МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Інформаційна підсистема підтримки роботи тренера залу функціонального тренінгу “Парашут”»

за спеціальністю 122 «Комп’ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студентка групи ІТ.м-01 Шкура Анастасія Володимирівна

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «___» грудня 2021 р.

Науковий керівник _____ к.т.н., доц., Ващенко С.М.

Голова комісії _____ Шифрін Д.М.

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів
без відповідних посилань.

Студент _____

Суми-2021

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. кафедри ІТ

_____
В. В. Шендрик
«___» _____ 2021 р.

ЗАВДАННЯ
на кваліфікаційну роботу магістра студентові

Шкура Анастасія Володимирівна

1 Тема проекту *Інформаційна підсистема підтримки роботи тренера залу функціонального тренінгу "Парашут"*

затверджена наказом по університету від «29» жовтня 2021 р. № 0787-VI

2 Термін здачі студентом закінченого проекту «17» грудня 2021 р.

3 Вхідні дані до проекту Вимоги до програмного забезпечення від замовника, перелік файлів для розробки дизайну, дані для заповнення бази даних

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) Аналіз предметної області, постановка задачі та методи дослідження, проектування інформаційної підсистеми, розробка інформаційної підсистеми.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Актуальність та мета розробки, мета та цілі проекту, аналіз існуючих аналогів, планування та проектування, інструменти та етапи розробки

6. Консультанти випускної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____

Завдання прийняв до виконання _____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Формування завдання та проблеми проекту	18.10.2021	
2	Формування та деталізація мети проекту	19.10.2021 - 20.10.2021	
3	Аналіз програмних продуктів-аналогів	21.10.2021 - 22.10.2021	
4	Розробка WBS	25.10.2021 - 26.10.2021	
5	Розробка OBS	28.10.2021	
6	Створення діаграми Ганта	28.10.2021 - 29.10.2021	
7	Проведення аналізу ризиків	01.11.2021 - 03.11.2021	
8	Створення IDEF-діаграми	04.11.2021 - 07.11.2021	
9	Створення Telegram-bot для розкладу	08.11.2021 - 11.11.2021	
10	Створення календпрю тренувань	12.11.2021 - 22.11.2021	
11	Створення Telegram-bot для опитувань	23.11.2021 - 30.11.2021	
12	Створення сторінки юзерів	01.12.2021 - 06.12.2021	
13	Створення сторінки статистики здоров'я	07.12.2021 - 12.12.2021	
14	Тестування за тест-кейсами	13.12.2021 - 21.12.2021	
15	Презентація проекту	22.12.2021	

Магістрант _____ Шкура А.В.

Керівник роботи _____ к.т.н., доц. Ващенко С.М.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Інформаційна підсистема підтримки роботи тренера залу функціонального тренінгу “Парашут”».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 30 найменувань, додатків. Загальний обсяг роботи − 95 сторінок, у тому числі 57 сторінок основного тексту, 5 сторінок списку використаних джерел, 33 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці інформаційної підсистеми організації роботи тренера окремого тренувального залу, що включає у себе Telegram-бота, кабінет тренера та розробку бази даних.

В роботі проведено аналіз існуючих систем-аналогів, дослідження сучасних методів розробки подібних систем та виконана постановка завдання.

У роботі виконано проектування підсистеми: розроблено IDEF0 та Use Case діаграми. Також виконано планування робіт по реалізації проекту та визначення потенційних ризиків.

Результатом проведеної роботи є розроблена інформаційна підсистема, що включає у себе Telegram-бота, кабінет тренера та бази даних, які будуть використовуватись тренером тренажерного залу “Парашут” для ведення обліку абонементів та відвідувань.

Практичне значення роботи полягає у забезпеченні тренера зручним інструментом відстеження прибутковості спортивного залу, ведення календаря тренувань.

Ключові слова: MySQL, .NET, TypeScript, MUI, JSON, WEB-ДОДАТОК, TELEGRAM API.

ЗМІСТ

ВСТУП

Взаємозв'язок між фізичною активністю та фізичним здоров'ям не викликає сумнівів на сьогоднішній день. Розуміння того, що сидячий спосіб життя має негативний вплив на здоров'я є розвиненим як серед учених, так і серед директивних органів, що малорухливість нині визнана серйозною проблемою охорони здоров'я [1]. Всесвітня організація охорони здоров'я підрахувала, що фізична неактивність є четвертим провідним фактором ризику глобальної смертності, що відповідає за 6% смертей у всьому світі [2].

Тренер відіграє важливу роль у фізичному розвитку свого вихованця, а він, в свою чергу, покладається на знання та довіряє інструктору своє здоров’я. На сьогодні для інструкторів існують додатки, які допомагають їм виконувати свої обов’язки. Хороший додаток для тренера допомагає ефективно керувати клієнтами, тренуваннями та адміністративними завданнями. Персональні тренери повинні відстежувати тренування, членство, платежі та прогрес клієнтів. З кожним новим клієнтом ці обов’язки стають все більш обтяжливими і займають багато часу. Додатки допомагають тренерам організувати всю цю інформацію та отримати доступ до неї в одному місці.

Додатки для персональних тренерів також заощаджують час під час створення тренувань. За допомогою програмного забезпечення персональні тренери можуть залучати більше клієнтів, не втрачаючи часу, щоб зосередитися на наданні якісних послуг.

Об’єкт дослідження – інформаційні технології підтримки діяльності тренерів спортивних залів.

Предмет дослідження – засоби реалізації при організації роботи інформаційних систем.

Наукова новизна роботи полягає в тому, що запропоновано технологію реалізації інформаційної підсистеми підтримки діяльності тренера спортивної зали.

Метою даної роботи є створити інформаційну підсистему підтримки роботи тренера залу функціонального тренінгу “Парашут”, яка допоможе будувати заняття, слідкувати за їх відвідуванням, а також стежити за витратами на підтримку залу та обладнання.

Для того, щоб досягти мети даної роботи потрібно виконати такі задачі:

- проаналізувати предметну область, скласти перелік вимог до програмного продукту;
- провести планування робіт по реалізації проекту;
- виконати моделювання системи, обрати технології та засоби реалізації;
- розробити модуль календаря тренувань та відвідувань;
- розробити модуль менеджменту ресурсів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Комплексні наслідки відкриттів у промислових, автомобільних та інформаційних сферах, а також інших соціально-культурних змін у всьому світі призвели до радикальних змін у способах виконання повсякденних завдань людьми, що в свою чергу знизило необхідність у фізичній активності для середньостатистичної людини. Згідно з даними всесвітньої організації охорони здоров'я більше чверті дорослого населення світу (1,4 мільярда дорослих) мають недостатній рівень фізичної активності, а позитивної тенденції ми не бачили за останні два десятиліття. [3]

Користь фізичної активності для здоров’я добре відома. Позитивні наслідки в себе включають зниження ризику неінфекційних захворювань, таких як серцево-судинні захворювання, діабет та різні види раку, а також позитивний вплив на психічне здоров’я за рахунок зменшення депресії, стресових реакцій і, можливо, затримки наслідків хвороби Альцгеймера та інших форм деменції. Крім того, фізична активність є ключовим фактором, що визначає витрати енергії, який підтримує енергетичний баланс і здорову масу тіла. [4] ВООЗ рекомендує дорослим (включаючи людей похилого віку) щотижня виконувати принаймні 150 хвилин аеробної фізичної активності помірної інтенсивності або 75 хвилин аеробної фізичної активності високої інтенсивності або еквівалентну комбінацію. [3]

Технологічний прогрес призвів до посилення оцифрування в сфері охорони здоров’я та спорту [5]. Поява доступних додатків для смартфонів у Google Play та App Store, сприяло кращому розумінню здоров’я людини, дозволяючи нам збирати величезні обсяги медичних даних [6]. Зокрема, деякі вдосконалення в технології додатків (наприклад, вбудована камера для оцінки

серцевого ритму, акселерометри тощо) відкрили нові можливості для збору відповідної інформації в клінічних і спортивних умовах.

Використання додатків для збору даних також привернуло широку увагу серед спортивних професіоналів і вчених. Насправді, деякі програми вже розроблені для збору фізіологічних, кінантропометричних та спортивних даних [7]. Використання додатків для збору даних, ймовірно, є найпопулярнішим у розважальних спортивних заходах, хоча вони також використовуються в контексті високопродуктивного спорту [8]. У високоефективних видах спорту досвід, необхідний для кількісної оцінки фізичної працездатності спортсмена за допомогою традиційних методів, часто є дорогим і не зручним для користувача, особливо для тренерів [7].

Додатки роблять вимірювання фізичної продуктивності для тренерів більш доступними. Гарним прикладом є різні програми, призначені для відстеження відстані або темпу під час занять спортом на витривалість. Однією з таким програм є Strava. Серед найпривабливіших функцій Strava є його здатність відстежувати всі аспекти зареєстрованої фізичної активності (наприклад, частота серцевих скорочень, темп, відстань) і здатність аналізувати їх щохвилинно [9]. Також, для змагань у спорті вже існують перевірені програми, призначені для тренерів, що оцінюють дані спортивних результатів, таких як механічні результати спринту [10] та техніка бігу [11].

Дослідження, що вивчає комерційні додатки для фізичної активності, показало, що соціальний компонент має великий потенціал для заохочення до фізичної активності. Можливість ділитися результатами фізичної активності та прогресом у спільнотах чи соціальних мережах є необхідним стимулом для людей. Близько 1300 дорослих взяли участь у дослідженні, більше половини з яких використовували комерційний додаток для фізичної активності (наприклад, Garmin, Fitbit, Strava). Результати показали, що більш суперницькі люди займались значно вищими рівнями фізичної активності завдяки ігровим стимулам та винагородам, вбудованим у програми. Таким чином, це дослідження показує, що соціальні компоненти таких додатків є особливо

корисними для сприяння залученню до фізичної активності через їхню здатність сприяти соціальній підтримці, а також позитивно впливати на мотивацію та віру в свою здатності. Однак, було також виявлено, що онлайн-взаємодія може мати негативний вплив на тих, хто тренується, в разі якщо проводиться пряме порівняння між користувачами [12].

Дослідження [13], в якому проаналізували 127 iOS фітнес-додатків на дотримання теорії здоров'я та зміни поведінки, довели, що часто вони не містять та не дотримуються теоретичних відомостей. Це також свідчить про те, що для досягнення найкращого рішення з кращими результати здоров'я повинна бути співпраця між тренерами, експертів зі зміни поведінки та розробниками додатків [13]. Більше того, нещодавнє дослідження систематичних оглядів показало, що лише 14 з 21 додатка, заснованого на фізичній активності, показали значне покращення здоров’я у тренуючихся [14].

Фізична підготовка вважається дуже індивідуальною, що є основною причиною необхідності експерта, який би зміг допомогти в тренуванні. Також дуже важко встигати за новими звичками, особливо коли йдеться про харчування та фізичні вправи. Особистий тренер забезпечує підзвітність і заохочення, які не може повторити жодна програма. На сьогоднішній день індустрія фітнес-додатків, в більшій мірі покладається на гейміфікування додатків, щоб мотивувати тренуючихся. Але ігри незмінно приходять до кінця. Коли новизна «призів», таких як бали, таблиця лідерів і кількість кроків, минає, досвід стає одноманітним і користувачі залишають тренування. Для довготривалого ефекту потрібно зробити так, щоб відвідувачу було завжди цікаво починати тренування, адже одноманіття швидко набридає. Це може надати саме тренер. Адже він має правильні знання про те як побудувати та урізноманітнити тренувальний процес.

1.2 Огляд існуючих програмних продуктів-аналогів

Як правило, фітнес-додатки на даний момент зосереджені на забезпеченні конкретних типів тренувань, їх планування та трекерів для бігу як частини тренерських заходів. Однак, цей тип додатків на більш просунутому рівні не застосовуються. Було проведено дослідження взаємодії людини та комп'ютера, щоб продемонструвати ефективність фітнес-додатків для заохочення людей до активного способу життя, у світлі літератури про зміни поведінки та мотивації. Однак існує порівняно мало досліджень такого взаємозв'язку, які зосереджуються саме на користувацькому досвіді, та оцінює рішення з точки зору підтримки довготривалої продуктивності [15].

Дослідження 2017 року говорить, що незважаючи на збільшення популярності додатків для здоров'я, цього недостатньо для того, щоб вплив лише на основі додатків був ефективним у збільшенні фізичної активності населення. Також було показано, що відмова від додатка пов'язана з кількома причинами, серед яких низька зацікавленість, недостатня зручність для користувачів та відсутність бажаних функцій [16].

Дослідження показали важливість залучення експертів у тренування, а також те, що програми для фітнесу недостатньо зосереджені на цьому. Для того, щоб розглянути це питання більш детально, було проведено огляд існуючих фітнес-додатків на ринку, які стверджують, що пропонують коучингові втручання, щоб зрозуміти, як вони підтримують зміну поведінки, продуктивність і мотивацію, як вони записують і відстежують прогрес і цілі, візуалізацію особистої інформатики та загальний досвід користувача. Для цього було використано теорію самовизначення, щоб оцінити, як програми сприяють мотивації, а визначити афективні перспективи [15].

Для порівняння було обрано безкоштовні додатки, що доступні на iOS та Android в категоріях «Здоров'я та фітнес» або «Спорт», що пропонують коучинг як одну з функцій.

Структура порівняння була розроблена для мінімізації суб’єктивності результатів. Основою для оцінки когнітивного та мотиваційного аспектів була теорія самовизначення [17]. Крім того, фреймворк технології як досвід було використано для оцінки афективного аспекту взаємодії [18]. Аналіз проводилось за допомогою надавання кожному додатку оцінки від одного до 5 після його використання протягом тижня у восьми категоріях, до яких входили:

- Автономність – як вони допомагають відстежувати цілі та прогрес тренування;
- Компетентність – хід виконання тренування та загальний досвід користувача;
- Спорідненість – як вони пропонують відгуки, і поради щодо підтримки зміни поведінки та продуктивності;
- Чуттєвий досвід – як вони надавали інформацію про здоров’я та фізичні вправи;
- Емоційний досвід – як вони нагадували та мотивували;
- Композиційний досвід – дозволяли ділитися на соціальних платформах або спільноті;
- Просторово-часовий – як користувачі відчувають зв’язок із програмою та тренером;
- Смислотворча діяльність – як користувач розуміє програму.

Результат наведено в таблиці 1.1.

Таблиця 1.1 – Порівняльна таблиця фітнес-додатків

Додаток	Теорія самовизначення			Технології як досвід					Сума
	Автономність	Компетентність	Спорідненість	Чуттєвий	Емоційний	Композиційний	Просторово-часовий	Смислотворча діяльність	
Fitbit Coach	4	5	3	4	3	5	4	4	4.00
Workout Trainer	5	4	4	2	2	3	4	2	3.25
C25K	4	2	2	3	2	4	3	5	3.13
Freeletics Bodyweight	3	3	3	3	3	3	3	3	3.00
MapMyRun	2	3	4	2	2	4	4	3	3.00
Endomondo	2	3	3	2	2	4	4	3	2.88
PEAR	4	2	2	3	2	3	3	3	2.75
5K Runner	4	2	2	1	1	4	3	5	2.75
The RUN Experience	5	1	2	2	2	3	2	2	2.38

Загалом, програми не змогли створювали значущого зв’язку з користувачем. «Fitbit Coach» (рис. 1.1) отримав найбільшу оцінку в категорії технології як досвід роботи, головним чином тому, що інтерфейс був чистим та більш інтуїтивним, порівняно з іншими додатками. Цей додаток також мав кращу інтеграцію зворотного зв’язку, запитуючи користувачів, як вони почуваються одразу після вправи та скільки повторень їм вдалося зробити.

