обробляти множинні запити паралельно, не блокуючи основні потоки виконання під час звернення до дискової підсистеми або бази даних.

Коли клієнтський застосунок надсилає HTTP-запит на генерацію або завершення тренування, контролер асинхронно передає дані до відповідних сервісів. Використовуючи технологію інтегрованих запитів LINQ (Language Integrated Query), система звертається до контексту бази даних ApplicationDbContext для виконання складних операцій вибірки.

Алгоритм враховує динамічний відкат для різних груп м'язів та аналізує історію оцінок RPE для розрахунку тривалості наступних вправ. В процесі формування відповіді об'єкти вправ, що містять назву, опис техніки виконання та базовий рівень складності (DifficultyLevel), серіалізуються у формат JSON і повертаються клієнту з відповідним HTTP-статусом.

Така реалізація забезпечує повну ізоляцію складної обчислювальної логіки від візуального інтерфейсу, що робить систему стабільною, масштабованою та готовою до експлуатації в умовах реального робочого процесу.

3.2 Програмна реалізація клієнтської частини та забезпечення інтерактивної взаємодії

Для реалізації фронтенду (Frontend) вебзастосунку було використано сучасну JavaScript-бібліотеку React. Вибір цієї технології зумовлений її компонентною архітектурою, яка дозволяє розбивати користувацький інтерфейс на незалежні, ізольовані модулі (компоненти), придатні для повторного використання. Розробка велася з використанням синтаксису JSX (JavaScript XML), що дозволило декларативно описувати структуру інтерфейсу безпосередньо всередині логіки компонентів.

Оскільки система спроєктована як односторінковий вебзастосунок (Single Page Application, SPA), маршрутизація та перемикання між основними екранами (панель управління, екран виконання вправи, екран оцінювання) реалізовані на стороні клієнта за допомогою бібліотеки react-router-dom. Це дозволило уникнути класичного перезавантаження сторінок при кожній дії користувача, забезпечивши миттєвий відгук інтерфейсу. Управління внутрішнім станом програми (State Management), зокрема збереженням ідентифікатора поточної вправи, часу таймера та обраного бала дискомфорту, реалізовано за допомогою вбудованих хуків React, таких як useState та useReducer.

Основним інтерактивним вузлом клієнтської частини є екран виконання рухового мікроциклу. Він складається з візуального таймера зворотного відліку та компонента-інструкції, який відображає текстовий опис або графічне зображення вправи. Для реалізації життєвого циклу таймера застосовано хук useEffect, який асинхронно відраховує час, заданий сервером для поточної інтенсивності. Після завершення таймера система автоматично демонтує компонент вправи та рендерить на екрані компонент зворотного зв'язку — форму оцінювання за шкалою RPE.

Інтерфейс форми спроєктовано з урахуванням когнітивної ергономіки: користувачеві пропонується шкала від 1 до 10, де крайні значення супроводжуються текстовими підказками (наприклад, «1 — Зовсім не відчуваю

напруги», «10 — Сильний біль або дискомфорт»). Коли користувач обирає відповідний бал і натискає кнопку підтвердження, клієнтський застосунок формує HTTP POST запит до розробленого раніше ASP.NET Core API. Мережева взаємодія реалізована за допомогою сучасного Fetch API (або бібліотеки Axios), що дозволяє асинхронно передати ідентифікатор вправи та оцінку RPE на сервер.

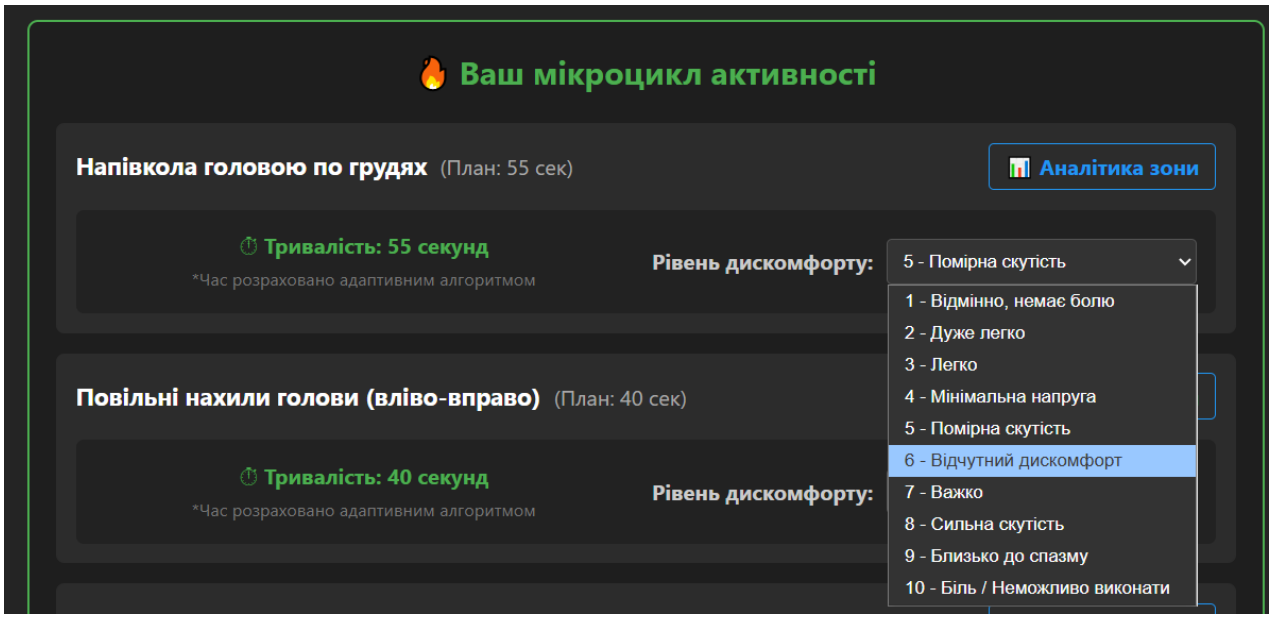


Рис. 3.1 Інтерфейс користувача для збору поведінкової аналітики за шкалою RPE

Після відправки запиту React-компонент переходить у стан очікування (Loading state), візуалізуючи індикатор завантаження, поки серверна частина обробляє дані та здійснює алгоритмічну заміну вправи. Отримавши у відповідь JSON-об'єкт із новою адаптованою вправою (з відкоригованою складністю або інтенсивністю), клієнтський застосунок оновлює глобальний стан. Завдяки механізму віртуального DOM (Virtual DOM) бібліотеки React, інтерфейс перемальовує лише ті частини сторінки, які змінилися, готуючи робочий простір до наступного мікроциклу.

Інтерфейс побудовано з використанням CSS-модулів (або бібліотек компонентів на кшталт Material-UI), що забезпечило його коректне відображення

як на широкоформатних моніторах, так і на екранах мобільних пристроїв. Кольорова палітра розроблена в нейтральних тонах, аби не викликати зорової втоми під час робочого дня, що цілком відповідає медико-профілактичній меті програмного продукту.

Таким чином, розроблена клієнтська частина повністю задовольняє вимоги щодо швидкодії, кросплатформності та зручності використання. Синхронізована робота інтерактивних React-компонентів із серверним алгоритмом адаптації забезпечує безперебійний та науково обґрунтований процес профілактики гіподинамії.

3.3 Реалізація взаємодії клієнтської та серверної частин через REST API

Як основний архітектурний стиль для мережевої взаємодії було обрано REST (Representational State Transfer). Цей підхід забезпечує інтерфейс обміну даними через стандартні HTTP-методи, що робить систему легко масштабованою та незалежною від внутрішньої реалізації окремих вузлів. Форматом передачі даних (Payload) було визначено легкочитуваний текстовий формат JSON (JavaScript Object Notation), який нативно підтримується як платформою .NET, так і середовищем виконання JavaScript у браузері.

Для забезпечення інформаційної безпеки та оптимізації мережевого трафіку під час обміну даними було імплементовано патерн проєктування DTO (Data Transfer Object — об'єкт передачі даних). Пряма передача сутностей бази даних (наприклад, моделі Exercise або User) клієнтському застосунку є антипатерном, оскільки це може призвести до витоку конфіденційної інформації про структуру бази даних та створити вразливість до атак типу Mass Assignment (надмірне призначення). Замість цього на стороні сервера ASP.NET Core було створено окремі класи DTO для кожного запиту та відповіді. Наприклад, коли клієнт відправляє результати виконання мікроциклу, він надсилає не всю інформацію про вправу, а лише спеціально сформований об'єкт

RpeEvaluationRequestDto, який містить виключно ідентифікатор виконаної активності та виставлений бал за шкалою RPE (від 1 до 10).