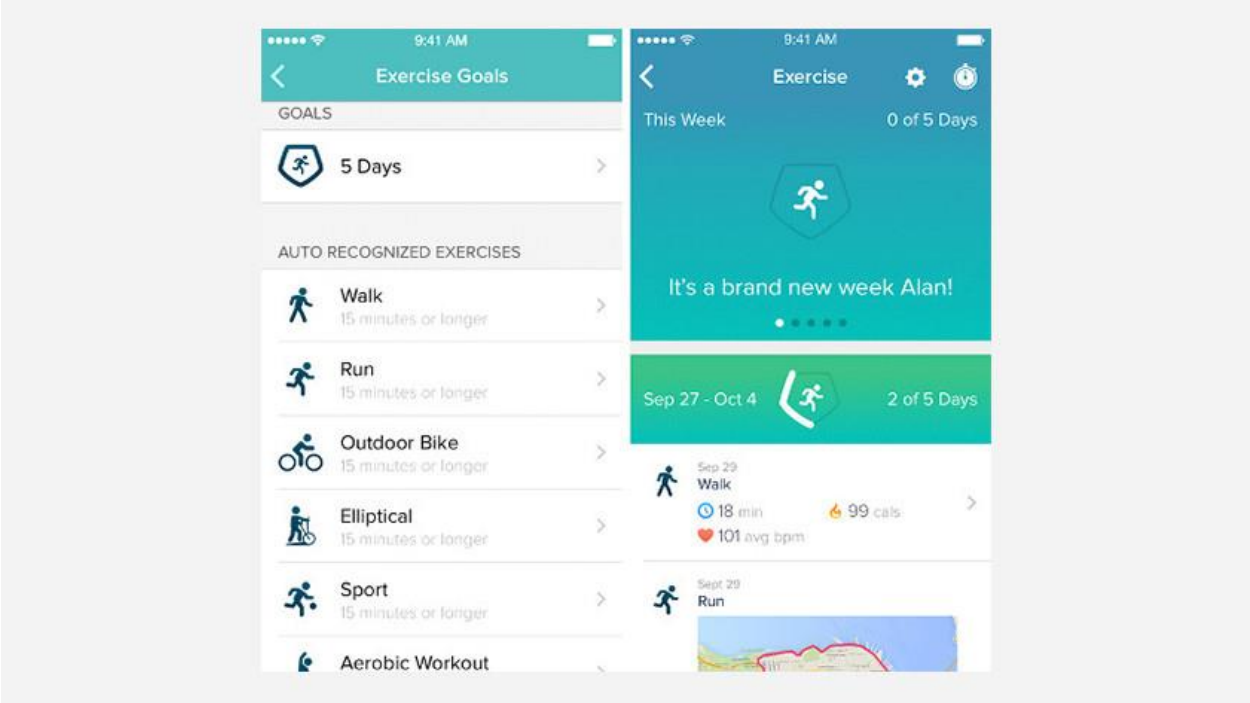


Рисунок 1.1 – Інтерфейс додатку Fitbit Coach

Як правило, більш фокусовані на взаємодії з користувачем додатки, отримали кращі результати в цьому секторі, оскільки вони могли зосередитися на користувачеві та забезпечувати простішу взаємодію. Наприклад, «C25K» (рис. 1.2) був другим кращим, оскільки він надавав гарне розуміння користувачеві, як досягти своєї мети, але також був зосереджений виключно на тому, щоб змусити його пробігти певну дистанцію.

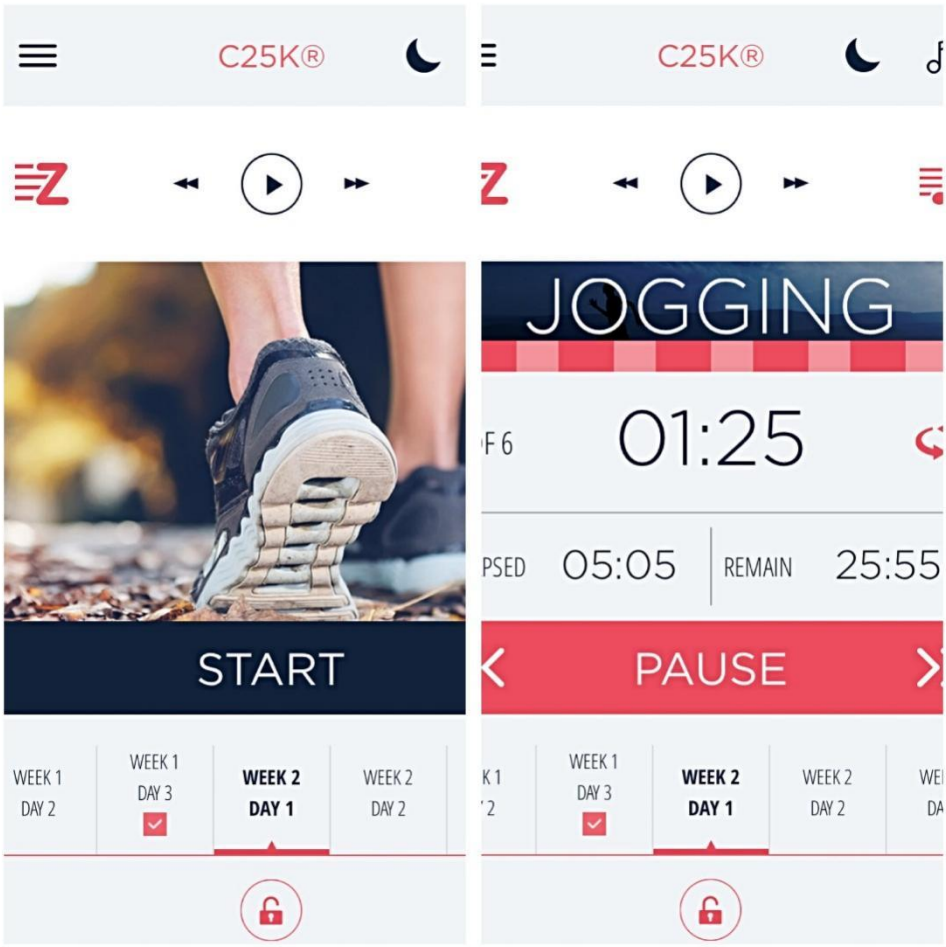


Рисунок 1.2 – Інтерфейс додатку C25K

На відміну від цього, «The RUN Experience» (рис 1.3) отримав найгірші результати, оскільки не було значущої взаємодії з користувачем, а скоріше набір деяких відео.

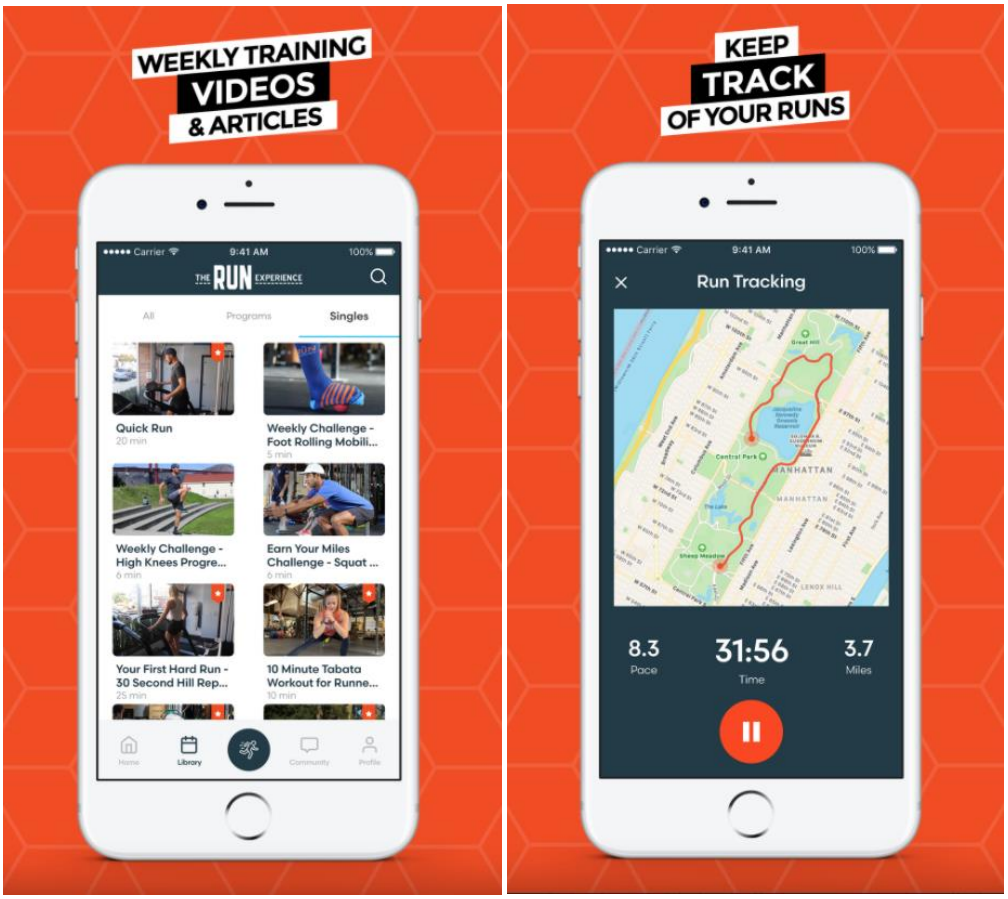


Рисунок 1.3 – Інтерфейс додатку The RUN Experience

Загалом, «коучинг» здебільшого застосовувався як форма персоналізованого втручання. Усі програми певною мірою пропонували персоналізоване навчання, однак існували різні способи реалізації цього. Деякі програми пропонували взаємодію з особистим тренером через чати та плани тренувань (у преміум-версії), інші пропонували персоналізовані тренування, включаючи рекомендації та пропозиції. Деякі додатки не робили акценту на персоналізації чи коучингу, незважаючи на заяву про це. Більше того, багато додатків не пояснювали, на чому базувалися персоналізовані тренування та рекомендації, і, здається, не враховували цілі, встановлені користувачем, де це було можливо.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою проекту є створення сайту, що допоможе тренерам планувати тренування, слідкувати за відвідувачами та фінансами, які витрачаються на підтримку залу функціонального тренування «Парашут».

Створена програма має містити інтуїтивно зрозумілий і зручний інтерфейс для користувача. Дозволяти слідкувати за змінами в потоках відвідувачів в реальному часі для того, аби планувати тренування в залежності від кількості відвідувачів. Програма повинна бути створена українською мовою.

Головними вікнами додатку повинні бути:

- календар тренувань;
- деталі про тренування;
- список користувачів;
- сторінка фінансового менеджменту.

Календар тренувань повинен мати можливість швидкого перегляду розкладу сесій по групам, при натисканні на конкретну сесію повинно з’являтися модульне вікно з деталями. Там повинні відображатися дата та час, група, яка відвідує, та список записаних на тренування. Тренер повинен мати можливість переносити та відміняти тренування. По завершенню тренування, він має можливість відмітити тих, хто не з’явився. Також на кожній даті повинна бути можливість створення тренувальної сесії. Ведення такого календаря у онлайн формі дозволяє наочно побачити свою зайнятість протягом тижня, розпланувати тренування на тиждень або місяць вперед, визначити, у які дні кількість відвідувань вища або нижча за звичайну і встановити додаткові сесії за необхідністю. Це також значно зменшує ризик виникнення непередбачуваних ситуацій, коли занадто багато відвідувачів

обрали один і той самий час для тренування, даючи можливість вчасно сповістити що набір на певну дату та час закрито.

У списку користувачів, тренер має можливість переглянути всіх, хто записаний у зал, переглянути стан абонементу, створити новий абонемент. Наявність такого функціоналу значно полегшує ведення обліку фінансів. На даний момент тренер використовує хоч і вірну, але застарілу методику: ведеться зошит із записами щодо відвідування та видачі абонементів, а також видаються спеціальні картки для відміток. Такий підхід має ряд суттєвих недоліків:

1) ведення зошиту не гарантує безпечне збереження даних, адже як фізичний об’єкт він може легко піддатися ряду небезпечних чинників: втрата, пошкодження і нерегулярне оновлення записів через людський фактор;

2) видача брендованих карток-абонементів, з одного боку, є гарним шляхом промоуції тренажерного залу, оскільки користувач може показати абонемент знайомим і потенційно залучити нових відвідувачів. Але слід зауважити, що при цьому: тренеру доводиться мати дублікат абонементу, щоб дані із абонементу користувача були актуальними і співпадали з реальною кількістю відвідувань; доводиться витрачати додаткові кошти на друк абонементів; велика кількість друкованого матеріалу не є екологічно-дружнім.

Сторінка фінансового менеджменту повинна надавати можливість перегляду витрат за певний проміжок часу, а також додавати нові записи. Цей набір функцій є дійсно важливим для тренера як для приватного підприємця, оскільки ведення аудиту витрат на утримання власного спортивного залу дає можливість оцінити прибутковість свого проекту. Таким чином, додаючи нові статті витрат можна побачити, яке обладнання потребує оновлення, або скільки вихідних матеріалів було придбано, додати новий запис про заплановані витрати, тощо.

Реалізація буде проводитись за допомогою таких технологій:

- Back end – .NET Core, Entity Framework, REST;
- Front end – react, typescript, redux, materialUI;

– База даних – MySQL.

.NET – це модульна платформа розробки програмного забезпечення, розроблена та підтримувана Microsoft для спрощення розробки настільних та веб-додатків. Це популярна безкоштовна платформа, яка в даний час використовується для різних типів додатків, оскільки вона надає середовище програмування для більшості етапів розробки програмного забезпечення. .NET добре підходить компаніям, які шукають широкий спектр функцій, таких як веб-служби, програмне забезпечення для настільних комп'ютерів та підтримка хмарної інфраструктури.

Для взаємодії з фронтендом буде використано архітектурний стиль REST. Для доступу до даних буде використано Entity Framework.

В проекту буде використано одна з найпопулярніших бібліотек JavaScript – React у поєднанні з typescript. Її швидкість, основа на компонентах, що дозволяє повторно використовувати код, а також ререндерити лише частину сторінки при зміні даних були вирішальними при виборі цього фреймворку. Для більш простого верстання було використано бібліотеку mui. Вона надає додаткові компоненти, а також їх базове стилювання, є легко кастомізованою, що прискорює побудову сторінок. Для управління станом додатку в конкретний момент часу використовується бібліотека Redux.

3 ПРОЕКТУВАННЯ

3.1 Моделювання в IDEF0

Процес проектування інформаційної системи розпочинається з розробки контекстної діаграми А-0. Вона представляє собою найвищий рівень абстракції, а також описує взаємодію інформаційної системи з навколишнім середовищем.

Основними компонентами даної діаграми є входи, дані управління, виходи та механізми.

Проаналізувавши основні компоненти системи, було сформовано перелік даних:

- входи – дані про активного користувача Telegram, дата нового тренування, дані про користувачів групи Telegram, дані про придбання матеріалів, дані про відвідування з Telegram, дані про придбання абонементів;
- дані управління – вимоги до відображення та збереження інформації на веб-сторінках, інструкція користувача;
- виходи – інформація про проведення тренування та відвідувачів, голосування про відвідування в Telegram, список та статистика витрат на утримання залу, інформація про клієнтів та абонементи, план тренувань;
- механізми – тренер, Telegram Bot, Web-додаток.

Контекстної діаграми А-0 відображена на рисунку 3.1.

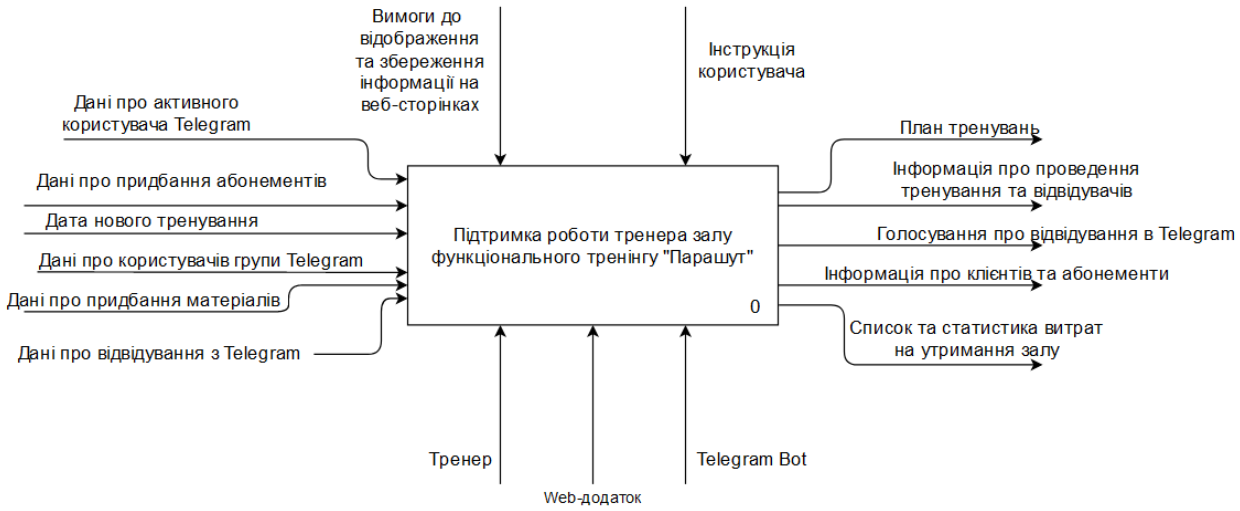


Рисунок 3.1 – Контекстна діаграма A-0

Оскільки, контекстна діаграма проводить опис системи у найбільш загальному виді, маємо потребу у декомпозиції. Це дозволить детальніше відобразити логіку послідовності робіт.

Для цього функціональний блок рівня A-0 розкладаємо на набір підфункцій. Після декомпозиції була розроблена діаграма IDEF1, що відображена на рисунку 3.2.

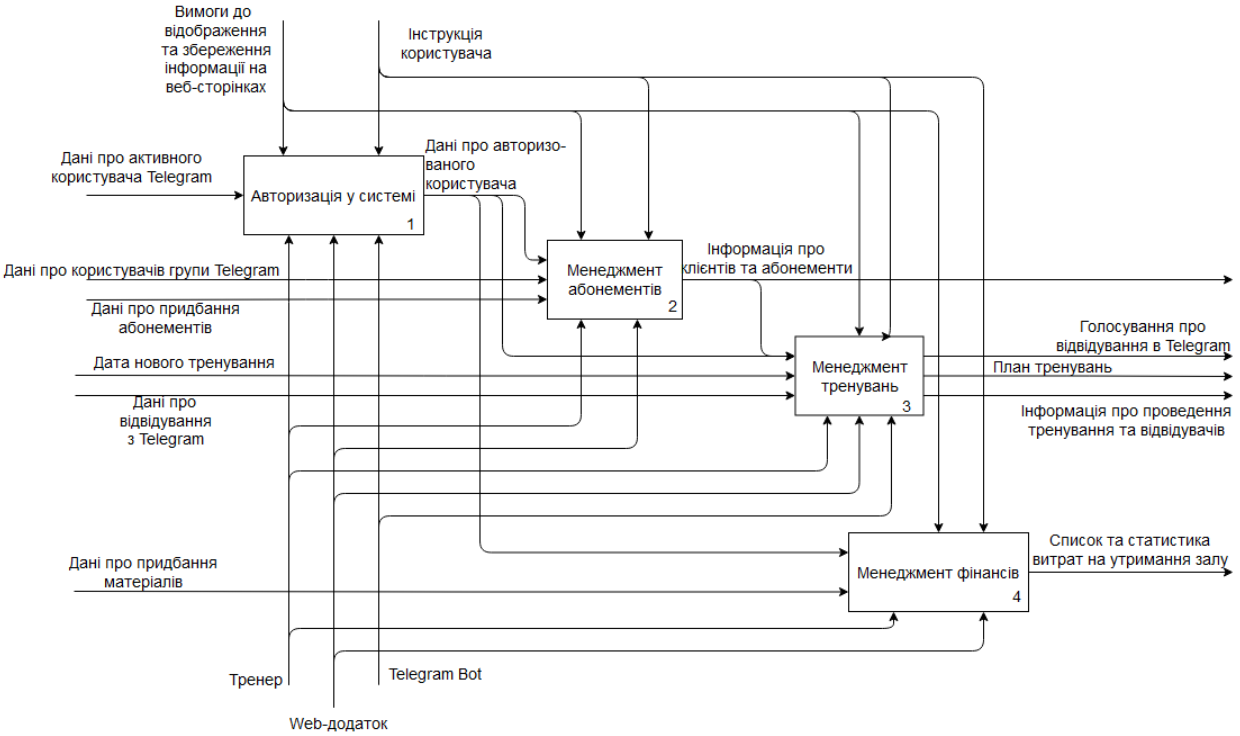


Рисунок 3.2 – Модель декомпозиції першого рівня

Тепер проведемо декомпозицію другого рівня, щоб точніше описати роботи систему. Діаграми наведено на рисунках 3.3 – 3.5

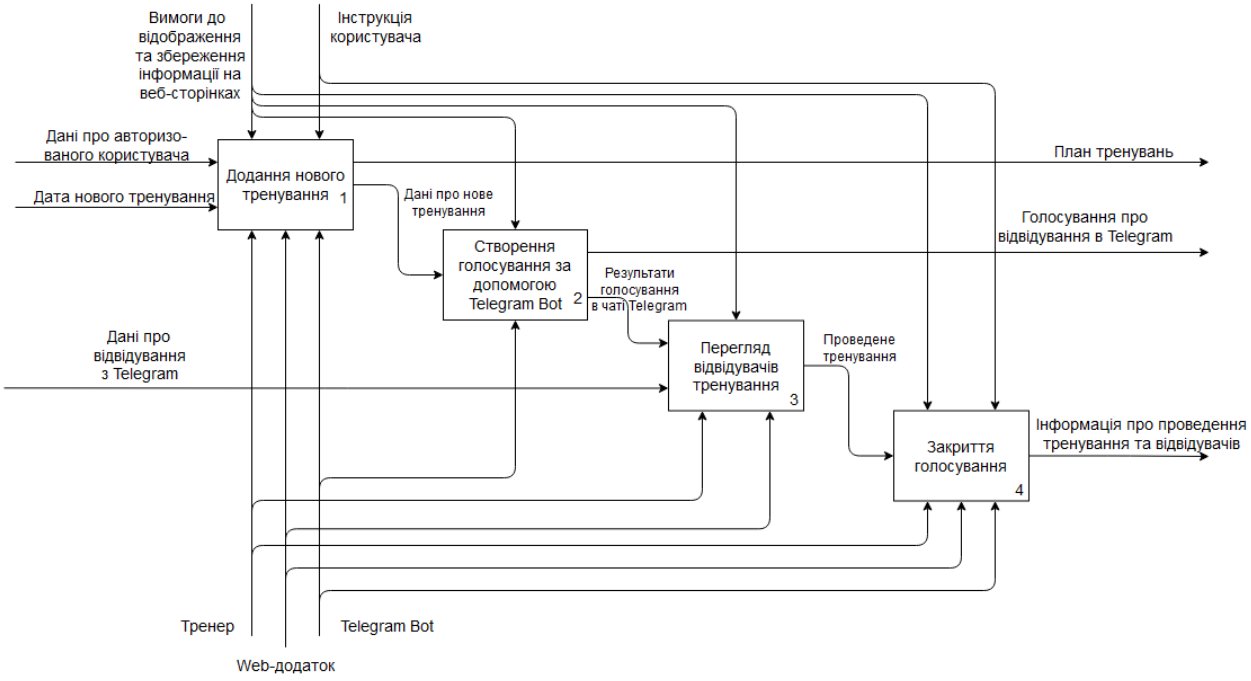


Рисунок 3.3 – Модель декомпозиції другого рівня підфункції «Менеджмент тренувань»

Рисунок 3.4 – Модель декомпозиції другого рівня підфункції «Менеджмент фінансів»

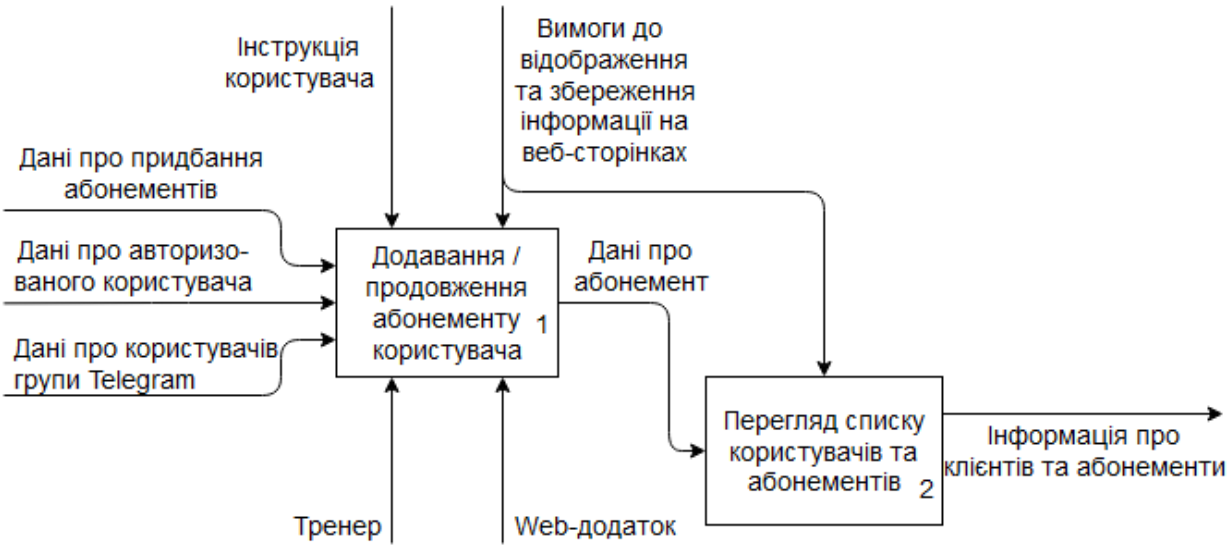


Рисунок 3.5 – Модель декомпозиції другого рівня підфункції «Менеджмент абонементів»

3.2 Моделювання варіантів використання

Для узагальнення деталей користувачів інформаційної системи, а також моделювання її функціональності в уніфікованій мові об’єктно-орієнтованого моделювання використовуються діаграма варіантів використання (Use Case). Така діаграма складається з варіантів використання, акторів, а також взаємодії між ними.

У даній інформаційній системі актором є тренер.

Артефактами системи є база даних, що зберігає інформацію, яка необхідна для функціонування системи, а також Telegram Bot, який необхідний для збору інформації про користувачів та відвідування тренувань.

До виділених варіантів використання належать:

- ВВ 1 Авторизація – авторизація у системі за допомогою віджету Telegram;
- ВВ 2 Додавання нового тренування – можливість створити нове тренування у відповідний час для деякої групи;

- ВВ 3 Додавання / продовження абонементу – створення абонементу на деяку кількість занять в деякий проміжок часу, а також можливість продовжити тривалість абонементу;
- ВВ 4 Додавання нового об’єкту витрат – можливість додати до системи деякий пункт витрат, що піде на підтримку залу;
- ВВ 5 Перегляд календаря тренувань – можливість переглянути тренування на календарі за датою та групою;
- ВВ 6 Перегляд деталей тренування – можливість переглянути деталі деякого тренування, в тому числі дату та відвідувачів;
- ВВ 7 Перегляд списку користувачів – переглянути список усіх активних тренуючихся та / або тих, хто полишив залу;
- ВВ 8 Перегляд сайту-візитки – можливість переглянути головну сторінку сайту на якій відображена інформація про зал, тренера, а також деякі.

Діаграма варіантів використання відображена на рисунку 3.6.

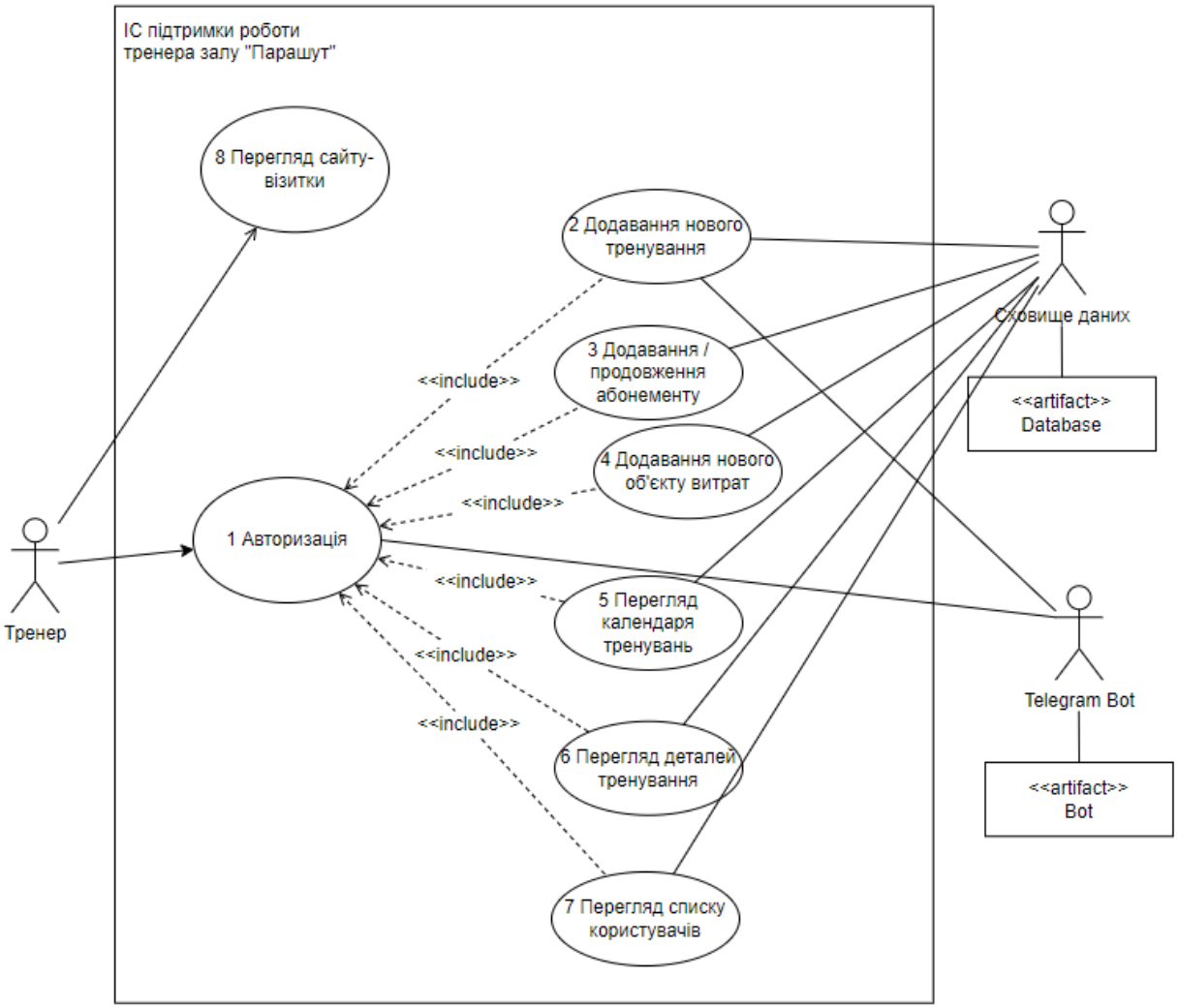


Рисунок 3.6 – Діаграма варіантів використання

3.3 Проектування бази даних

На етапі проектування бази даних (БД) розробляється загальний план, який дозволить ефективно реалізувати БД.

Реляційна база даних – це така база, що основана на реляційній моделі даних. Вона складається з таблиць, які в свою чергу складається з рядків (кортежів) та колонок (атрибутів).

Для розробки моделі «сутність – зв'язок» (ERD), яка допомогає описати концептуальну схему предметної області, потрібно визначити список сутностей, їх атрибути та зв’язки між ними.

На етапі моделювання було визначено такі сутності:

- User – користувач системи, в тому числі адміністратори системи, тренери та клієнти залу;
- Group – група тренуючихся, для яких створюються тренування;
- MembershipType – тип абонементу, який клієнт може придбати;
- Membership – придбані клієнтами абонементи;
- TrainingSession – одне тренування;
- TrainingSessionAttendee – відвідувач тренування;
- Expense – об’єкт витрат на підтримку роботи залу.

В таблиці 3.1 наведено список атрибутів кожної таблиці та їх опис.

На рисунку 3.7 наведено модель бази даних підсистеми.

Таблиця 3.1 – Опис атрибутів таблиць

Таблиця	Атрибут	Зміст	Тип	Ключі	Обмеження
User	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null
	GroupId	Ідентифікатор групи	Varchar (30)	FK	Not null
	TelegramUserId	Ідентифікатор користувача в Telegram	Varchar (30)		Not null
	FirstName	Ім’я користувача в Telegram	Varchar (100)		Not null
	LastName	Фамілія користувача в Telegram	Varchar (100)		
	UserName	Нік користувача в Telegram	Varchar (100)		

27

Продовження таблиці 3.1

	Role	Роль користувача в системі (тренер або тренуючийся)	Varchar (10)		Not null
	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null
	IsDeleted	Чи користувач покинув групу в Telegram	Boolean		Not null
Group	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null
	Name	Назва групи	Varchar(60)		Not null
	ChatId	Унікальний ідентифікатор чату групи в Telegram	Varchar(30)		Not null

28

Продовження таблиці 3.1

	TrainerId	Тренер, який тренує групу	Varchar(30)	FK	Not Null
	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null
MembershipType	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null
	Duration	Тривалість активності абонементу	Timespan		Not Null
	NumberOfSessions	Кількість занять, що входять до абонементу	int		Not Null
	Cost	Вартість абонементу	int		Not Null

29

Продовження таблиці 3.1

	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null
Membership	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null
	UserId	Унікальний ідентифікатор користувача, який придбав абонемент	Varchar(30)	FK	Not Null
	MembershipTypeId	Тип абонементу	Varchar(30)	FK	Not Null
	StartDate	Початок активності абонементу	Date		Not Null

30

	EndDate	Закінчення роботи абонементу	Date		Not Null
	WasPaidFor	Чи клієнт вже заплатив за абонемент	Boolean		Not Null
	WasExtended	Чи абонемент було продовжено	Boolean		Not Null
	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null
TrainingSession	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null
	GroupId	Ідентифікатор групи	Varchar (30)	FK	Not null

31

Продовження таблиці 3.1

	PollId	Унікальний ідентифікатор голосування в Telegram	Varchar (30)		Not null
	Date	Дата проведення заняття	Date		Not null
	Status	Статус заняття (проведено, перенесено тощо)	Varchar (15)		Not null
	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null
TrainingSessionAttendee	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null

32

Продовження таблиці 3.1

	SessionId	Унікальний ідентифікатор тренування	Varchar(30)	FK	Not Null
	MembershipId	Ідентифікатор абонементу того, хто відвідав тренування	Varchar(30)	FK	Not Null
	DidCancel	Чи користувач сам відмінив своє відвідування	Boolean		Not Null
	DidAttend	Чи користувач з’явився на тренуванні	Boolean		
	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null

33

Продовження таблиці 3.1

Expense	Id	Унікальний ідентифікатор об’єкту	Varchar(30)	PK	Unique, not null
	TrainerId	Ідентифікатор тренера, який створив запис	Varchar(30)	FK	Not Null
	Name	Назва об’єкту витрат	Varchar(200)		Not Null
	CostPerOne	Ціна за один	Float		Not Null
	Amount	Кількість куплених об’єктів	Int		Not Null
	Notes	Додаткова інформація	Text		
	CreatedAt	Дата створення об’єкту	Date		Not Null
	UpdatedAt	Дата останнього редагування об’єкту	Date		Not Null