Для забезпечення безпеки інформації та оптимізації мережевого трафіку під час обміну даними між клієнтським застосунком і серверним REST API було імплементовано патерн об'єктів передачі даних (Data Transfer Object, DTO). Пряма передача сутностей бази даних є потенційно вразливою до атак надлишкового призначення, тому для прийому результатів виконання мікроциклу розроблено спеціалізований клас. Нижче наведено реалізацію DTO для фіксації зворотного зв'язку, де за допомогою вбудованих атрибутів валідації (Data Annotations) забезпечується суворий контроль вхідних параметрів на рівні прив'язки моделі (Model Binding), зокрема автоматичне відхилення оцінок RPE, що виходять за межі діапазону від 1 до 10.

```
using System.ComponentModel.DataAnnotations;

namespace SmartWorkoutPlanner.API.Models.DTOs
{
    /// <summary>
    /// Об'єкт передачі даних (DTO) для безпечного отримання результатів мікроциклу.
    /// Ізолює моделі бази даних від зовнішніх запитів.
    /// </summary>
    public class RpeEvaluationRequestDto
    {
        /// <summary>
        /// Унікальний ідентифікатор виконаної вправи
        /// </summary>
        [Required(ErrorMessage = "Ідентифікатор вправи є обов'язковим параметром.")]
        public int ExerciseId { get; set; }

        /// <summary>
        /// Оцінка суб'єктивного дискомфорту.
        /// Атрибут Range гарантує відхилення запитів із некоректними значеннями на рівні фреймворку.
        /// </summary>
        [Required(ErrorMessage = "Оцінка RPE є обов'язковою.")]
        [Range(1, 10, ErrorMessage = "Показник RPE повинен бути строго в діапазоні від 1 до 10.")]
        public int RpeScore { get; set; }

        /// <summary>
        /// Опціональний коментар користувача щодо самопочуття
```

```
        /// </summary>
        [StringLength(500, ErrorMessage = "Довжина коментаря не
може перевищувати 500 символів.")]
        public string? UserComment { get; set; }
    }
}
```

Впровадження DTO також дозволило чітко розмежувати контракти API та бізнес-моделі. Для автоматизації процесу перетворення (мапінгу) даних між моделями бази даних та DTO-об'єктами доцільно використовувати бібліотеки об'єктного відображення, такі як AutoMapper, або ж виконувати ручне перетворення за допомогою інтегрованих запитів LINQ у сервісному шарі застосунку.

Важливою технічною задачею під час налаштування мережевої взаємодії стало вирішення проблеми спільного використання ресурсів з різних джерел (Cross-Origin Resource Sharing — CORS). Оскільки клієнтський React-застосунок під час розробки (та потенційно на етапі експлуатації) запускається на одному мережевому порту або домені, а серверний API — на іншому, сучасні веббраузери за замовчуванням блокують такі асинхронні запити з міркувань безпеки (політика Same-Origin Policy). Для подолання цього обмеження у конфігураційному файлі Program.cs серверної частини було налаштовано політику CORS (використання проміжного ПЗ UseCors). Це дозволило вказати довірені URL-адреси клієнтського застосунку, з яких дозволено приймати HTTP-запити із заголовками Content-Type: application/json.

Для стандартизації розробленого програмного інтерфейсу та спрощення процесу тестування взаємодії між клієнтом і сервером, до проєкту було інтегровано специфікацію OpenAPI. Практична реалізація цього стандарту відбулася за допомогою підключення NuGet-пакетів Swashbuckle.AspNetCore (Swagger). Завдяки цій технології серверна частина автоматично аналізує всі створені контролери (наприклад, RpeLogsController, ExercisesController) та генерує інтерактивну документацію у вигляді зручного графічного вебінтерфейсу. Інтерфейс Swagger UI дозволяє розробнику переглядати доступні ендпоінти, формати очікуваних JSON-запитів, типи відповідей сервера та

виконувати тестові запити безпосередньо з браузера, без використання стороннього інструментарію на кшталт Postman.

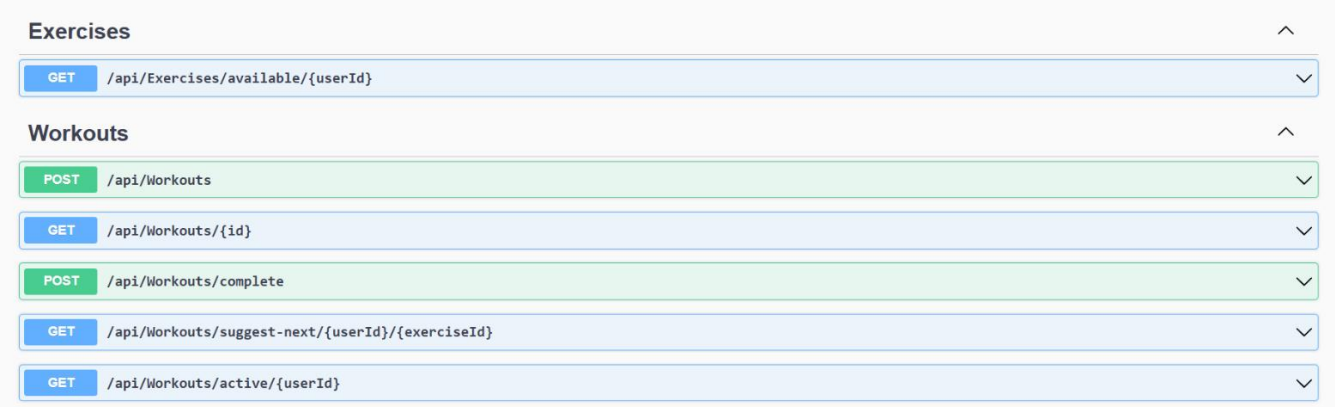


Рис. 3.2 Інтерактивна документація розробленого REST API на базі Swagger UI

Використання паттерну DTO, правильне налаштування політик безпеки CORS та автодокументування через Swagger забезпечили створення гнучкого, прозорого та захищеного каналу комунікації. Злагоджена робота мережевого рівня є критичною передумовою для успішної реалізації головного обчислювального завдання системи — програмного виконання алгоритму поведінкової адаптації, який керує логікою підбору мікроциклів на стороні сервера.

3.4 Програмна реалізація алгоритму поведінкової адаптації мікроциклів

Згідно з обраною архітектурою, обчислювальна логіка була повністю ізольована від мережевих контролерів та винесена у спеціалізований сервісний шар застосунку. Для цього мовою C# було створено клас AdaptationService, який імплементує інтерфейс IAdaptationService. Такий підхід забезпечує можливість

легкого юніт-тестування логіки адаптації шляхом створення мок-об'єктів (Mock objects) для бази даних.

Процес програмної обробки розпочинається з моменту отримання сервісом об'єкта DTO, який містить унікальний ідентифікатор щойно виконаної вправи (Guid ExerciseId) та значення зворотного зв'язку за шкалою RPE (int RpeScore у діапазоні від 1 до 10). За допомогою методів Entity Framework Core (зокрема асинхронного методу FirstOrDefaultAsync) система звертається до контексту бази даних для отримання повної інформації про поточну вправу. Ключовими властивостями є ідентифікатор цільової м'язової категорії (CategoryId), поточний індекс фізичної складності (DifficultyLevel) та базова тривалість виконання у секундах (DurationSeconds).

Основна математична логіка алгоритму реалізована за допомогою умовних конструкцій if-else, які розділяють отриманий бал RPE на три операційні зони та обчислюють зміщення (дельту) для цільової складності та інтенсивності. Програмний код перевіряє умову: якщо RpeScore <= 3 (зона недостатнього стимулу), цільовий індекс складності інкрементується (targetDifficulty = currentDifficulty + 1). Водночас передбачено механізм запобігання виходу за межі існуючих рівнів складності. Якщо розрахована складність перевищує максимальну в системі, алгоритм застосовує вторинний вектор адаптації — збільшення інтенсивності, додаючи до базового часу виконання фіксований крок (наприклад, targetDuration += 15).

У протилежному випадку, якщо програмний код фіксує значення RpeScore >= 7 (зона перенапруження), ініціюється процес деескалації навантаження. Цільовий індекс складності декрементується (targetDifficulty = currentDifficulty - 1). Як і в попередньому блоці умов, алгоритм містить захисний механізм: якщо поточна вправа вже має мінімальний базовий рівень складності (рівень 1), програма зменшує інтенсивність, віднімаючи час від цільової тривалості (targetDuration -= 15), щоб забезпечити користувачеві щадний режим відновлення. Значення RPE від 4 до 6 залишають цільові змінні без змін.