34

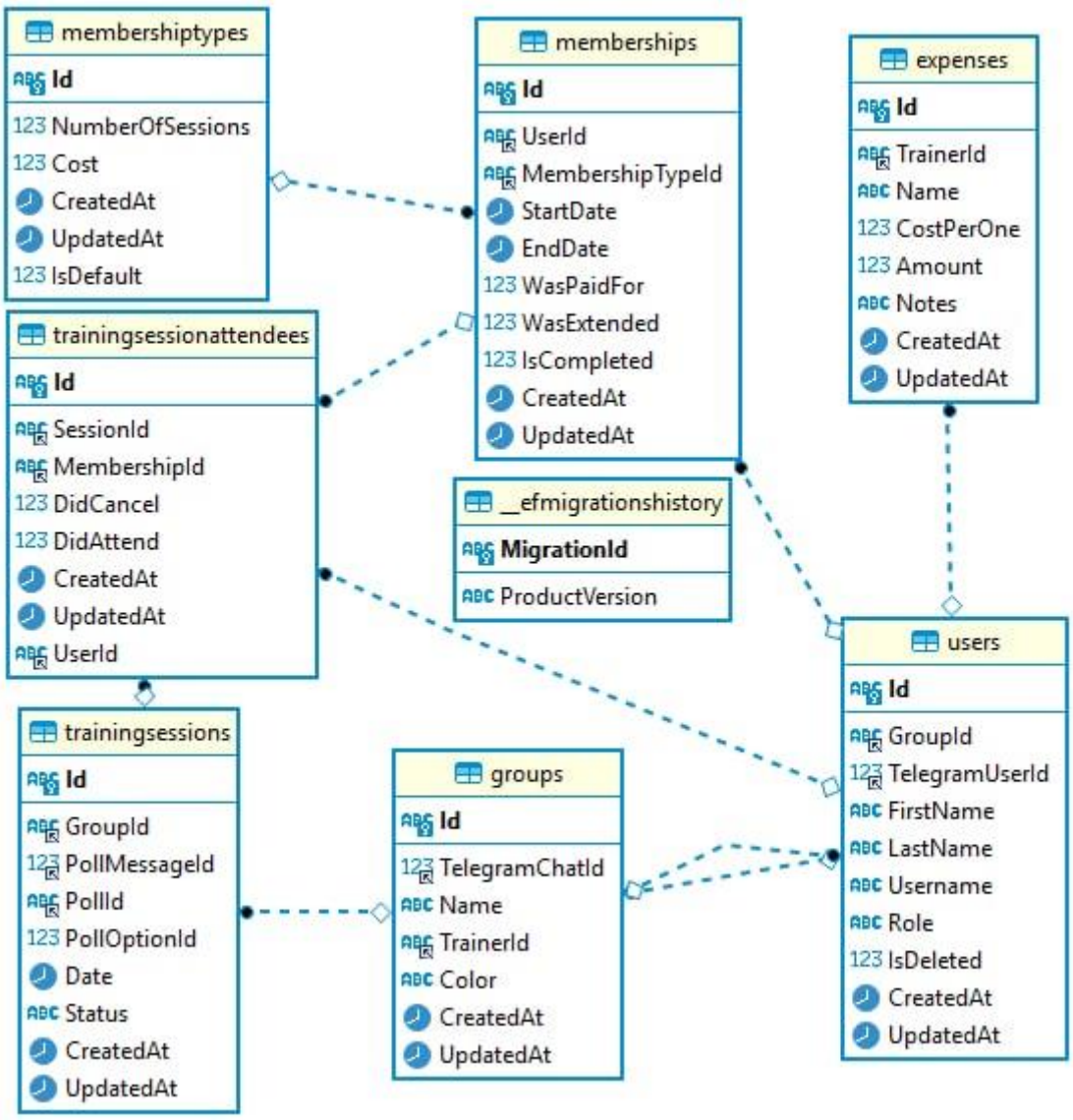


Рисунок 3.7 – Модель бази даних

4 РОЗРОБКА ІНФОРМАЦІЙНОЇ ПІДСИСТЕМИ

4.1 Реалізація бази даних

Для повноцінного функціонування розроблюваного проекту проводимо реалізацію бази даних. До файлу appsettings,json додаємо запис, що дозволить додатку з'єднуватись із базою даних. Перший крок відображено на рисунку 4.1.

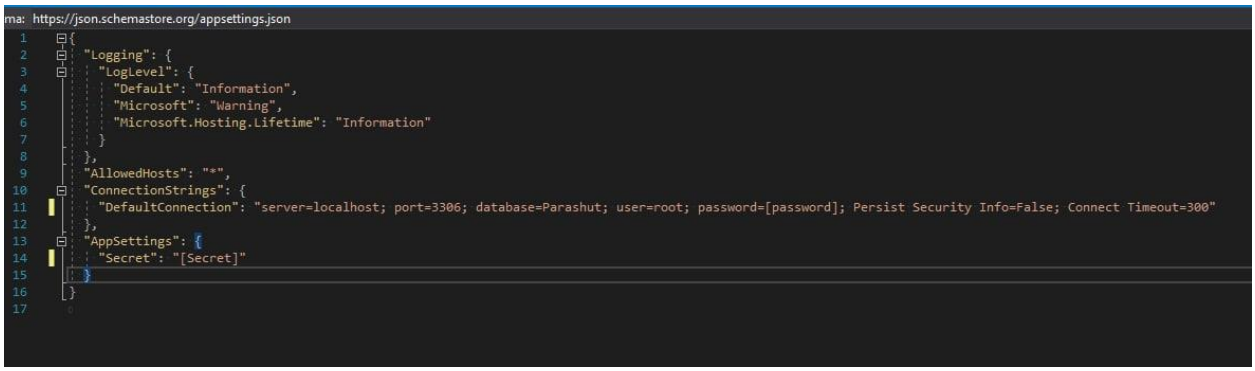


Рисунок 4.1 – Модифікація файлу appsettings.json

Для опису структури бази даних створимо відповідний файл ParashutDbContext.cs і приєднаємо його до проекту. Наступний рисунок 4.2 відображає створення таблиць у базі даних.

```
public DbSet<User> Users { get; set; }
public DbSet<Group> Groups { get; set; }
public DbSet<MembershipType> MembershipTypes { get; set; }
public DbSet<Membership> Memberships { get; set; }
public DbSet<TrainingSession> TrainingSessions { get; set; }
public DbSet<TrainingSessionAttendee> TrainingSessionAttendees { get; set; }
public DbSet<Expense> Expenses { get; set; }
public DbSet <UserStats> UserStats { get; set; }
```

Рисунок 4.2 – Створення таблиць у базі даних

Для конфігурації кожної із таблиць викликається відповідний метод та застосовується конфігурація.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.ApplyConfiguration(new UserConfiguration());
    modelBuilder.ApplyConfiguration(new GroupConfiguration());
    modelBuilder.ApplyConfiguration(new MembershipTypeConfiguration());
    modelBuilder.ApplyConfiguration(new MembershipConfiguration());
    modelBuilder.ApplyConfiguration(new TrainingSessionConfiguration());
    modelBuilder.ApplyConfiguration(new TrainingSessionAttendeeConfiguration());
    modelBuilder.ApplyConfiguration(new ExpenseConfiguration());
    modelBuilder.ApplyConfiguration(new UserStatsConfig());
}
```

Рисунок 4.3 – Метод конфігурації таблиць

Розглянемо на прикладі таблиці користувачів загальний принцип роботи. Для реалізації кожної конфігурації використовується окремий клас, у даному випадку UserConfiguration, що наслідує базовий клас BaseEntityConfiguration. У базовому класі описані налаштування що застосовуються до кожної моделі у базі даних: унікальний номер ідентифікації, дата створення запису і дата останнього оновлення запису.

```
namespace Parashut.Api.Db.ModelConfigurations
{
    public class BaseEntityConfiguration<TEntity> : IEntityTypeConfiguration<TEntity>
        where TEntity : BaseEntity
    {
        public virtual void Configure(EntityTypeBuilder<TEntity> builder)
        {
            builder.HasKey(_ => _.Id);

            builder.Property(_ => _.Id).HasColumnType("CHAR(36)");

            builder.Property(_ => _.CreatedAt)
                .HasColumnType("datetime")
                .HasDefaultValueSql("now()")
                .IsRequired();

            builder.Property(_ => _.UpdatedAt)
                .HasColumnType("datetime")
                .HasDefaultValueSql("now()")
                .IsRequired();
        }
    }
}
```

Рисунок 4.4 – Створення класу Базової сутності

Спочатку вкажемо, що додавання даних відбуватиметься до таблиці користувачів. Встановимо індекси та додамо відповідні колонки до бази даних: ідентифікатор користувача, його телеграм-ідентифікатор, ім’я та прізвище тощо. Для кожного поля встановимо відповідний тип даних, а також чи є це поле обов’язковим. Останнім рядком встановимо зв’язки з іншими таблицями. Дані модифікації ілюструє рисунок 4.5.

```
// Map entities to tables
builder.ToTable("Users");

// Configure indexes
builder.HasIndex(_ => _.TelegramUserId).HasDatabaseName("Idx_TelegramUserId");

// Configure columns
builder.Property(_ => _.GroupId).HasColumnType("CHAR(36)");
builder.Property(_ => _.TelegramUserId).HasColumnType("BIGINT").IsRequired();
builder.Property(_ => _.FirstName).HasColumnType("CHAR(64)").IsRequired();
builder.Property(_ => _.LastName).HasColumnType("CHAR(64)");
builder.Property(_ => _.Username).HasColumnType("CHAR(32)");
builder.Property(_ => _.Role).HasColumnType("CHAR(10)").HasConversion<string>().HasDefaultValue(Role.Trainee).IsRequired();
builder.Property(_ => _.IsDeleted).HasDefaultValue(false).IsRequired();

builder.HasOne(u => u.Group).WithMany(g => g.Trainees).HasForeignKey(u => u.GroupId);
```

Рисунок 4.5 – Налаштування полів бази даних

Наступним кроком передаємо початкові записи до бази даних на рисунку 4.6, щоб проводити тестування в подальшому.

```
List <User> userList = new()
{
    new User
    {
        Id = new Guid("9d9f2bbb-2f51-40bb-ad55-641c9b612eec"),
        TelegramUserId = 344923873,
        Role = Role.Trainer,
        FirstName = "Anastasia",
        LastName = "Shadow",
        Username = "AnastasiaShadow",
    },
    new User
    {
        Id = new Guid("fef481a8-9992-4eb7-9971-15b4efe7c413"),
        TelegramUserId = 141557617,
        Role = Role.Trainee,
        FirstName = "Alisa",
        LastName = "Rieznikova",
        Username = "alicegrnwd",
    },
    new User
    {
        Id = new Guid("fb750073-836e-4da2-92de-483cb7229bcc"),
        TelegramUserId = 987547656,
        Role = Role.Trainee,
        FirstName = "Светлана",
        LastName = "Глущенко",
    },
};
builder.HasData(userList);
```

Рисунок 4.6 – Додавання тестових записів до таблиці

4.2 Створення загальної структури клієнтської частини

Роботу над клієнтською частиною починаємо зі створення проекту за допомогою команди create-react-app. React є однією з найпопулярніших бібліотек JavaScript, яка використовується для розробки інтерфейсу. Завдяки підходу на основі компонентів розробка додатків є легкою та швидшою. Проте React не надає жодного способу прямого зв'язку між компонентами. Саме цю проблему вирішує Redux. Він надає «сховище», в якому зберігається поточний стан системи, та до якого можна отримати доступ з любого місця додатку.

Отже, почнемо роботу з реалізації сховища в проекті. Загальна структура наведена на рисунку 4.7.

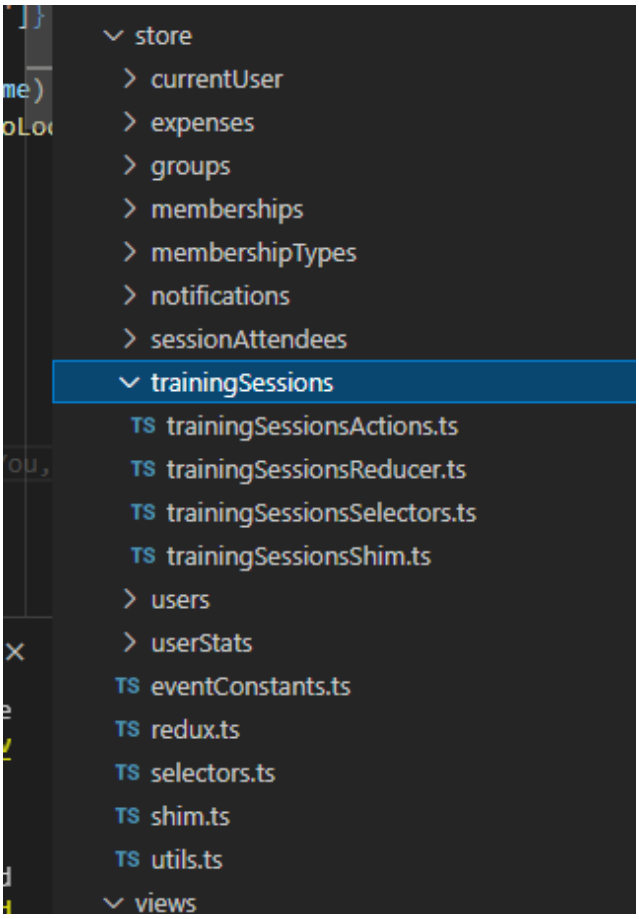


Рисунок 4.7 – Структура сховища проекту

Опишемо вміст редюсерів на основі trainingSessions, що відображено на рисунку 4.8.

Файл trainingSessionsShim містить в собі методи для які викликають API, а також проводять першочергову перевірку результату.

```
export const createTrainingSession = async (
  groupId: string,
  dates: Date[],
  textMessage: string,
) => {
  try {
    const response = await axios.post(
      `${ApiConstants.backendUrl}${ApiConstants.trainingSessions}`,
      { groupId, dates: dates.map((d) => d.getTime() / 1000), textMessage },
    )
    return response.data
  } catch (error) {
    await handleError(error)
    return {}
  }
}

export const cancelTrainingSession = async (sessionId: string) => {
  try {
    const response = await axios.put(
      `${ApiConstants.backendUrl}${ApiConstants.cancelTrainingSession.replace(
        '[id]',
        sessionId,
      )}`,
    )
    return response.data
  } catch (error) {
    await handleError(error)
    return {}
  }
}
```

Рисунок 4.8 – Методи виклику API для тренувань

Ці методи викликаються за допомогою дій (action). Дія – це простий об’єкт JavaScript, який містить поле type. Дії, загалом, розглядають як події, якої описують, що щось сталося в програмі. Типовий action має такий вигляд (рис. 4.9).

```
export const getAllSessionsForUser = () => {
  return async (dispatch: Dispatch, getState: Function) => {
    const result = await getAllSessionsForUserRequest()

    dispatch({
      type: types.GET_ALL_SESSIONS_FOR_USER,
      payload: result,
    })
  }
}
```

Рисунок 4.9 – Методи виклику API для тренувань

Редуктор (reducer) – це функція, яка отримує поточний стан і об’єкт дії, вирішує, як оновити стан, якщо необхідно, і повертає новий стан. Редуктор можна розглядати як прослуховувач подій, який обробляє їх на основі отриманого типу. Приклад редюсеру наведено на рис 4.10.

```
export interface ITrainingSessionsStore {
  sessions: Map<string, ITrainingSession>
  isFetching: boolean
}

const defaultState = {
  sessions: new Map<string, ITrainingSession>(),
  isFetching: true,
} as ITrainingSessionsStore
💡
const trainingSessionsReducer = (state = defaultState, action: any) => {
  switch (action.type) {
    case types.GET_ALL_SESSIONS_FOR_USER: {
      return {
        sessions: toMap<ITrainingSession>(action.payload),
        isFetching: false,
      }
    }

    case types.TRAINING_SESSIONS_RECEIVED: {
      return {
        ...state,
        sessions: copyAndAddArray<ITrainingSession>(
          state.sessions,
          action.payload,
        ),
      }
    }

    case types.TRAINING_SESSION_UPDATED: {
      return {
        ...state,
        sessions: copyAndUpdateItem<ITrainingSession>(
          state.sessions,
          action.payload,
        ),
      }
    }

    default:
      return state
  }
```

Рисунок 4.10 – Приклад редуктору

Останнім файлом в папці є trainingSessionsSelectors (рис.4.11). Селектори (selectors) – це функції, які знають, як витягувати конкретну інформацію зі стану. Вони допомогають уникнути повторення логіки, оскільки різні частини програми повинні читати ті самі дані. В прикладі наведене створення чотирьох селекторів для тренувань. В проекті використовується npm-пакет «reselect». Це є бібліотека для створення запам'ятовуваних селекторів. Вона значно підвищує їх швидкодію.