Головне функціональне ядро інформаційної системи — алгоритм поведінкової адаптації — інкапсульовано у спеціалізованому сервісному шарі застосунку. Процес обчислення розпочинається з аналізу отриманого бала RPE та поточних характеристик щойно виконаної активності. Нижче наведено фрагмент програмної реалізації методу розрахунку параметрів наступного мікроциклу. За допомогою умовних конструкцій обчислювальна логіка розділяє стан користувача на три операційні зони: при недостатньому фізичному стимулі (RPE менше або дорівнює 3) система інкрементує цільову складність та збільшує загальну тривалість вправи; у зоні перенапруження (RPE 7 і вище) застосовується зворотний вектор адаптації для зниження навантаження. Згенеровані цільові змінні в подальшому використовуються для запиту до бази даних та пошуку найбільш релевантної вправи.

Після розрахунку нових параметрів навантаження, програма формує динамічний запит до бази даних за допомогою технології LINQ (Language Integrated Query). Метод Where фільтрує таблицю вправ, залишаючи лише ті об'єкти, що збігаються за CategoryId та мають розрахований targetDifficulty. Для забезпечення різноманітності рухових пауз та уникнення монотонності, до LINQ-запиту додано сортування у випадковому порядку (OrderBy(x => Guid.NewGuid())), після чого метод FirstOrDefaultAsync повертає нову адаптовану вправу.

Знайдена вправа пакується у вихідний об'єкт DTO разом із розрахованою новою тривалістю виконання (targetDuration) і повертається на рівень REST-контролера для відправки клієнтському React-застосунку. У разі, якщо в базі даних відсутня вправа з точно заданим рівнем складності, у коді передбачено механізм відмовостійкості (Fallback), який автоматично повертає найближчий за складністю аналог у межах цільової категорії.

Розроблений програмний модуль успішно закриває вимоги щодо інтелектуального планування та завершує цикл створення бізнес-логіки системи. Об'єднання сучасних фреймворків, криптографічних засобів та поведінкових алгоритмів дозволило створити цілісний функціональний продукт.

4 ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ІНФОРМАЦІЇ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Комплексне забезпечення інформаційної безпеки вебзастосунку

Розробка сучасної інформаційної системи, орієнтованої на збір, обробку та зберігання персональної поведінкової аналітики користувачів, вимагає імплементації багаторівневої архітектури інформаційної безпеки. Оскільки розроблюваний вебзастосунок оперує даними, що побічно характеризують фізичний стан людини (через оцінки за шкалою суб'єктивного сприйняття навантаження RPE), забезпечення конфіденційності, цілісності та доступності цієї інформації є критичним нефункціональним завданням. Архітектура безпеки системи будувалася з урахуванням міжнародних рекомендацій OWASP (Open Web Application Security Project) та охоплює захист на рівнях бази даних, серверної обробки та клієнтського інтерфейсу.

У розробленій системі основний акцент архітектурної безпеки зміщено на ізоляцію бізнес-логіки та захист каналів обміну даними. Одним з фундаментальних векторів захисту є протидія маніпуляціям із даними на рівні API. Для цього замість прямого доступу до моделей бази даних використовується патерн об’єктів передачі даних (Data Transfer Objects — DTO). Впровадження DTO дозволяє приховати внутрішню структуру реляційних таблиць від клієнтського застосунку.

Завдяки строгій типізації мови C# та вбудованим механізмам валідації ASP.NET Core, серверна частина автоматично відхиляє будь-які некоректні запити, наприклад, спроби передати текстові шкідливі дані замість очікуваних числових значень оцінки RPE, запобігаючи атакам типу Mass Assignment (надлишкове призначення параметрів).

Другим фундаментальним вектором захисту є протидія ін'єкціям, зокрема атакам типу SQL Injection (SQLi), які стабільно очолюють рейтинги

вебвразливостей. Суть цієї атаки полягає у впровадженні шкідливого SQL-коду через поля введення даних з метою маніпуляції базою даних. У розробленій системі цей ризик нівелюється на архітектурному рівні завдяки використанню об'єктно-реляційного картографа (ORM) Entity Framework Core.

Замість динамічної конкатенації SQL-рядків, фреймворк автоматично застосовує механізм параметризованих запитів (Parameterized queries). Будь-які дані, що надходять від клієнта, обробляються базою даних MS SQL Server виключно як безпечні параметри (літерали), а не як виконуваний код.

Основною загрозою для клієнтських інтерфейсів є атаки типу міжсайтового скриптингу (Cross-Site Scripting — XSS). Вони дозволяють зловмисникам виконувати шкідливі JavaScript-скрипти у браузері легітимного користувача. Вибір бібліотеки React для розробки інтерфейсу автоматично забезпечив потужний рівень захисту від цієї загрози.

Під час рендерингу компонентів механізми React (синтаксис JSX) за замовчуванням здійснюють екранування (Escaping) будь-яких строкових змінних, що вбудовуються в DOM-дерево. Навіть якщо шкідливий код якимось чином потрапить до бази даних, браузер відобразить його як звичайний текст, не виконуючи як скрипт.

Окремим інноваційним рівнем безпеки системи є експертний аналіз аномалій фізичного стану користувача (Medical Safety Logic). На рівні контролерів бізнес-логіки імплементовано систему розпізнавання нетипового болю. Якщо система фіксує критично високий рівень дискомфорту (RPE 8 і вище) під час виконання найпростіших базових вправ (наприклад, гімнастика для очей або розминка кистей), алгоритм ідентифікує це не як звичайну втому, а як потенційну медичну проблему (тунельний синдром, порушення очного тиску тощо). У такому разі застосунок блокує подальше збільшення навантаження та генерує цільове медичне попередження, захищаючи користувача від можливого травматизму чи погіршення стану здоров'я.

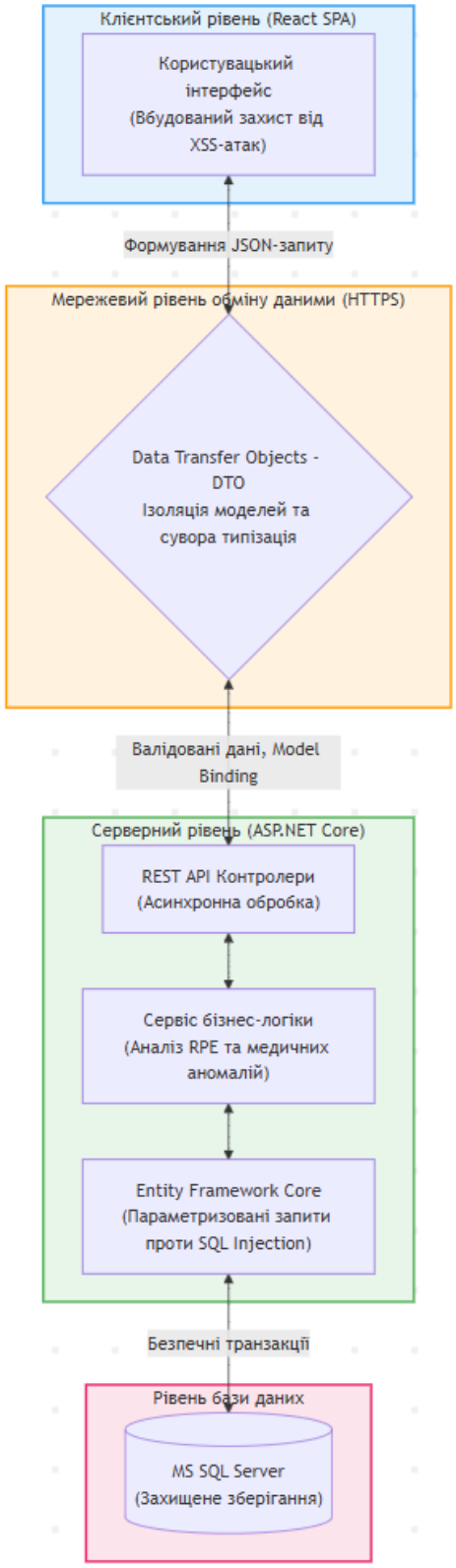


Рис. 4.1 Архітектура безпечної взаємодії між клієнтським SPA та REST API через DTO

4.2 Тестування бізнес-логіки та інтеграційної взаємодії

Завершальним, але не менш важливим етапом життєвого циклу розробки програмного забезпечення є комплексне тестування створеної інформаційної системи. Метою цього процесу є верифікація (підтвердження того, що система розроблена правильно) та валідація (підтвердження того, що система відповідає початковим вимогам користувача).