```
import { Shim } from '../shim'
import { createSelector } from 'reselect'
import { ITrainingSessionsStore } from './trainingSessionsReducer'

const trainingSessionsState = (
  state: Partial<Shim>,
): ITrainingSessionsStore => {
  const { trainingSessionsReducer = {} as ITrainingSessionsStore } = state || {}
  return trainingSessionsReducer
}

const getSessionsMap = createSelector(trainingSessionsState, (state) => {
  return state.sessions
})

const getSessionsList = createSelector(trainingSessionsState, (state) => {
  return Array.from(state.sessions.values())
})

const getIsFetching = createSelector(trainingSessionsState, (state) => {
  return state.isFetching
})

const trainingSessionsSelectors = {
  trainingSessionsState,
  getSessionsMap,
  getSessionsList,
  getIsFetching,
}

export default trainingSessionsSelectors
```

Рисунок 4.11 – Файл trainingSessionsSelectors

Сховище (рис. 4.12) створюється при передачі до нього редукторів. Для того, аби можна було відправляти асинхронні дії, було також застосовано middleware «thunk».

```
const appReducer = combineReducers({
  currentUserReducer,
  usersReducer,
  groupsReducer,
  trainingSessionsReducer,
  sessionAttendeesReducer,
  notificationReducer,
  membershipsReducer,
  membershipTypesReducer,
  userStatsReducer,
  expensesReducer,
})

const rootReducer = (state: any, action: any) => {
  if (action.type === types.LOGOUT) {
    state = undefined
  }

  return appReducer(state, action)
}

const composeEnhancers = composeWithDevTools({
  serialize: true,
})

const store = createStore(
  rootReducer,
  composeEnhancers(applyMiddleware(thunk), applyMiddleware(signalRMiddleware)),
)

export default store
```

Рисунок 4.12 – Створення сховища

Також для можливості отримання повідомлень у реальному часі було створено власний middleware (4.13) за допомогою бібліотеки signalR. Під час входу в акаунт, користувач буде підключатись до групи, що буде слухати повідомлення, які відправляються з API.

```
  const connectionHub = `${ApiConstants.backendUrl}${ApiConstants.sessionsHub}`

  // create the connection instance
  if (!connection) {
    connection = new HubConnectionBuilder()
      .withUrl(connectionHub)
      .configureLogging(LogLevel.Information)
      .build()

    connection.on(hubEventConstants.GROUP_CREATE, (arg: string) => {···
    })

    connection.on(hubEventConstants.USERS_CREATED, (arg: string) => {···
    })

    connection.on(hubEventConstants.EXPENSE_CREATED, (arg: string) => {···
    })

    connection.on(···
    )

    connection.on(···
    )

    connection.on(···
    )

    connection.on(···
    )

    connection.on(···
    )
  }

  connection.start().then(() => {
    console.info('SignalR Connected')
    const scriptId = action.payload
    connection
      ?.invoke('SubscribeToTrainingSessionsUpdate', scriptId)
      .catch((err: Error) => {
        return console.error(err.toString())
      })
      .catch((err) => console.error('SignalR Connection Error: ', err))
  })
  break
}
case types.DISCONNECT_FROM_TRAINER_HUB:
  connection?.stop()
  break
default:
  break
```

Рисунок 4.13 – SignalR middleware

4.3 Реалізація кабінету тренера

Під час входу користувача на сайт відбувається отримання усіх даних, що йому необхідні. Для реалізації функціоналу контролю за абонементами,

тренуваннями, користувачами тощо було реалізовано відповідні сервіси. Наприклад, за допомогою MembershipService.cs відбувається керування такими процесами як створення нового абонементу, збереження історії абонементів, отримання актуальної інформації про поточний використовуваний абонемент. Сервіс TrainingSessionService.cs відповідає за створення нових тренувань, отримує інформацію із Телеграм-голосувань щодо відвідання певним користувачем певного тренування, передає дані про відміну тренування тощо.

Таким чином, викликавши метод GetAllMembershipsForUser, що наведений на рисунку 4.14, тренер отримає усі абонементи своїх клієнтів, а тренуючийся отримує інформацію лише про ті, що належать йому. Таким же чином були отримані дані про тренування, користувачів, тощо.

```
public async Task<Result<List<Membership>>> GetAllMembershipsForUser()
{
    try
    {
        Entities.User user = httpContextAccessor.HttpContext.Items["User"] as User;

        if(user.Role == Role.Trainer)
        {
            List<Guid> groupIds = (await groupService.GetAllGroupIdsForTrainer(user.Id)).Item;
            List<Guid> userIds = await dbContext.Users
                .Where(_ => _.GroupId.HasValue && groupIds.Contains((Guid)_.GroupId))
                .Select(_ => _.Id)
                .ToListAsync();

            List<Membership> memberships = await dbContext.Memberships
                .Where(_ => userIds.Contains(_.UserId))
                .ToListAsync();

            return Result<List<Membership>>.Success(memberships);
        }
        else
        {
            List<Membership> memberships = await dbContext.Memberships
                .Where(_ => _.UserId == user.Id)
                .ToListAsync();

            return Result<List<Membership>>.Success(memberships);
        }
    }
    catch (Exception ex)
    {
        string errorMessage = "Unexpected error on getting all memberships.";
        logger.LogError(ex, errorMessage);
        return Result<List<Membership>>.Fail(StatusCodes.Status500InternalServerError, errorMessage);
    }
}
```

Рисунок 4.14 – Компонент календаря тренувань

Перейдемо до реалізації кабінету користувача. Головною сторінкою для тренера буде виступати календар тренувань. Тут будуть відображатись усі тренування, до яких тренер матиме доступ. Для його реалізації було обрано бібліотеку «fullcalendar» (рис. 4.15) з плагінами daygrid, interaction, timegrid. Вона надає широкий та гнучкий функціонал, що дозволяє швидко створити необхідного вигляду календар.

```
42      return (
43        <>
44          {clickedDate && (
45            <SessionCreationModal
46              closeModal={this.closeCreationModal}
47              date={clickedDate}
48            />
49          )}
50          {selectedSession && (
51            <SessionDetailsModal
52              closeModal={this.closeDetailsModal}
53              session={selectedSession}
54            />
55          )}
56          <div className={classes.calendarWrapper}>
57            <FullCalendar
58              plugins={[dayGridPlugin, interactionPlugin, timeGridPlugin]}
59              headerToolbar={{
60                left: 'prev,next today',
61                center: 'title',
62                right: 'dayGridMonth,timeGridWeek,timeGridDay',
63              }}
64              initialView="dayGridMonth"
65              events={this.calendarEvents}
66              editable={true}
67              selectable={false}
68              selectMirror={false}
69              droppable={false}
70              eventClick={this.openDetailsModal}
71              dateClick={this.openCreationModal}
72              dayMaxEvents={4}
73              eventContent={this.renderEventContent}
74              eventTimeFormat={timeFormat}
75              locale={uaLocale}
76            />
77          </div>
78        </>
79      )
80    }
```

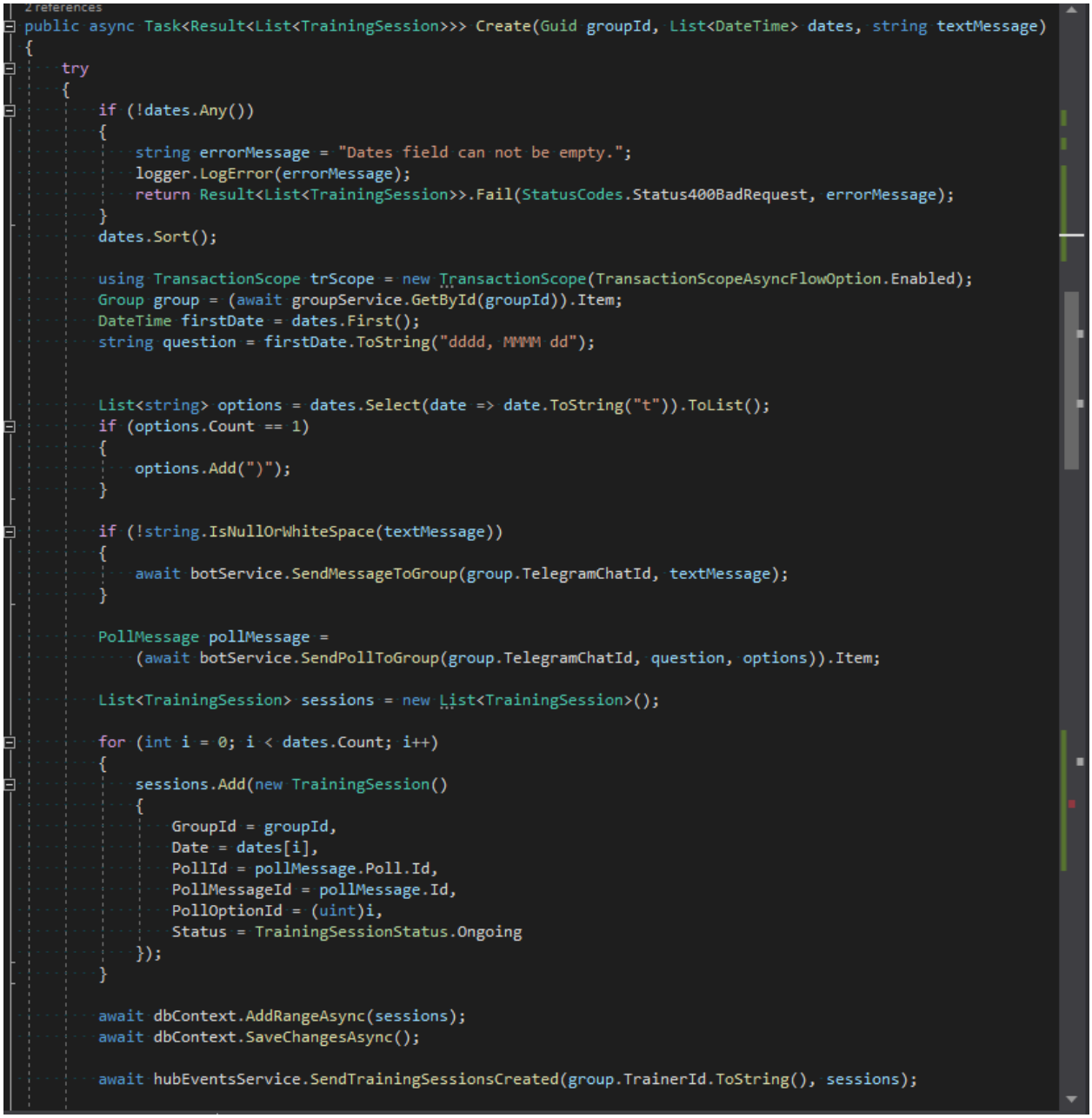
Рисунок 4.15 – Компонент календаря тренувань

При натисканні на існуючий івент буде виведений діалог з деталями про тренування, де можна буде відмінити тренування або завершити його.

При натисканні на дату в календарі буде виведений діалог (рис 4.16) де можна створити нове тренування (4.17). Цей процес проводиться використовуючи транзакцію з метою не залишати сміттєвої інформації у базі в разі виникнення помилки. За допомогою TelegramBotService`у створюється голосування у чаті відповідної групи. Потім зберігається нове тренування у базі та відправляється клієнтами, які слухають ці оновлення.

```
<Dialog
  open={true}
  onClose={closeModal}
  aria-labelledby="customized-dialog-title"
  aria-describedby="modal-modal-description"
  fullWidth
>
  <CustomDialogTitle id="customized-dialog-title" onClose={closeModal}>
    Створення тренування {date.toDateString()}
  </CustomDialogTitle>
  <CustomDialogContent dividers>
    <InputLabel id="select-label" required>
      Оберіть групу
    </InputLabel>
    <Select
      labelId="select-label"
      value={selectedGroupId}
      onChange={this.handleGroupChange}
      fullWidth
      variant="outlined"
    >
      {groups?.map((group, i) => {
        return (
          <MenuItem key={i} value={group.id}>
            {group.name}
          </MenuItem>
        )
      })}
    </Select>
    <TextField
      id="filled-textarea"
      label="Повідомлення Telegram (необов'язкове)"
      placeholder="Placeholder"
      multiline
      fullWidth
      rows={2}
      margin="normal"
      variant="outlined"
      value={textMessage}
      onChange={this.handleTextChange}
    />
    <MuiPickersUtilsProvider utils={DateFnsUtils}>
      <Grid
        container
        direction="column"
        justifyContent="space-around"
        alignItems="flex-start"
      >
        {selectedTimes.map((time, i) => (
```

Рисунок 4.16 – Компонент діалогу створення тренування



```
2 references
public async Task<Result<List<TrainingSession>>> Create(Guid groupId, List<DateTime> dates, string textMessage)
{
    try
    {
        if (!dates.Any())
        {
            string errorMessage = "Dates field can not be empty.";
            logger.LogError(errorMessage);
            return Result<List<TrainingSession>>.Fail(StatusCodes.Status400BadRequest, errorMessage);
        }
        dates.Sort();

        using TransactionScope trScope = new TransactionScope(TransactionScopeAsyncFlowOption.Enabled);
        Group group = (await groupService.GetById(groupId)).Item;
        DateTime firstDate = dates.First();
        string question = firstDate.ToString("dddd, MMMM dd");


        List<string> options = dates.Select(date => date.ToString("t")).ToList();
        if (options.Count == 1)
        {
            options.Add(")");
        }

        if (!string.IsNullOrWhiteSpace(textMessage))
        {
            await botService.SendMessageToGroup(group.TelegramChatId, textMessage);
        }

        PollMessage pollMessage =
            (await botService.SendPollToGroup(group.TelegramChatId, question, options)).Item;

        List<TrainingSession> sessions = new List<TrainingSession>();

        for (int i = 0; i < dates.Count; i++)
        {
            sessions.Add(new TrainingSession()
            {
                GroupId = groupId,
                Date = dates[i],
                PollId = pollMessage.Poll.Id,
                PollMessageId = pollMessage.Id,
                PollOptionId = (uint)i,
                Status = TrainingSessionStatus.Ongoing
            });
        }

        await dbContext.AddRangeAsync(sessions);
        await dbContext.SaveChangesAsync();

        await hubEventsService.SendTrainingSessionsCreated(group.TrainerId.ToString(), sessions);
```

Рисунок 4.17 – Код створення нового тренування

На сторінці клієнтів тренер може подивитись окремо інформацію про користувачів, групи та абонементи. Вибір відбувається за допомогою вкладок (рис. 4.18).

```
private renderContent() {
  const { classes } = this.props
  const { activeTab } = this.state
  return (
    <div className={classes.root}>
      <AppBar position="static">
        <Tabs
          value={activeTab}
          onChange={this.handleChange}
          aria-label="simple tabs example"
          centered
        >
          <Tab className={classes.statsTab} label="Групи" {...a11yProps(0)} />
          <Tab
            className={classes.statsTab}
            label="Клієнти"
            {...a11yProps(1)}
          />
          <Tab
            className={classes.statsTab}
            label="Абонементи"
            {...a11yProps(2)}
          />
        </Tabs>
      </AppBar>
      <TabPanel value={activeTab} index={0}>
        <GroupsList />
      </TabPanel>
      <TabPanel value={activeTab} index={1}>
        <ClientsList />
      </TabPanel>
      <TabPanel value={activeTab} index={2}>
        <MembershipsList />
      </TabPanel>
    </div>
  )
}
```

Рисунок 4.18 – Компонент ClientPage

Компонент, що відображає список груп відображений на рис. 4.19.