Оскільки розроблений вебзастосунок оперує інтелектуальним алгоритмом поведінкової адаптації, стратегія тестування була зосереджена на перевірці коректності роботи механізмів «Динамічного відкату», інтеграційному тестуванні ендпоінтів REST API та функціональній перевірці користувацького інтерфейсу.

Функціональне тестування бізнес-логіки було спрямоване на перевірку ядра системи — сервісу адаптації мікроциклів та генерації вправ. У ході тестування перевірялася реакція алгоритму на різні вхідні параметри (значення шкали RPE) та дотримання часових обмежень для різних груп м'язів.

Перший сценарій тестування симулював виконання мікроциклу з низьким рівнем дискомфорту (наприклад, RPE = 3). Система, як і очікувалося, коректно збільшувала цільовий час виконання наступної паузи для забезпечення прогресії навантаження.

Другий сценарій перевіряв стан перенапруження (RPE > 8). Результати підтвердили, що контролер миттєво зменшує тривалість майбутніх вправ та генерує відповідні медичні рекомендації. Також успішно пройшов перевірку алгоритм ротації: тестування підтвердило, що система блокує повторну генерацію вправ для шийного відділу на 3 години, тоді як гімнастика для зорового апарату стає доступною вже через 1 годину, що математично доводить правильність програмної реалізації продукційних правил.

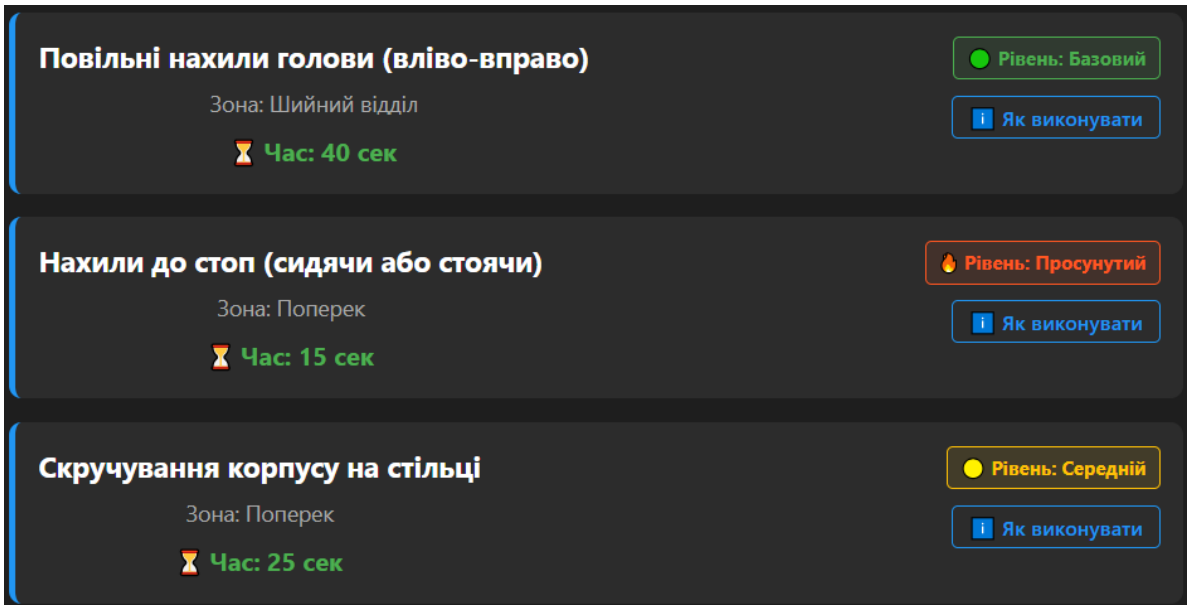


Рис. 4.2 Результати тестування алгоритму генерації з присвоєнням рівнів складності: «Базовий», «Середній» та «Просунутий»

Наступним кроком стало інтеграційне тестування розробленого REST API. Основною задачею цього етапу була перевірка коректності обробки HTTP-запитів контролерами ASP.NET Core, правильності серіалізації та десеріалізації JSON-даних (DTO-об'єктів) та безперебійного зв'язку з базою даних. Тестування проводилося за допомогою вбудованого інтерфейсу OpenAPI (Swagger UI), що є стандартом індустрії для документування та верифікації API.

Було імітовано процес завершення тренування через метод POST. Серверна частина успішно прийняла масив логів з оцінками RPE, обробила їх та зберегла у таблиці бази даних, повернувши клієнту HTTP-статус 200 (OK). Додатково перевірялися GET-запити для отримання аналітичного профілю користувача, які продемонстрували коректну агрегацію середніх значень дискомфорту по кожній групі м'язів.

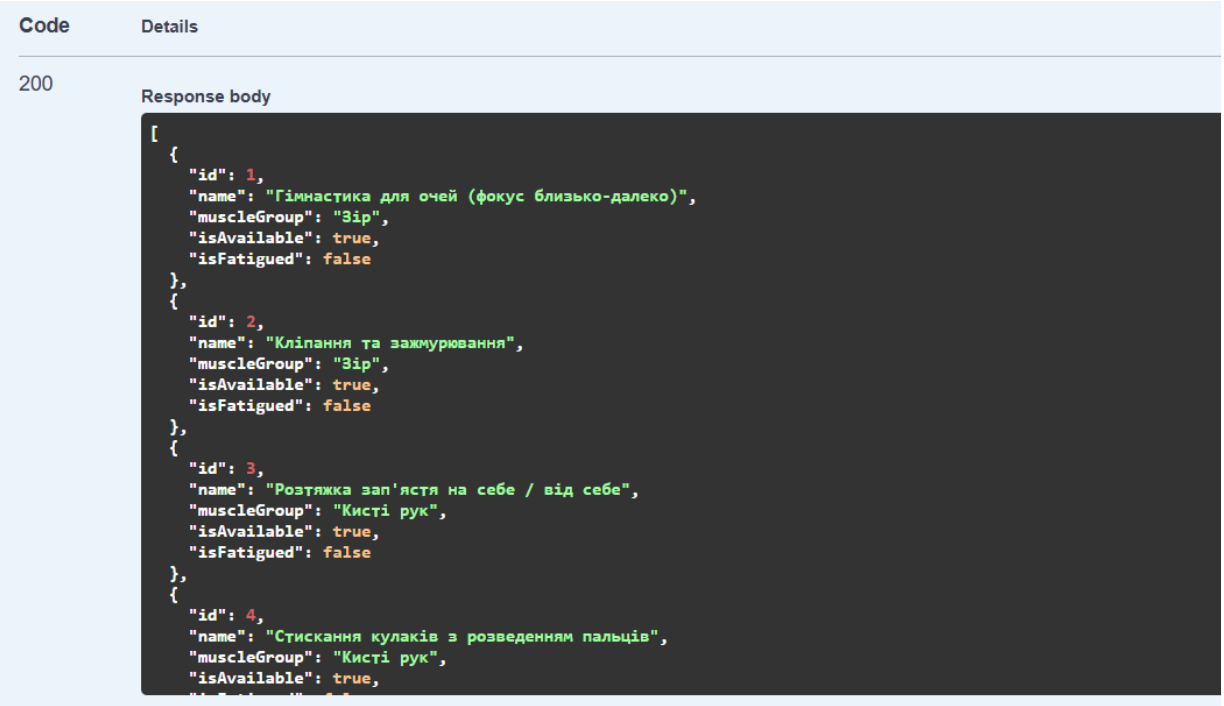


Рис. 4.3 Інтерактивне тестування REST API ендпоінтів за допомогою вбудованого інструменту Swagger UI)

Фінальна стадія полягала у функціональному тестуванні клієнтського React-застосунку методом чорного ящика (Black-box testing). Робота перевірялася безпосередньо у вікні веббраузера шляхом імітації дій реального користувача. Перевірялася плавність відмальовування компонентів інтерфейсу, коректна робота хуків стану та оновлення інтерактивних графіків Recharts.

Окремо було протестовано механізм розгортання інформаційних блоків з технікою виконання вправ та відображення адаптивних бейджів складності. Натискання на кнопку відправки форми оцінювання миттєво ініціювало відображення індикатора завантаження, після чого радарний графік профілю дискомфорту плавно оновлювався з урахуванням нових даних. Успішне проходження всіх етапів тестування підтверджує, що розроблена система є повністю функціональною, відмовостійкою та готовою до експлуатації.