```
render() {
  const { classes, groups } = this.props
  const { selectedGroup } = this.state
  return (
    <>
      {selectedGroup && (
        <GroupDetailsModal
          closeModal={this.closeDetailsModal}
          group={selectedGroup}
        />
      )}
      <Table stickyHeader size="small">
        <TableHead>
          <TableRow>
            <TableCell>№</TableCell>
            <TableCell>Назва</TableCell>
            <TableCell align="right">Кількість учасників</TableCell>
            <TableCell align="right">Дата створення</TableCell>
          </TableRow>
        </TableHead>
        <TableBody className={classes.tableBody}>
          {groups.map((group, i) => {
            return (
              <TableRow
                key={i}
                onClick={(e) => {
                  this.openDetailsModal(group)
                }}
                className={classes.tableRow}
              >
                <TableCell>{i + 1}</TableCell>
                <TableCell>{group.name}</TableCell>
                <TableCell align="center">
                  {this.getGroupUserCount(group.id)}
                </TableCell>
                <TableCell align="right">
                  {group.createdAt &&
                    new Date(group.createdAt * 1000).toLocaleString('uk-UA')}
                </TableCell>
              </TableRow>
            )
          })}
        </TableBody>
      </Table>
    </>
  )
}
```

Рисунок 4.19 – Компонент GroupsList

4.4 Приклади роботи програми

Для зручного менеджменту тренувань було розроблено календар, на якому тренер має можливість не тільки побачити заплановані тренування на поточний місяць, а і одразу побачити скільки людей записалося на сесію, додати нове тренування або видалити його за необхідністю. Приклад його відображення на рисунку 4.20.

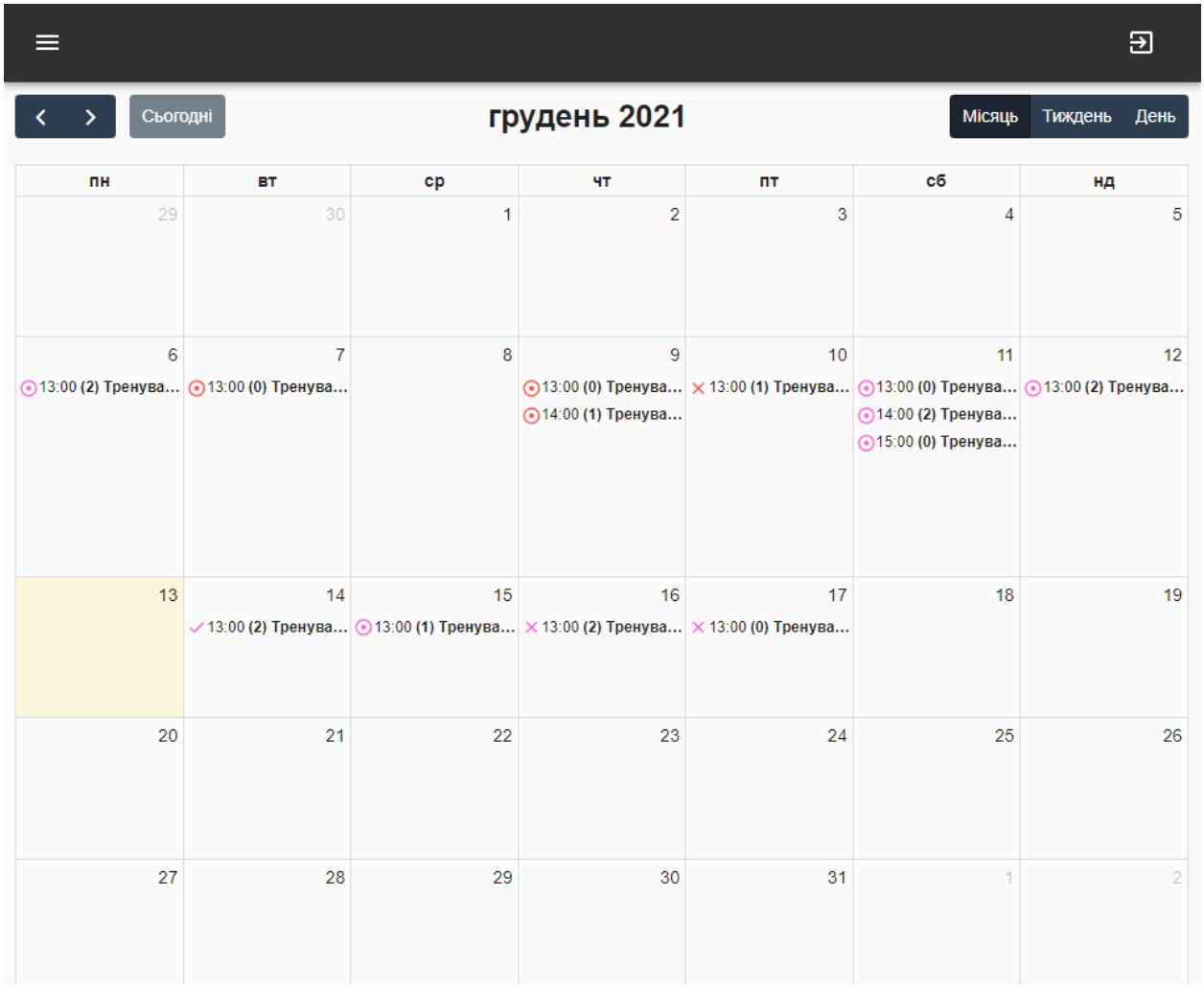


Рисунок 4.20 – Загальний вигляд календаря тренера.

Натиснувши на певний запис як на рисунку 4.21, з’являється діалогове вікно із відображенням даних про тренування: дата і час, група що відвідує заняття, чи є запис актуальним і подробиці про голосування відвідувачів.

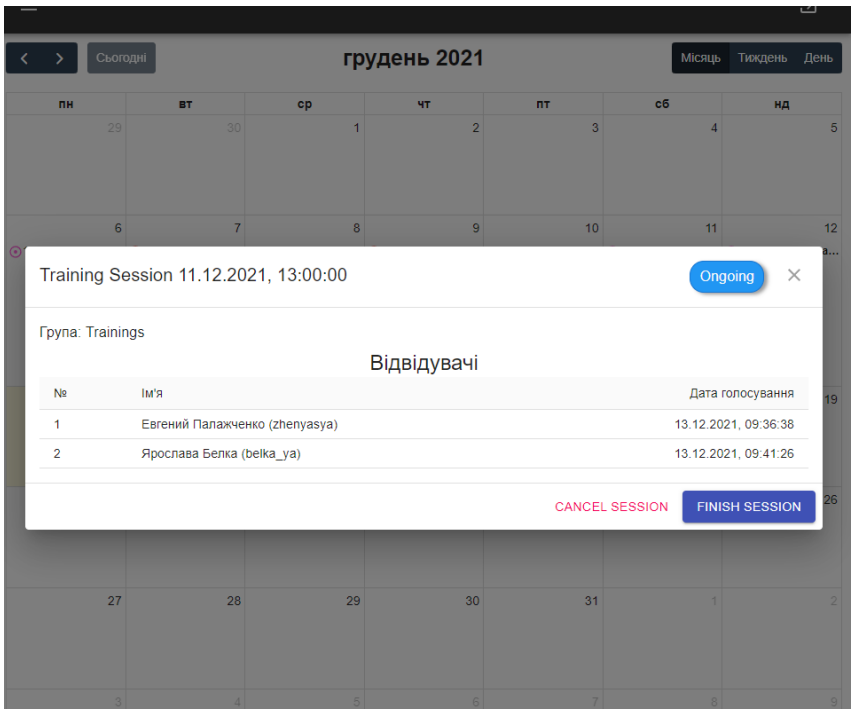


Рисунок 4.21 – Діалогове вікно із інформацією про тренування.

Тренер має можливість відмінити сесію, встановити флаг що означає завершення набору у групу, а також відредагувати список тих, хто відвідав заняття. Цей функціонал зображено на рисунках 4.22 та 4.23.

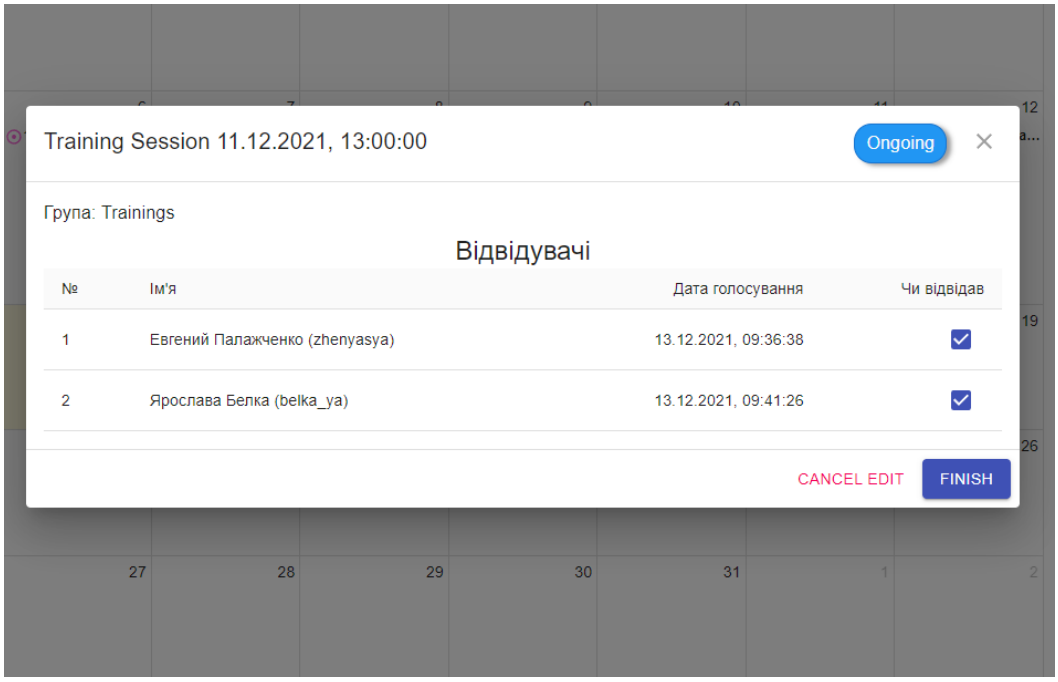


Рисунок 4.22 – Діалогове вікно редагування інформації про тренування.

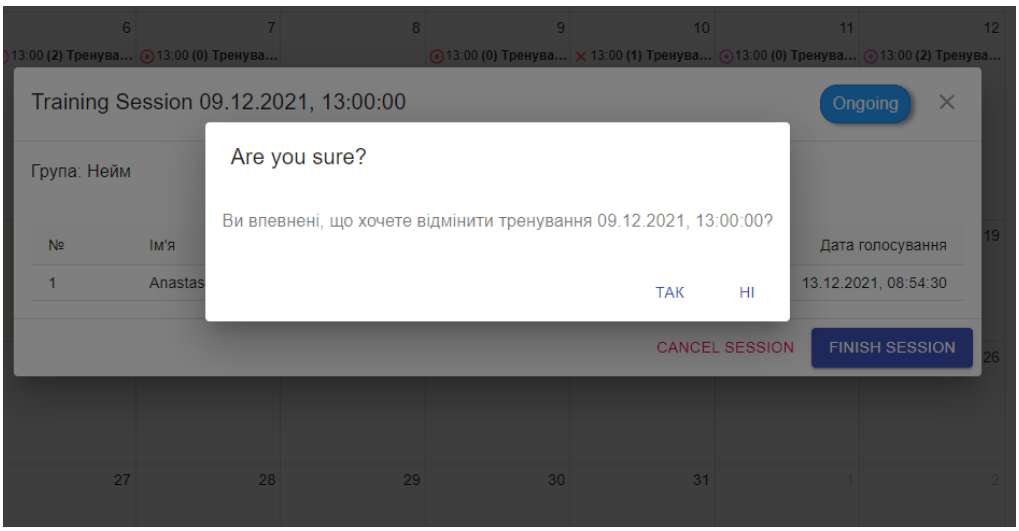


Рисунок 4.23 – Видалення запису про тренування.

Важливою частиною втіленого функціоналу є можливість додавати нові тренування: тренер має можливість встановити групу, час тренування, а також написати додаткове повідомлення для користувачів, що передається за допомогою боту до групи. Вищеописані дії відображені на рисунках 4.24 та 4.25.

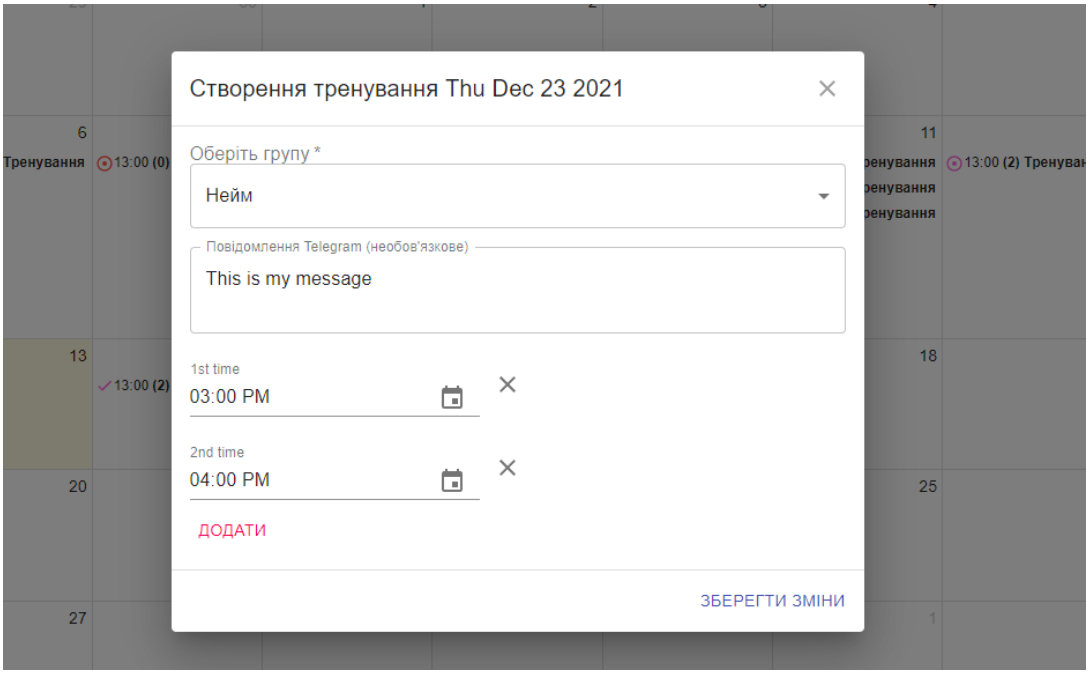


Рисунок 4.24 – Додавання нового тренування.

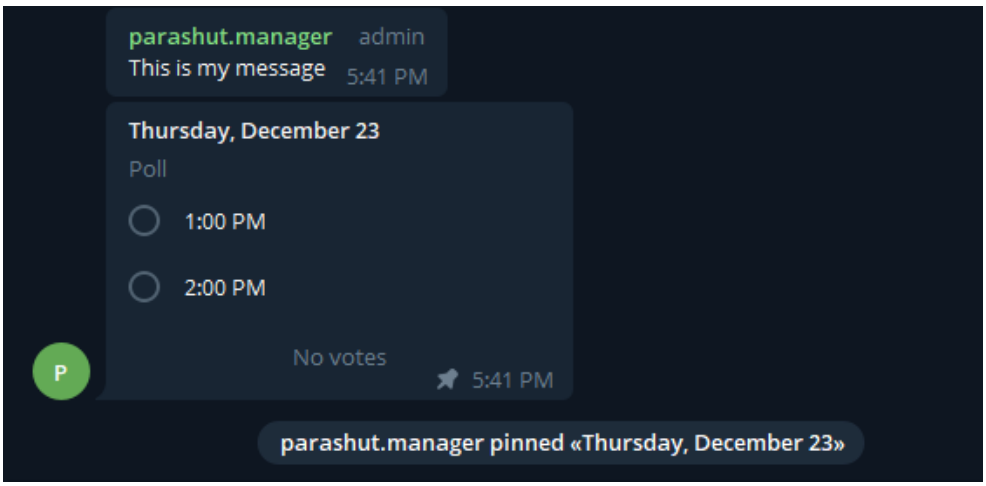


Рисунок 4.25 – Голосування про відвідування тренування у чаті.

Тренер має можливість переглянути групи відвідувачів, а також дати їх створення та кількість відвідувачів у них як на рисунках 4.26-4.29.

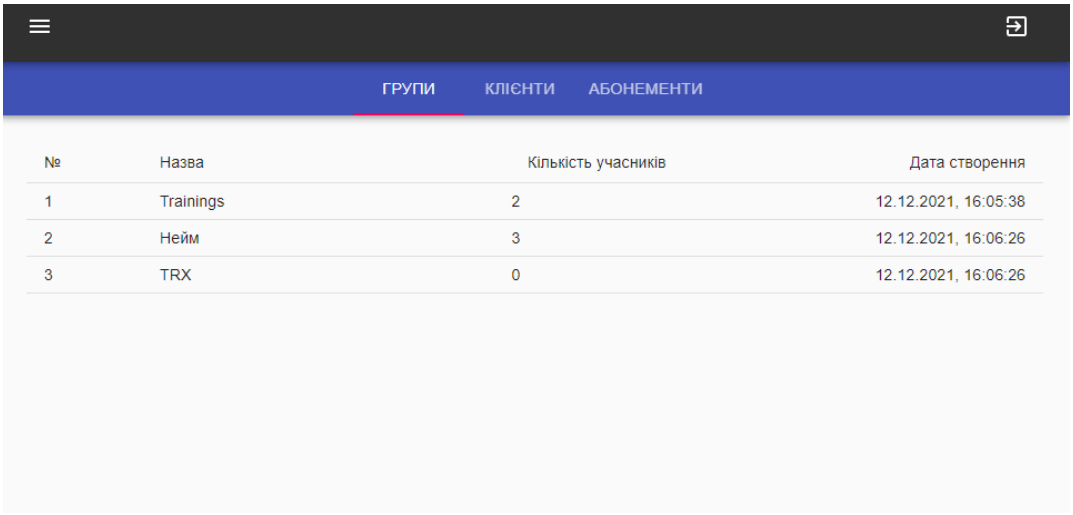


Рисунок 4.26 – Список груп відвідувачів.

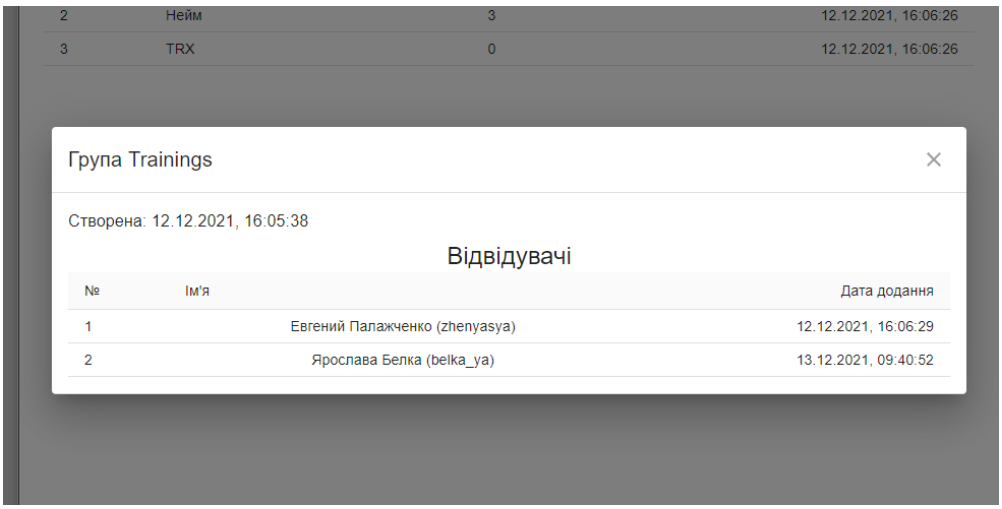


Рисунок 4.27 – Детальна інформація про группу.

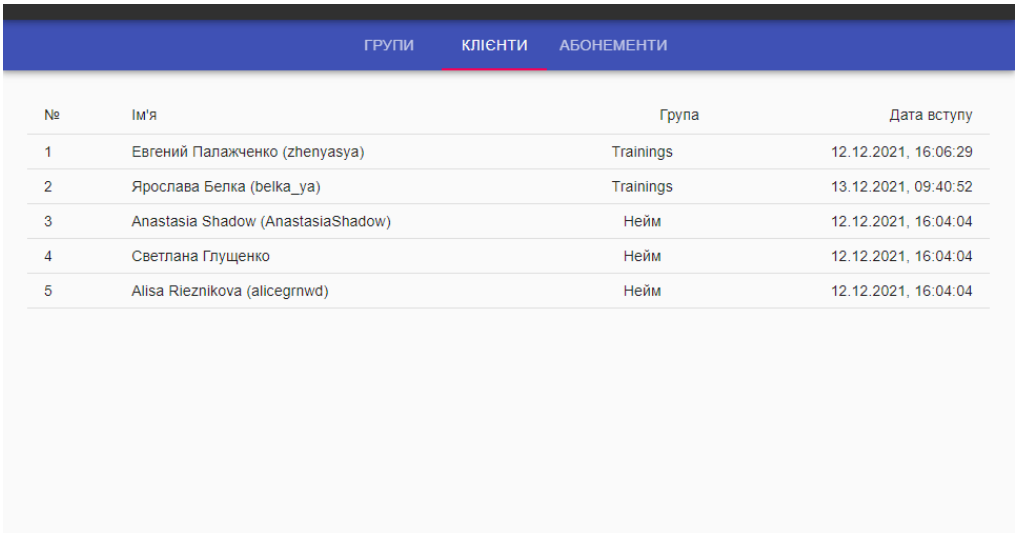


Рисунок 4.28 – Загальний список клієнтів

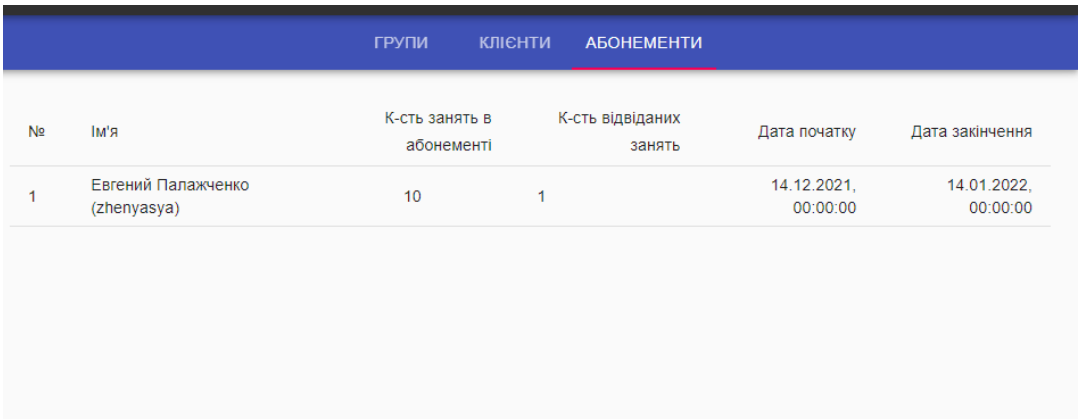


Рисунок 4.29 – Інформація про абонементи

Важливим аспектом ведення бізнесу є документування витрат на тренувальну залу: завдяки розробленій частині підсистеми тренер має можливість вести список витрат, і у майбутньому прогнозувати зміни цін на абонементи. Менеджмент витрат відображено на рисунках 4.30 та 4.31.

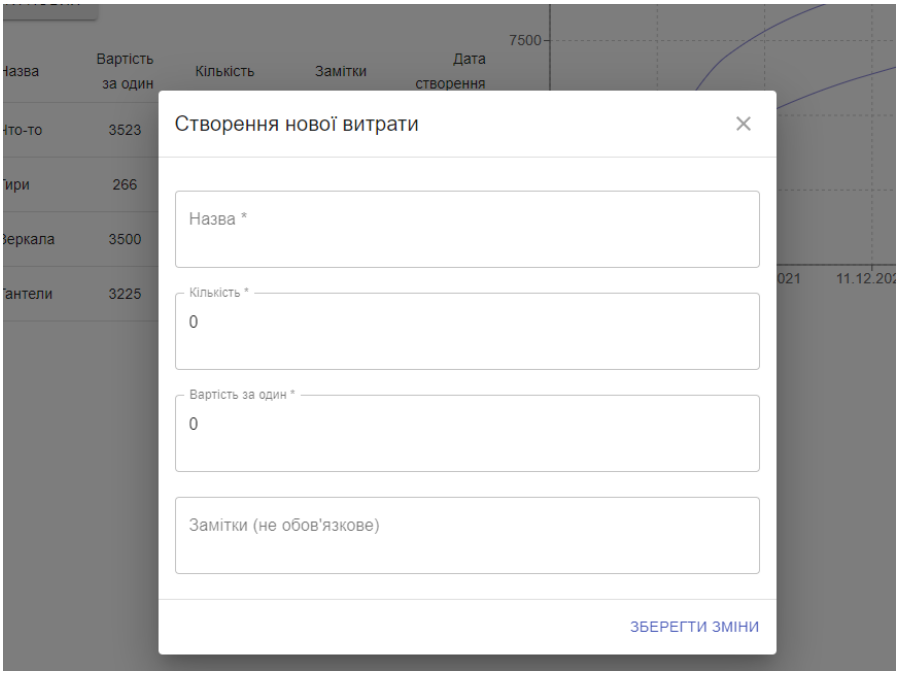


Рисунок 4.30 – Додавання нової витрати

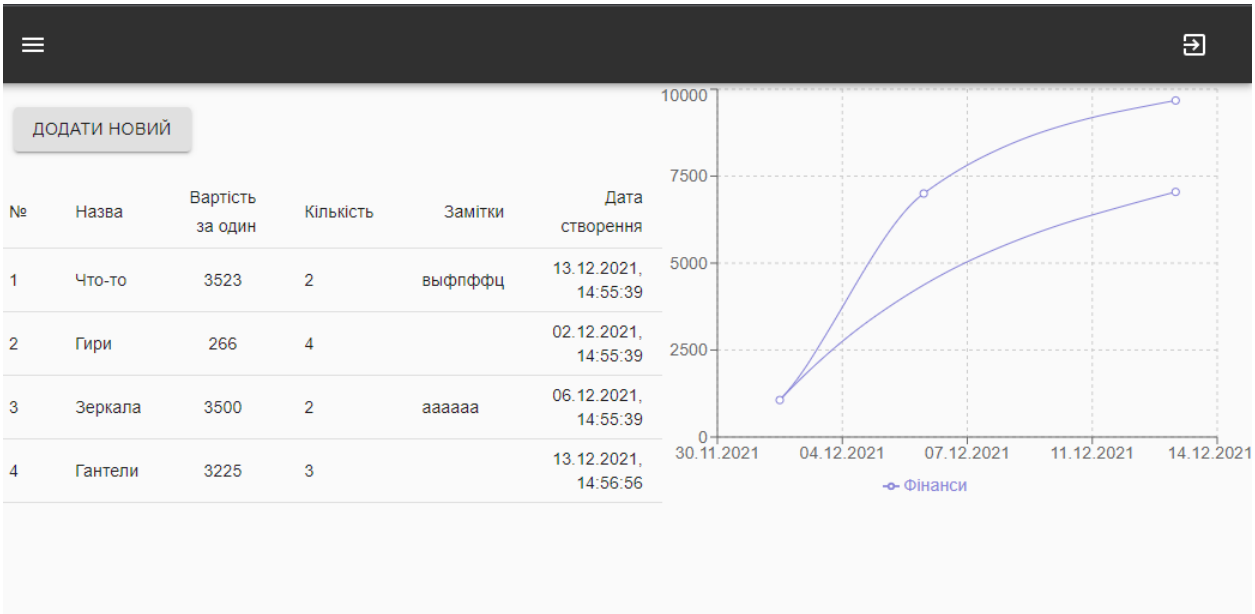


Рисунок 4.31 – Графіки витрат та прибутку

ВИСНОВКИ

Під час роботи над кваліфікаційною роботою магістра було описано вплив нових технологій на спосіб життя людей, а також те як фітнес-додатки впливають на заняття руховою активністю. Було проведено огляд існуючих програмних продуктів-аналогів та проведене порівняння за восьми категоріями. За результатами було створено таблицю з оцінками додатків та результати проведеного аналізу підтвердили актуальність обраної тематики.

Були сформульоновані мета та задачі дослідження. Детально описано постановку задачі на виконання роботи.

На етапі планування робіт було ідентифіковано мету проекту за допомогою методу SMART, сплановано зміст робіт за допомогою структури декомпозиції робіт.

Наступним етапом було проведено проектування системи. Були побудовані контекстна діаграма та діаграма декомпозиції структурно-функціональної моделі основного процесу системи підтримки роботи тренера залу функціонального тренінгу «Парашут». Для опису функціоналу інформаційної системи була створена діаграма варіантів використання.

Останнім кроком стала реалізація проекту, що включила в себе роботу над імплементацією бази даних, кабінетом тренера із веденням витрат на утримання спортивної зали.

Даний проект використовуватиметься власницею тренувального залу «Парашут» та буде отримувати оновлення та новий функціонал.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Who guidelines on physical activity and sedentary behaviour [Електронний ресурс] – 2020. – Режим доступу до ресурсу: https://apps.who.int/iris/bitstream/handle/10665/337001/9789240014886-eng.pdf
2. Global Recommendations on Physical Activity for Health. Geneva: World Health Organization [Електронний ресурс] – 2010. Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/books/NBK305049/
3. Worldwide trends in insufficient physical activity from 2001 to 2016: a pooled analysis of 358 population-based surveys with 1·9 million participants [Електронний ресурс] / R.Guthold, G. Stevens, L. Riley, F. Bull // Lancet Glob Health. – 2018. – Режим доступу до ресурсу: https://www.thelancet.com/journals/langlo/article/PIIS2214-109X(18)30357-7/fulltext#%20.
4. Lack of exercise is a major cause of chronic diseases [Електронний ресурс] / Frank W. Booth, Christian K. Roberts, Matthew J. Laye. – 2014. – Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4241367/.
5. Mobile technology and the digitization of healthcare [Електронний ресурс] / Bhavnani SP, Narula J, Sengupta PP. – 2016. – Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4914890/.
6. The measure of a man [Електронний ресурс] / Savage N. – 2017. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/pii/S0092867417306281.
7. Use of mobile applications to collect data in sport, health, and exercise science [Електронний ресурс] / Peart DJ, Shaw MP, Balsalobre-Fernández C. – 2018. – Режим доступу до ресурсу: https://journals.lww.com/nsca-