4.3 Оцінка експлуатаційної ефективності системи та керівництво користувача

Після успішної програмної реалізації, налаштування механізмів безпеки та проходження всіх етапів тестування, фінальним кроком є оцінка експлуатаційних характеристик розробленої інформаційної системи та формалізація процесу взаємодії з нею кінцевого користувача. Оцінка ефективності програмного продукту проводилася за двома основними напрямами: швидкодія обчислювальної логіки (серверна частина) та чуйність інтерфейсу (клієнтська частина).

Завдяки використанню асинхронного програмування у фреймворку ASP.NET Core, серверна частина здатна обробляти паралельні запити без блокування основних потоків виконання. Алгоритм поведінкової адаптації, який здійснює розрахунок таймінгів відновлення та пошук нових вправ у базі даних MS SQL Server, продемонстрував високу ефективність.

Завдяки використанню оптимізованих запитів Entity Framework Core, час від прийняття оцінки RPE до генерації та повернення нового плану мікроциклу складає лічені мілісекунди. Зі свого боку, клієнтська архітектура Single Page Application на базі React повністю усуває необхідність перезавантаження сторінок браузера.

Оновлення візуальних компонентів відбувається миттєво через механізм Virtual DOM. У синергії ці технологічні рішення забезпечують безшовний користувацький досвід: користувач не відчуває жодних технічних затримок під час робочого дня, що є критично важливим для збереження його концентрації.

Робочий процес кінцевого користувача (User Flow) спроєктовано з акцентом на мінімізацію когнітивного навантаження та інтуїтивну зрозумілість. Життєвий цикл взаємодії з системою починається з етапу ознайомлення з правилами експлуатації.

При першому запуску застосунку користувача зустрічає інтерактивне модальне вікно з медичним дисклеймером, яке наголошує на профілактичному

характері системи та необхідності консультації з лікарем у разі наявності хронічних захворювань.

Після допуску до системи користувач потрапляє на головну панель управління (Dashboard). Центральним елементом цього екрану є радарний графік "Профіль дискомфорту", який автоматично будується на основі минулої історії тренувань і візуально демонструє рівень скутості у п'яти ключових зонах (шийний відділ, поперек, зір, кисті рук, нижні кінцівки). На цьому ж етапі користувач обирає поточну локацію (вільний простір або робоче місце) та ініціює алгоритм генерації рухової паузи натисканням відповідної кнопки.

Наступний крок передбачає виконання згенерованого мікроциклу. Система формує перелік вправ, враховуючи необхідний час на відновлення кожної групи м'язів. Для кожної вправи відображається цільовий час виконання у секундах та інтерактивний бейдж рівня складності.

У разі необхідності користувач може розгорнути інформаційну панель, що містить детальний текстовий опис техніки виконання та цільові медичні застереження для конкретної групи м'язів, що мінімізує ризик неправильного виконання рухів.

Ключовий етап процесу настає після виконання рухової паузи — збір зворотного зв'язку. Користувачеві пропонується оцінити рівень відчутного дискомфорту, зусиль або болю за шкалою RPE від 1 до 10. Завершення цього процесу відправляє дані на сервер для подальшого аналізу.

Фінальним кроком є миттєва реакція системи: алгоритм аналізує отриманий бал і автоматично підбирає параметри для наступних сесій. Користувач перенаправляється на оновлену головну сторінку, де візуальні графіки вже відображають зниження рівня скутості пропрацьованих зон, що стимулює подальше використання системи.

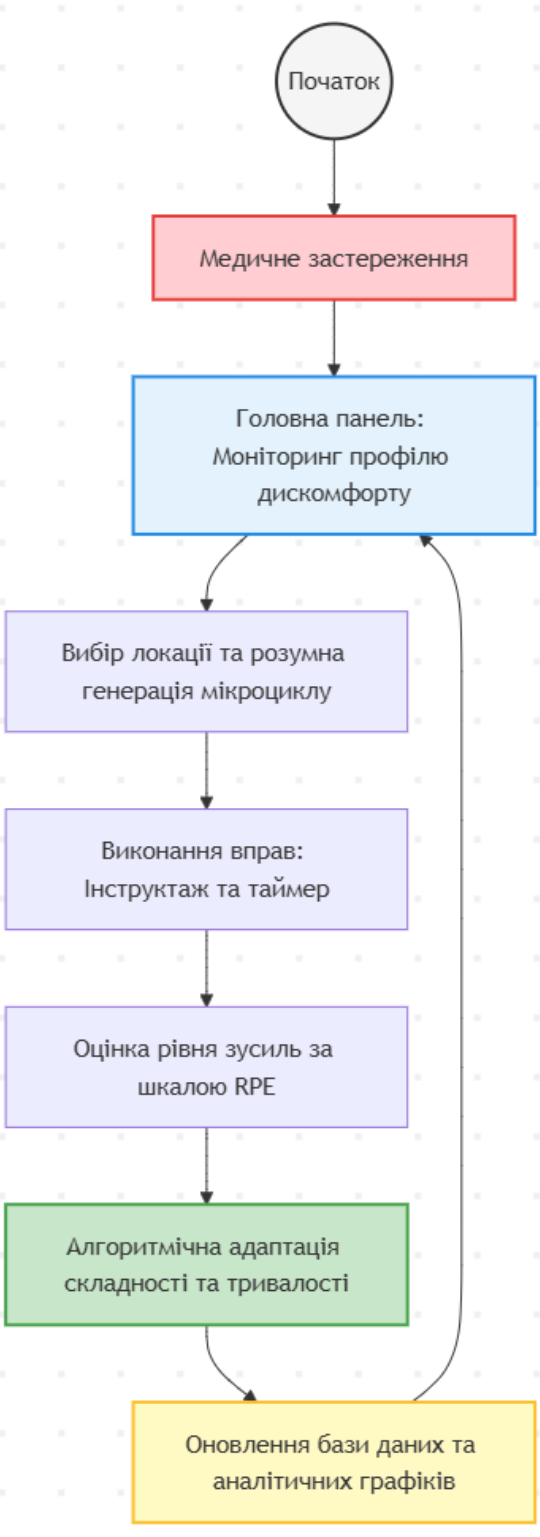


Рис. 4.4 Алгоритм користувацького сценарію: від медичного допуску до адаптивної аналітики

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розв’язано актуальне науково-прикладне завдання — розроблено інформаційну систему профілактики гіподинамічних станів для осіб із малорухливим характером праці. Створений адаптивний вебзастосунок дозволяє автоматизувати процес планування рухових мікроциклів, базуючись на індивідуальному фізіологічному зворотному зв’язку користувача.

Під час виконання роботи було досягнуто наступних результатів:

1. Проведено аналіз предметної області та досліджено проблему професійної гіподинамії серед фахівців ІТ-сфери. Огляд існуючих програмних рішень (таймерів продуктивності та фітнес-застосунків) виявив їхні суттєві недоліки: відсутність персоналізації, ігнорування індивідуального больового синдрому та статичність пропонованих вправ. Це обґрунтувало доцільність створення нової системи, здатної гнучко налаштовувати контент мікроциклів на основі 10-бальної шкали суб'єктивного сприйняття навантаження (RPE).

2. Сформульовано функціональні та нефункціональні вимоги до програмного забезпечення. Спроєктовано багаторівневу клієнт-серверну архітектуру (SPA), яка забезпечує високу модульність, кросплатформність та швидкість відгуку інтерфейсу.

3. Розроблено логічну та фізичну моделі реляційної бази даних. Ключовим архітектурним рішенням стало введення до сутності вправ незмінного атрибута фізичної складності (Difficulty Level) та створення транзакційного журналу для збереження оцінок RPE, що створило необхідний фундамент для роботи обчислювальних алгоритмів.

4. Спроєктовано та програмно реалізовано математичну модель алгоритму поведінкової адаптації. Алгоритм забезпечує багатовекторне коригування мікроциклів: при низьких балах RPE (1-3) система автоматично підвищує складність або збільшує інтенсивність виконання вправи, а при

високих балах (7-10) — знижує індекс складності та час виконання для забезпечення безпечного відновлення.

5. Здійснено програмну реалізацію серверної частини за допомогою фреймворку ASP.NET Core та клієнтської частини на базі бібліотеки React. Взаємодію між рівнями організовано через захищений REST API з використанням патерну DTO (Data Transfer Object) та налаштуванням політик CORS.

6. Впроваджено комплекс механізмів інформаційної безпеки: захист бази даних від SQL-ін'єкцій реалізовано засобами Entity Framework Core, протидію XSS-атакам забезпечено на рівні компонентів React, а безпека API гарантована використанням патерну DTO. Проведено функціональне та інтеграційне тестування системи за допомогою інструментарію Swagger UI. Результати тестування повністю підтвердили коректність роботи алгоритму адаптації.