jscr/Abstract/2019/04000/Use_of_Mobile_Applications_to_Collect_Data_in.29.aspx.

8. Worldwide survey of fitness trends for 2019 [Електронний ресурс] / Thompson WR. – 2019. – Режим доступу до ресурсу: https://journals.lww.com/acsm-healthfitness/FullText/2018/11000/WORLDWIDE_SURVEY_OF_FITNESS_TRENDS_FOR_2019.6.aspx.
9. A Systematic Review of Fitness Apps and Their Potential Clinical and Sports Utility for Objective and Remote Assessment of Cardiorespiratory Fitness [Електронний ресурс] / Adrià Muntaner-Mas, Carl J. Lavie, Robert Ross, Steven N. Blair, Ross Arena. – 2019. – Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6422959/#CR28.
10. Applied sport sciences. Sprint performance and mechanical outputs computed with an iPhone app: Comparison with existing reference methods [Електронний ресурс] / Natalia Romero-Franco, Fernando Capelo-Ramírez, Adrián Castaño-Zambudio. – 2017. – Режим доступу до ресурсу: http://repositorio.ucam.edu/bitstream/handle/10952/3156/romerofranco2016.pdf?sequence=1.
11. The validity and reliability of an iPhone app for measuring running mechanics [Електронний ресурс] / Balsalobre-Fernández C, Agopyan H, Morin J-B. – 2017. – Режим доступу до ресурсу: https://journals.humankinetics.com/view/journals/jab/33/3/article-p222.xml.
12. Psychological mechanisms underlying the relationship between commercial physical activity app use and physical activity engagement [Електронний ресурс] / Jasmine M. Petersen, Lucy K. Lewis, Eva Kemps. – 2020. – Режим доступу до ресурсу: https://www.sciencedirect.com/science/article/abs/pii/S1469029219308568?via%3Dihub
13. Apps of steel: are exercise apps providing consumers with realistic expectations? A content analysis of exercise apps for presence of behavior

change theory [Електронний ресурс] / Logan T Cowan, Brittany A Brown, P Cougar Hall, Sarah A Van Wagenen, Riley J Hedin / Health Education & Behavior 40. – 2013. – Режим доступу до ресурсу: https://www.researchgate.net/publication/230880117_Apps_of_Steel_Are_Exercise_Apps_Providing_Consumers_With_Realistic_Expectations_A_Content_Analysis_of_Exercise_Apps_for_Presence_of_Behavior_Change_Theory.

14. Efficacy of interventions that use apps to improve diet, physical activity and sedentary behaviour: a systematic review [Електронний ресурс] / Stephanie Schoeppe, Wendy Van Lippevelde, Stephanie Alley // w. International Journal of Behavioral Nutrition and Physical Activity 13. – 2016. – Режим доступу до ресурсу: https://www.academia.edu/35669048/Efficacy_of_interventions_that_use_apps_to_improve_diet_physical_activity_and_sedentary_behaviour_a_systematic_review?auto=download.
15. "Keep Going!": Understanding the Implications of Coaching through Fitness Apps to Support Physical Training [Електронний ресурс] / Raluca-Alexandra Stoica. – 2018. – Режим доступу до ресурсу: https://uclic.ucl.ac.uk/content/2-study/4-current-taught-course/1-distinction-projects/12-18/stoica_ralucaalexandra_2018.pdf.
16. Evaluation of a personalized coaching system for physical activity: User appreciation and adherence [Електронний ресурс] / Julia S Mollee, Saskia J te Velde, Anouk Middelweerd. – 2017. – Режим доступу до ресурсу: https://www.researchgate.net/publication/322533523_Evaluation_of_a_personalized_coaching_system_for_physical_activity_user_appreciation_and_adherence.
17. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being [Електронний ресурс] / Richard M Ryan, Edward L Deci // American psychologist 55. – 2000. – Режим доступу до ресурсу:

https://selfdeterminationtheory.org/SDT/documents/2000_RyanDeci_SDT.pdf.

18. Technology As Experience. [Електронний ресурс] / John McCarthy and Peter Wright // Interactions 11. – 2004. – Режим доступу до ресурсу: https://www.researchgate.net/publication/224927635_Technology_as_Experience
19. Process Definition [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#process-definition
20. Process Instance [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#process-instance
21. User [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://core.telegram.org/bots/api#user
22. User [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#user
23. Telegram Messenger [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://telegram.org/
24. Workflow and Decision Automation Platform | Camunda [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://camunda.com/.
25. Developing it security metrics with goal question metric approach and smart criteria [Електронний ресурс] / Augustono Basuki. – 2017. – Режим доступу до ресурсу: https://www.researchgate.net/publication/321655537_DEVELOPING_IT_SECURITY_METRICS_WITH_GOAL_QUESTION_METRIC_APPROACH_AND_SMART_CRITERIA.
26. Work breakdown structure (WBS) [Електронний ресурс] / Mohaymen Alutbi. – 2020. – Режим доступу до ресурсу:

https://www.researchgate.net/publication/342163727_WORK_BREAKDOWN_STRUCTURE_WBS.

27. Process Definition [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#process-definition
28. Process Instance [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.camunda.org/manual/7.3/api-references/rest/#process-instance
29. Telegram Messenger [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://telegram.org/
30. Workflow and Decision Automation Platform | Camunda [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://camunda.com/

ДОДАТОК А
ПЛАНУВАННЯ РОБІТ

1. Ідентифікація мети IT-проекту

Для визначення мети проекту буде використовувати метод SMART за допомогою таких показників як:

- Specific – однозначна та конкретна ціль, яку можливо інтерпретувати лише одним чином,
- Measurable – ціль, що вимірюється в кількісних показниках,
- Achievable – ціль, яка є досяжною та реалістичною,
- Relevant – ціль, яка відповідає актуальним тенденціям сьогоднішнього ринку,
- Time-framed – ціль, що є обмежена у часі [25].

Результати деталізації мети проекту методом SMART розміщені у табл. А.1.

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Розробити веб-орієнтовану інформаційну систему підтримки діяльності роботи тренерів.
Measurable (вимірювана)	Передбачити в складі системи модулі для підтримки роботи тренерів, в тому числі для контролю фінансів, відвідуваннями занять, керування тренуваннями.
Achievable (досяжна)	Реалізувати програмні модулі інформаційної системи за допомогою використання мов C# та Typescript на фреймворка React та .NET Core відповідно. Для збереження даних розробити базу даних MySQL.
Relevant (актуальна)	Реалізувати зручний та інтуїтивно зрозумілий інтерфейс., простий та лаконічний дизайн.
Time-framed (обмежена у часі)	Виконати розробку з обмеженням у часі згідно календарного плану.

2. Планування змісту структури робіт IT-проекту