7. Проведено всебічне модульне, інтеграційне та функціональне тестування системи. Результати тестування повністю підтвердили коректність роботи алгоритму адаптації, надійність серверних контролерів та високу експлуатаційну ефективність користувацького інтерфейсу.

Практичне значення отриманих результатів полягає у тому, що розроблена інформаційна система є готовим до використання програмним продуктом. Вона може бути впроваджена в корпоративні екосистеми IT-компаній або використовуватися індивідуально для організації здорового робочого процесу, зниження ризиків професійних захворювань опорно-рухового апарату та підтримки високого рівня когнітивної продуктивності спеціалістів.

Напрямками подальшого розвитку системи є впровадження методів машинного навчання для більш глибокої предиктивної аналітики стану користувача, а також розробка нативних мобільних застосунків-компаньйонів.

Робота пройшла апробацію. За її результатом було опубліковано наступні тези доповіді:

1) Куцик В.М., Бажан Ю.П. Визначення вимог до веб-застосунку для адаптивного планування тренувань. Матеріали II Всеукраїнської науково-технічної конференції “Виклики та рішення в програмній інженерії”, 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. С.163-165.

2) Куцик В.М., Бажан Ю.П. Забезпечення безпеки інформації у web-застосунку для адаптивного планування тренувань. Матеріали VII Всеукраїнської науково-технічної конференції “Застосування програмного забезпечення в інформаційно-комунікаційних технологіях”, 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. С.35-37.

ПЕРЕЛІК ПОСИЛАНЬ

1. Організація праці та ергономіка : навч. посіб. / О. М. Коваленко та ін. Київ : Центр учбової літератури, 2022. 256 с.

2. The "Active Couch Potato" Phenomenon: Why Regular Exercise Is Not Enough / J. Smith et al. *Journal of Physical Activity and Health*. 2023. Vol. 15, No. 2. P. 112–119.

3. Borg G. Borg's Perceived Exertion and Pain Scales. Champaign : Human Kinetics, 1998. 104 p.

4. Stretchly – The break time reminder app. URL: https://hovancik.net/stretchly/ (date of access: 15.04.2026).

5. Wakeout: Exercise & Movement for Work. URL: https://wakeout.app/ (date of access: 16.04.2026).

6. Freeletics: AI Fitness Coach. URL: https://www.freeletics.com/ (date of access: 16.04.2026).

7. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2018. 432 p.

8. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 340 p.

9. Freeman A. Pro ASP.NET Core 6. 9th ed. New York : Apress, 2022. 1186 p.

10. Король О. В. Системи управління базами даних : підручник. Київ : КПІ ім. І. Сікорського, 2021. 312 с.

11. Smith J. Entity Framework Core in Action. 2nd ed. Shelter Island : Manning Publications, 2021. 576 p.

12. Алгоритми та структури даних : навч. посіб. / Т. О. Короткова та ін. Львів : Новий Світ-2000, 2023. 280 с.

13. Masse M. REST API Design Rulebook. Sebastopol : O'Reilly Media, 2011. 116 p.

14. OpenAPI Specification v3.1.0. OpenAPI Initiative. URL: https://spec.openapis.org/oas/v3.1.0 (date of access: 10.05.2026).

15. OWASP Top 10:2021 – The Ten Most Critical Web Application Security Risks. OWASP Foundation. URL: https://owasp.org/Top10/ (date of access: 20.04.2026).

16. Інформаційна безпека та криптографія : підручник / В. М. Томашевський, О. Г. Рибальський. Вінниця : ВНТУ, 2024. 345 с.

17. Osherove A. The Art of Unit Testing: with examples in C#. 3rd ed. Shelter Island : Manning Publications, 2023. 360 p.

18. Тестування програмного забезпечення : навч. посіб. / О. І. Коваленко, В. П. Сидоренко. Харків : ХНУРЕ, 2022. 210 с.

19. Гамма Е., Хелм Р., Джонсон Р., Вліссідес Дж. Прийоми об'єктно-орієнтованого проєктування. Патерни проєктування. Київ : в-во «Діалектика», 2021. 368 с. (Джерело для обґрунтування використання патернів, таких як DTO та Repository).

20. Williams N. The Borg Rating of Perceived Exertion (RPE) scale. Occupational Medicine. 2017. Vol. 67, Issue 5. P. 404–405. (Наукове обґрунтування шкали RPE у контексті гігієни праці).

21. Chinnathambi K. Learning React: A Hands-On Guide to Building Maintainable Rich-Internet Applications. 4th ed. Addison-Wesley Professional, 2022. 512 p. (Сучасне джерело по React).

22. ДСТУ ISO/IEC/IEEE 12207:2023. Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Київ : ДП «УкрНДНЦ», 2023. 124 с. (Критично важливе джерело для підрозділу 4.2 про життєвий цикл та тестування).

23. Кравченко В. І. Проєктування та розробка вебзастосунків на платформі .NET : навч. посіб. Житомир : ЖДТУ, 2023. 210 с. (Україномовне джерело по .NET).

24. React Documentation. React – A JavaScript library for building user interfaces. URL: https://react.dev/ (date of access: 10.05.2026). (Офіційна документація).

25. Microsoft Learn. ASP.NET Core documentation. URL: https://learn.microsoft.com/en-us/aspnet/core/ (date of access: 12.05.2026). (Офіційна документація по бекенду).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка адаптивної інформаційної системи планування мікроциклів рухової активності як засіб превенції гіподинамічних станів

Виконав студент4 курсу
групи ПД-44
Куцик Віктор Миколайович
Керівник роботи
Викладач Бажан Юрій Павлович

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: превенція гіподинамічних станів за рахунок використання адаптивної інформаційної системи планування мікроциклів рухової активності.

Об'єкт дослідження: процес планування планування мікроциклів рухової активності як засіб превенції гіподинамічних станів

Предмет дослідження: методи, засоби та технології створення веб-застосунків для планування мікроциклів рухової активності.

2

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область та обгрунтувати вимоги до інформаційної системи превенції гіподинамічних станів, які характеризуються спазмами, остеохондрозом, падінням продуктивності та хронічною втомою.
2. Провести моделювання функціональних та нефункціональних вимог до розроблюваного програмного забезпечення.
3. Здійснити вибір програмних платформ та технологічних засобів для реалізації клієнт-серверної системи.
4. Спроєктувати архітектуру системи, реляційну базу даних та модель алгоритму поведінкової адаптації.
5. Реалізувати клієнт-серверний Web-застосунок для адаптивного планування мікроциклів.
6. Забезпечити захист інформації та провести комплексне тестування розробленої системи.

3

АНАЛІЗ АНАЛОГІВ

Критерій	Nike Training Club	Stretchly	Wakeout	Freeletics	ACM
Тип системи	База фітнес-тренувань	Таймер перерв для ПК	Каталог офісних вправ	ШІ-фітнес тренер	Адаптивна система мікроциклів
Автоматична генерація мікроциклів	–	–	–	+	+
Адаптація за рівнем скутості (RPE)	–	–	–	–	+
Вправи націлені на превенцію гіподинамії	–	+	+	–	+
Динамічний розрахунок часу вправи	–	–	–	–	+
Врахування наявного інвентарю/локації	+	–	+	+	+
Аналітичні можливості стану користувача	+	–	–	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

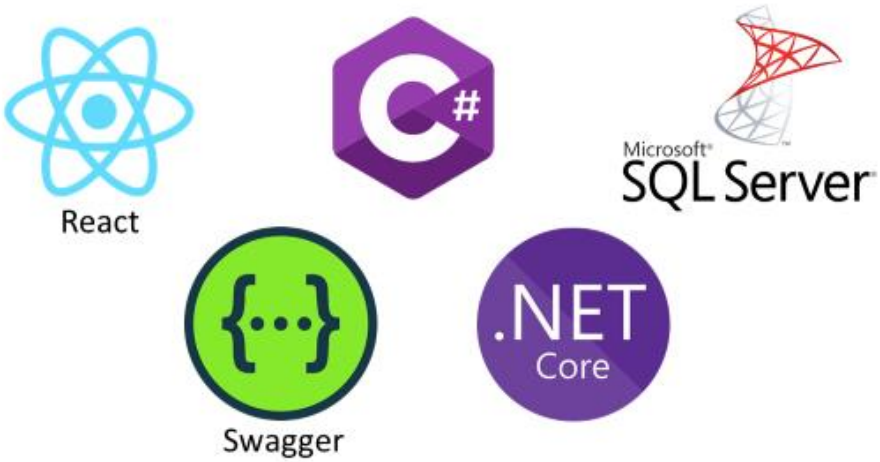
1. Збір вхідних даних користувача (робочий графік, наявність інвентарю, проблемні зони).
2. Автоматична генерація мікроциклів (рухових пауз), балансуючи навантаження на різні групи м'язів (шия, спина, очі тощо).
3. Збір зворотного зв'язку після мікроциклу за допомогою RPE.
4. Візуалізація аналітики та прогресу адаптації користувача (Радарний графік для відображення проблемних зон, Лінійний графік для відображення динаміки адаптації).

Нефункціональні вимоги:

1. Доступність системи через веб-браузер на десктопних та мобільних пристроях завдяки архітектурі SPA (React).
2. Використання односторінкової архітектури для швидкого введення оцінки RPE в один клік та Virtual DOM для миттєвої адаптивності інтерфейсу без перезавантаження сторінки.
3. Сервер стабільно забезпечує обслуговування до 10 000 одночасних користувацьких запитів без блокування потоків та падіння продуктивності.
4. Надійний захист API та бази даних від поширених веб-вразливостей (SQL-ін'єкцій, XSS, Mass Assignment).

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

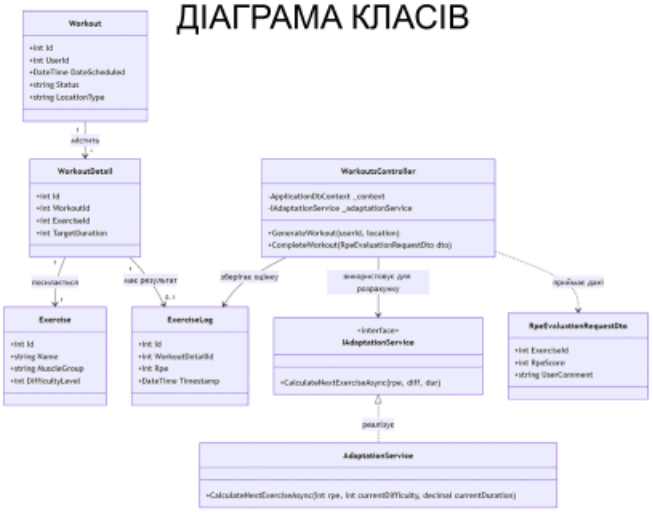


6

ДІАГРАМА ПРЕЦЕДЕНТІВ

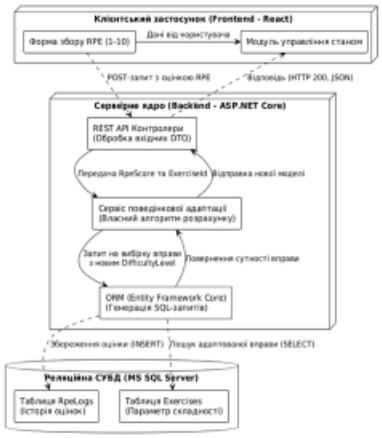
7

ДІАГРАМА КЛАСІВ



8

КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА РОЗРОБЛЮВАНОЇ СИСТЕМИ



9

УЗАГАЛЬНЕНИЙ АЛГОРИТМ АДАПТИВНОГО ПЛАНУВАННЯ

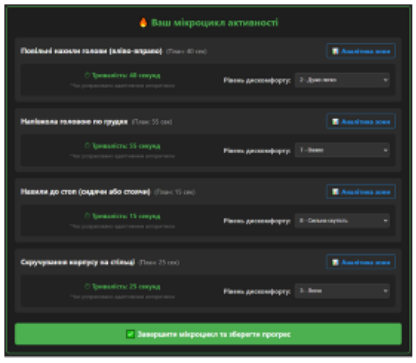
10

СТРУКТУРА РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

11

ЕКРАННІ ФОРМИ

Згенерований руховий мікроцикл

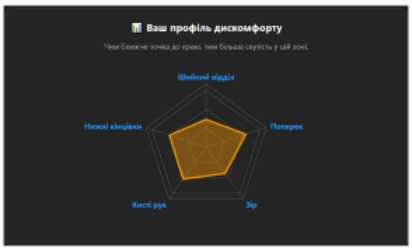


Оцінювання мікроциклу користувачем

12

ЕКРАННІ ФОРМИ

Графік алгоритму адаптації



Профіль дискомфорту користувача

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1) Куцик В.М., Бажан Ю.П. Визначення вимог до веб-застосунку для адаптивного планування тренувань. Матеріали II Всеукраїнської науково-технічної конференції "Виклики та рішення в програмній інженерії", 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. С.163-165.

2) Куцик В.М., Бажан Ю.П. Забезпечення безпеки інформації у web-застосунку для адаптивного планування тренувань. Матеріали VII Всеукраїнської науково-технічної конференції "Застосування програмного забезпечення в інформаційно-комунікаційних технологіях", 23 квітня 2026 року, Київ, ДУІКТ. Збірник тез. С.35-37.

14

ВИСНОВКИ

1. Проаналізовано предметну область та сформовано комплекс вимог до інтелектуальної системи превенції гіподинамічних станів.
2. Визначено ключові функціональні потреби та нефункціональні обмеження
3. Спроєктовано клієнт-серверну архітектуру, реляційну базу даних з 4 основними сутностями та модель алгоритму адаптації на основі шкали RPE.
4. Обрано бібліотеку React для клієнтської частини, фреймворк ASP.NET Core для серверної логіки та реляційну СУБД MS SQL Server для збереження даних.
5. Програмно реалізовано готовий адаптивний вебзастосунок із використанням сучасного стеку технологій (React, ASP.NET Core, MS SQL Server).
6. Забезпечено 3 рівні захисту даних (DTO, ORM, XSS) та успішно проведено комплексне тестування системи, що підтвердило її ефективність.

15

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using SmartWorkoutPlanner.API.Models;

namespace SmartWorkoutPlanner.API.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext(
            DbContextOptions<ApplicationDbContext> options)
            : base(options) { }

        public DbSet<Exercise> Exercises { get; set; }
        public DbSet<Workout> Workouts { get; set; }
        public DbSet<WorkoutDetail> WorkoutDetails { get; set; }
        public DbSet<ExerciseLog> ExerciseLogs { get; set; }

        protected override void OnModelCreating(
            ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
        }
    }
}
```

Exercise.cs

```
namespace SmartWorkoutPlanner.API.Models
{
    public class Exercise
    {
        public int Id { get; set; }
        public string Name { get; set; } = string.Empty;
        public string? MuscleGroup { get; set; }
        public string? Description { get; set; }

        // Рівень складності: 1 - Базовий, 2 - Середній
        public int DifficultyLevel { get; set; }
    }
}
```

RpeEvaluationRequestDto.cs

```
using System.ComponentModel.DataAnnotations;

namespace SmartWorkoutPlanner.API.Models.DTOs
{
    public class RpeEvaluationRequestDto
    {
        [Required]
        public int ExerciseId { get; set; }

        [Required]
        [Range(1, 10, ErrorMessage = "RPE від 1 до 10")]
        public int RpeScore { get; set; }

        [StringLength(500)]
        public string? UserComment { get; set; }
    }
}
```

AdaptationService.cs

```
using SmartWorkoutPlanner.API.Models;
using System;
using System.Threading.Tasks;

namespace SmartWorkoutPlanner.API.Services
{
    public class AdaptationService : IAdaptationService
    {
        public async Task<ExerciseParameters> CalculateNextExerciseAsync(
            int rpeScore,
            int currentDifficulty,
            decimal currentDuration)
        {
            int targetDifficulty = currentDifficulty;
            decimal targetDuration = currentDuration;

            if (rpeScore <= 3)
            {
                if (targetDifficulty < 3) targetDifficulty++;
                targetDuration += 5.0m;
            }
            else if (rpeScore >= 7)
            {
                if (targetDifficulty > 1) targetDifficulty--;
                targetDuration = Math.Max(15.0m, targetDuration - 5.0m);
            }
            else
            {
                targetDuration = currentDuration;
            }

            return new ExerciseParameters
            {
                TargetDifficulty = targetDifficulty,
                TargetDuration = targetDuration
            };
        }
    }
```

```
        }
        AnalyticsController.cs
        using Microsoft.AspNetCore.Mvc;
        using Microsoft.EntityFrameworkCore;
        using SmartWorkoutPlanner.API.Models;
        using System;
        using System.Linq;
        using System.Threading.Tasks;

        namespace
SmartWorkoutPlanner.API.Controllers
        {
          [Route("api/[controller]")]
          [ApiController]
          public class AnalyticsController :
ControllerBase
          {
            private readonly FitnessAppDbContext
_context;

            public
AnalyticsController(FitnessAppDbContext context)
            {
              _context = context;
            }


[HttpGet("recommendation/{userId}/{exerciseId}")]
            public async Task<IActionResult>
GetRecommendation(
              int userId, int exerciseId)
            {
              var lastLog = await
_context.ExerciseLogs
                .Include(l => l.WorkoutDetail)
                .ThenInclude(wd => wd!.Exercise)
                .Where(l =>
l.WorkoutDetail!.Workout!.UserId == userId
                    &&
l.WorkoutDetail.ExerciseId == exerciseId)
                .OrderByDescending(l =>
l.WorkoutDetail!.Workout!.DateScheduled)
                .FirstOrDefaultAsync();

              if (lastLog == null) return Ok(new {
advice = "Нова вправа" });

              int rpe = lastLog.Rpe ?? 0;
              string advice = "";
              string zone =
lastLog.WorkoutDetail!.Exercise!.MuscleGroup;

              if (rpe >= 8) {
                if (zone == "Зір") advice = "🚨
Зверніться до офтальмолога";
                else if (zone == "Шийний відділ")
advice = "🚨 Защемлення";
                else advice = "Зменшуємо
навантаження";
              }

              return Ok(new { advice = advice });
            }
          }
        }
        WorkoutSession.jsx
        import React, { useState, useEffect } from
'react';
        import axios from 'axios';

        const WorkoutSession = ({ exercise }) => {
          const [timeLeft, setTimeLeft] =
useState(exercise.duration);
          const [isFinished, setIsFinished] =
useState(false);
          const [rpe, setRpe] = useState(5);

          useEffect(() => {
            if (timeLeft > 0) {
              const timerId = setTimeout(() =>
                setTimeLeft(timeLeft - 1), 1000);
              return () => clearTimeout(timerId);
            } else {
              setIsFinished(true);
            }
          }, [timeLeft]);

          const handleSubmitRpe = async () => {
            try {
              await
axios.post('http://localhost:5087/api/workouts/complete
', {
                exerciseId: exercise.id,
                rpeScore: parseInt(rpe)
              });
              alert('Дані збережено! Алгоритм
адаптовано.');
            } catch (error) {
              console.error('Помилка:', error);
            }
          };

          return (
            <div className="workout-container">
              <h2>{exercise.name}</h2>
              {!isFinished ? (
                <div className="timer">
                  Залишилось: {timeLeft} сек
                </div>
              ) : (
                <div className="rpe-form">
                  <h3>Оцініть рівень
дискомфорту (RPE)</h3>
                  <select
                    value={rpe}
                    onChange={(e) =>
setRpe(e.target.value)}>
                    <option value="1">1 -
Відмінно, немає болю</option>
                    <option value="3">3 -
Легко</option>
                    <option value="5">5 - Помірна
скутість</option>
                    <option value="8">8 - Сильна
скутість</option>
```

```
                        <option value="10">10 - Біль / Неможливо</option>
                    </select>
                    <button onClick={handleSubmitRpe}>
                        Відправити
                    </button>
                </div>
            )}
        </div>
    );
};

export default WorkoutSession;
```

ExerciseChart.jsx

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { LineChart, Line, XAxis, YAxis, Tooltip } from 'recharts';

const ExerciseChart = ({ userId, exerciseId }) => {
    const [data, setData] = useState([]);

    useEffect(() => {
        const fetchAnalytics = async () => {
            const res = await axios.get(
`http://localhost:5087/api/analytics/${userId}/${exerciseId}`
            );
            setData(res.data);
        };
        fetchAnalytics();
    }, [userId, exerciseId]);

    return (
        <div className="chart-wrapper">
            <h3>Прогрес адаптації</h3>
            <LineChart width={400} height={250} data={data}>
                <XAxis dataKey="date" />
                <YAxis yAxisId="left" stroke="#4CAF50" />
                <YAxis yAxisId="right" orientation="right" stroke="#ff9800" />
                <Tooltip />
                <Line yAxisId="left" type="monotone"
                    dataKey="duration" stroke="#4CAF50" />
                <Line yAxisId="right" type="monotone"
                    dataKey="rpe" stroke="#ff9800" />
            </LineChart>
        </div>
    );
};

export default ExerciseChart;
```

Workout.cs

```
using System;
using System.Collections.Generic;

namespace SmartWorkoutPlanner.API.Models
{
    public class Workout
    {
        public int Id { get; set; }
        public int UserId { get; set; }
        public DateTime DateScheduled { get; set; }

        // Статус: "Pending", "Completed", "Skipped"
        public string Status { get; set; } = "Pending";

        // Тип локації: "Office", "Home"
        public string LocationType { get; set; } = "Office";

        public virtual ICollection<WorkoutDetail>
            WorkoutDetails { get; set; } = new List<WorkoutDetail>();
    }
}
```

WorkoutsController.cs

```
[HttpGet("generate/{userId}/{location}")]
public async Task<ActionResult<WorkoutDto>> GenerateWorkout(
    int userId, string location)
{
    // Отримуємо вправи, які зараз не в "відкаті"
    var availableExercises = await _context.Exercises
        .Where(e => !_context.ExerciseLogs
            .Any(log => log.WorkoutDetail.ExerciseId == e.Id &&
                log.WorkoutDetail.Workout.UserId == userId &&
                log.Timestamp > DateTime.Now.AddHours(-3)))
        .ToListAsync();

    var workout = new Workout {
        UserId = userId,
        DateScheduled = DateTime.Now,
        LocationType = location
    };

    // Алгоритм випадкового вибору 3-х різних зон
    var selected = availableExercises
        .OrderBy(x => Guid.NewGuid()).Take(3).ToList();

    foreach (var ex in selected) {
```

```
            workout.WorkoutDetails.Add(new
WorkoutDetail {
                ExerciseId = ex.Id,
                TargetWeight = 30 // Базовий час у
сек.
            });
        }

        _context.Workouts.Add(workout);
        await _context.SaveChangesAsync();
        return Ok(MapToDto(workout));
    }
    ExerciseLog.cs

    namespace SmartWorkoutPlanner.API.Models
    {
        public class ExerciseLog
        {
            public int Id { get; set; }
            public int WorkoutDetailId { get; set; }
            public int? Rpe { get; set; }
            public string? UserNote { get; set; }
            public DateTime Timestamp { get; set; }
= DateTime.Now;

            public virtual WorkoutDetail?
WorkoutDetail { get; set; }
        }
    }

    Dashboard.jsx

    import React, { useState, useEffect } from
'react';
    import { Radar, RadarChart, PolarGrid,
        PolarAngleAxis, ResponsiveContainer }
from 'recharts';
    import axios from 'axios';

    const Dashboard = ({ userId }) => {
        const [profile, setProfile] = useState([]);

        useEffect(() => {

axios.get(`/api/analytics/profile/${userId}`)
            .then(res => setProfile(res.data));
        }, [userId]);

        return (
            <div className="dashboard-grid">
                <div className="card">
                    <h3>Ваш профіль
дискомфорту</h3>
                    <ResponsiveContainer
width="100%" height={300}>
                        <RadarChart data={profile}>
                            <PolarGrid stroke="#444" />
                            <PolarAngleAxis
dataKey="subject" stroke="#ccc" />
                            <Radar
                                name="RPE"
                                dataKey="a"
                                stroke="#2196F3"
                                fill="#2196F3"
                                fillOpacity={0.6}
                            />
                        </RadarChart>
                    </ResponsiveContainer>
                </div>
            </div>
        );
    };

    App.css

    :root {
      --bg-dark: #121212;
      --card-bg: #1e1e1e;
      --primary: #2196f3;
      --accent-rpe: #ff9800;
    }

    body {
      background-color: var(--bg-dark);
      color: #ffffff;
      font-family: 'Segoe UI', sans-serif;
    }

    .card {
      background: var(--card-bg);
      border-radius: 12px;
      padding: 20px;
      border: 1px solid #333;
      margin-bottom: 20px;
    }

    .btn-generate {
      background: linear-gradient(45deg, #2196F3,
#00BCD4);
      border: none;
      padding: 15px 30px;
      color: white;
      border-radius: 8px;
      font-weight: bold;
      cursor: pointer;
      transition: transform 0.2s;
    }

    .btn-generate:hover {
      transform: scale(1.05);
    }

    .rpe-badge-high { color: #f44336; font-weight:
bold; }
    .rpe-badge-low { color: #4caf50; }
```