Важливим етапом роботи над проектом є планування змісту структури робіт. Правильно зроблене планування відіграє важливе значення при формуванні чітких очікувань та задач для кожного учасника команди, щоб завершити проект вчасно та в рамках бюджету. Основним інструментом на цьому етапі є Work Breakdown Structure (структура декомпозиції робіт).

WBS дозволяє розкласти великий проект на дрібні керовані секції, що дають кожному учаснику команди зосередитись на конкретних задачах. Чітко визначені задачі легко призначити одному з учасників команди. Також це допомагає точно оцінити потрібні ресурси на виконання кожної з робіт та відстежувати їх завершення.

Створена WBS для проекту наведена на рисунку А.1. В ній було виділено елементарні роботи, що необхідно закінчити для успішного завершення проекту.

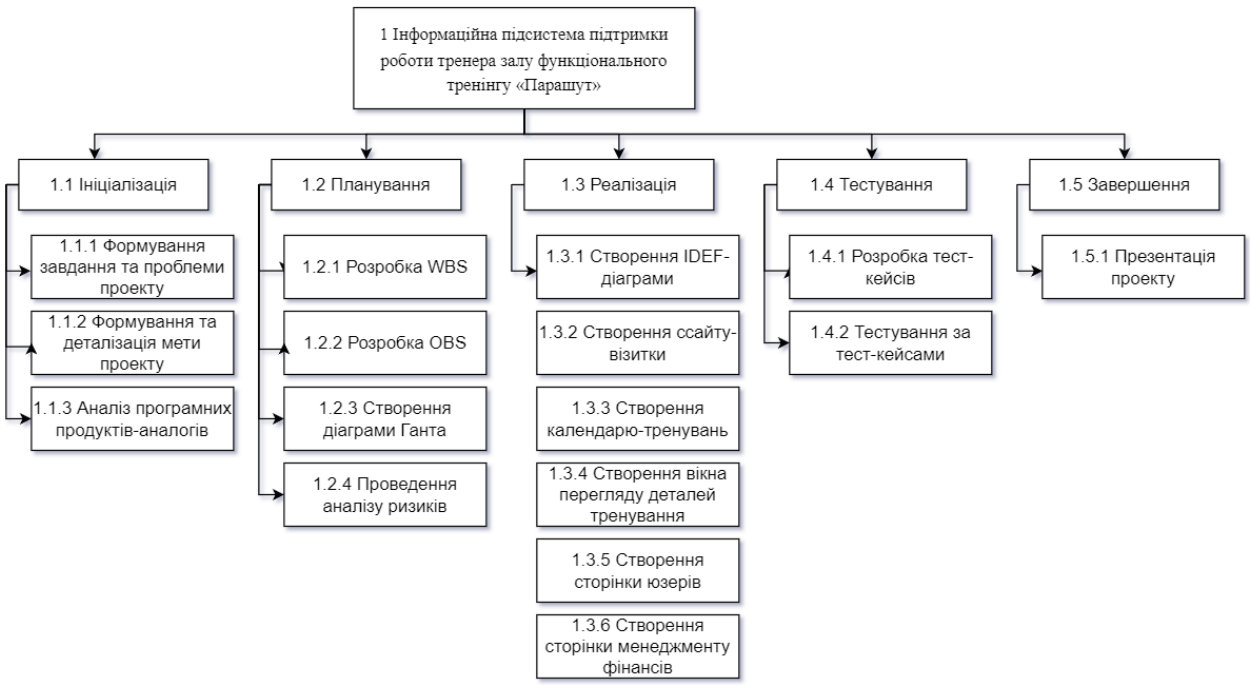


Рисунок А.1 – Структура декомпозиції робіт

Після побудови WBS тепер є необхідність у побудові OBS, яка має на увазі графічне відображення учасників проекту, що задіяні у його реалізації. Організаційна структура проекту показує які організаційні підрозділи задіяні у виконанні певних робіт. OBS створюється на основі уже розробленої WBS.

Список учасників проекту та їх ролі вказано в таблиці А.2. Діаграма OBS наведена на рисунку А.2.

Таблиця А.2 − Учасники проекту та їх ролі

Роль	Ім’я	Проектна роль
Виконавець	Шкура А.В.	Виконує збір та аналіз даних. Виконує розробку інформаційної системи та її тестування.
Консультант проекту	Ващенко С.М.	Формує завдання на розробку проекту.
Замовник	Глущенко С.В.	Надає вимоги до продукту, приймає готовий продукт.

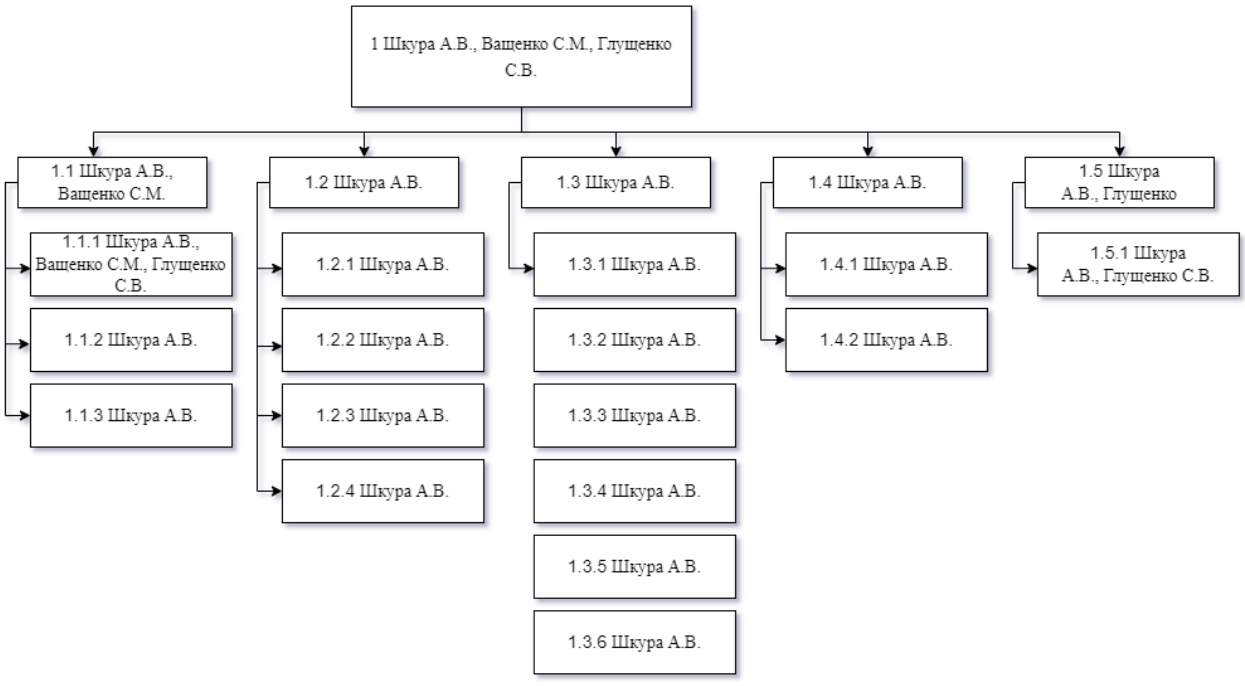


Рисунок А.2 – Організаційна структура проекту

3. Побудова календарного графіку виконання IT-проекту

Для того, щоб успішного виконати проект в зазначений термін створимо календарний план. Він допоможе графічно представити дати початок та закінчення кожної роботи. Також календарний план показує взаємозв'язки між ними. Тривалість кожної з робіт описана в годинах.

Роботи для побудови діаграми наведені на рисунку А.3. Графічне представлення діаграми Ганта зображене на рисунку А.4.

Задача	Начало	Завершен	Длит	Предшес	
	18.10.2021	22.12.2021	66д		
Ініціалізація	**18.10.2021**	25.10.2021	8д		⋮
Формування завдання та проблеми проекту	18.10.2021	19.10.2021	2д		⋮
Формування та деталізація мети проекту	20.10.2021	21.10.2021	2д	1.1	⋮
Аналіз програмних продуктів-аналогів	22.10.2021	25.10.2021	4д	1.2	⋮
Добавить задачу | Добавить веху					
Планування	**26.10.2021**	03.11.2021	9д	**1.3**	⋮
Розробка WBS	26.10.2021	27.10.2021	2д	1.3	⋮
Розробка OBS	28.10.2021	28.10.2021	1д	2.1	⋮
Створення діаграми Ганта	29.10.2021	31.10.2021	3д	2.2	⋮
Проведення аналізу ризиків	01.11.2021	03.11.2021	3д	2.3	⋮
Добавить задачу | Добавить веху					
Реалізація	**04.11.2021**	12.12.2021	39д	**2.4**	⋮
Створення IDEF-діаграми	04.11.2021	07.11.2021	4д	2.4	⋮
Створення ссайту-візитки	08.11.2021	12.11.2021	5д	3.1	⋮
Створення календарю-тренувань	13.11.2021	22.11.2021	10д	3.2	⋮
Створення вікна перегляду деталей тренування	23.11.2021	30.11.2021	8д	3.3	⋮
Створення сторінки юзерів	01.12.2021	06.12.2021	6д	3.4	⋮
Створення сторінки менеджменту фінансів	07.12.2021	12.12.2021	6д	3.5	⋮
Добавить задачу | Добавить веху					
Тестування	**13.12.2021**	21.12.2021	9д	**3.6**	⋮
Розробка тест-кейсів	13.12.2021	16.12.2021	4д	3.6	⋮
Тестування за тест-кейсами	17.12.2021	21.12.2021	5д	4.1	⋮
Добавить задачу | Добавить веху					
Завершення	**22.12.2021**	22.12.2021	1д	**4.2**	⋮
Презентація проекту	22.12.2021	22.12.2021	1д	4.2	⋮
Добавить задачу | Добавить веху					
Добавить новый подпроект					

Рисунок А.3 – Роботи для побудови діаграми

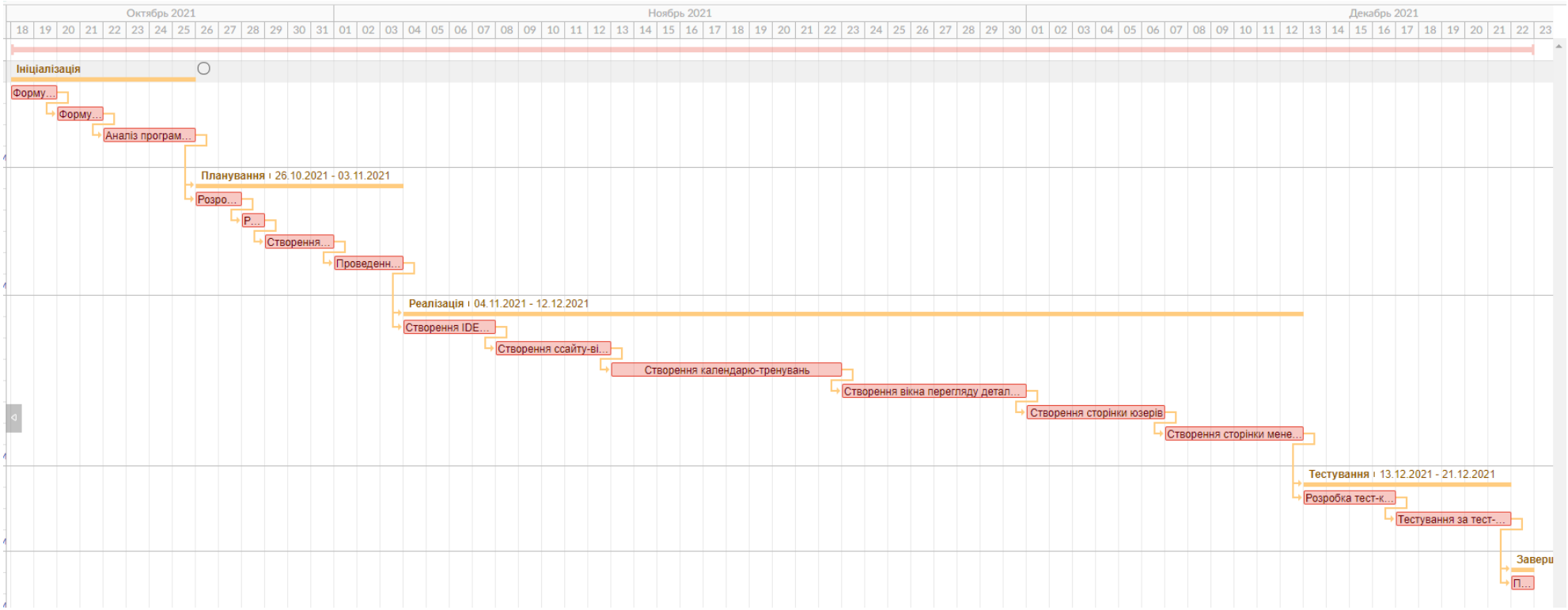


Рисунок А.4 – Графічне представлення діаграми Ганта

4 Планування ризиків проекту

Під час розробки проекту важливим його етапом є визначення та аналіз ризиків, адже вони можуть суттєво вплинути на час та якість розробки. Для того, щоб уникнути цього, виконаємо аналіз ризиків.

Таблиця А.3 містить виділені ризики, що можуть виникнути під час розробки проекту. Проведемо їх класифікацію за такими критеріями як ймовірність виникнення та величина втрат.

Таблиця А.3 – Класифікація ризиків

№	Назва ризику	Ймовірність виникнення	Величина витрат
R1	Некоректно складені вимоги до проекту	Слабка	Висока
R2	Зміна вимог до продукту	Середня	Середня
R3	Некоректна робота інформаційної системи	Середня	Висока
R4	Недотримання графіку	Висока	Середня
R5	Посереднє тестування	Середня	Слабка
R6	Хвороба виконувача проекту	Слабка	Середня

Побудуємо матрицю на основі створеної класифікації ризиків. В таблиці зелений колір відзначає прийнятні ризики, жовтий – виправдані, а червоний – недопустимі.

Таблиця А.4 – Матриця ризиків

Ймовірність виникнення	Величина витрат		
	1 Слабка	2 Середня	3 Висока
3 Висока		R4	
2 Середня	R5	R2	R3
1 Слабка		R6	R1

На основі отриманих значень після побудови матриці ризиків розподілимо за рівнем ризику.

Результати показано у таблиці А.5.

Таблиця А.5 – Класифікація за рівнем

№	Назва	Ризики, що входять
1	Прийнятні	R5, R6
2	Виправдані	R1, R2
3	Недопустимі	R3, R4

Для того, аби мінімізувати вплив будь-якого з описаних ризиків на виконання проекту, складемо план запобігання та реагування (табл. А.6).

Таблиця А.6 – Повна оцінка ризиків проекту

№	Опис ризику	План запобігання	План реакції
R1	Некоректно складені вимоги до проекту	Замовникові необхідно детально описати вимоги до проекту та оформити усе в письмовому чи цифровому вигляді.	Уважно проаналізувати поставлені задачі та у випадку неточностей, обговорити їх з замовником.

Продовження таблиці А.6

R2	Зміна вимог до продукту	Вимоги повинні бути описані чітко та бути затверджені до початку розробки продукту.	Коригувати план та терміни виконання, спробувати мінімізувати вплив на час завершення проекту.
R3	Некоректна робота інформаційної системи	Тестування роботи системи під час її розробки та після закінчення.	Аналіз та коригування знайденого у системі багу.
R4	Недотримання графіку	Розробник повинен створити чіткий та реалістичний графік за яким буде проводитись розробка продукту.	Обговорення можливого перенесення термінів чи відведення деякого функціоналу у подальші версії продукту.
R5	Посереднє тестування	Написання тестів, які покривають код продукту згідно до поставленого трешхолду.	Повторне тестування. Виправлення знайдених багів.
R6	Хвороба виконувача проекту	Мінімізувати можливість захворіти, виконувати запобіжні заходи.	Звернення до лікаря чи лікування вдома.

ДОДАТОК Б

ЛІСТИНГ КОДУ

ExpensesService.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using Parashut.Api.Db;
using Parashut.Api.Entities;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Parashut.Api.Services
{
    public class ExpensesService : IExpensesService
    {
        private readonly ParashutDbContext dbContext;
        private readonly ILogger<ExpensesService> logger;
        private readonly IHttpContextAccessor httpContextAccessor;

        public ExpensesService(ParashutDbContext dbContext, ILogger<ExpensesService> logge
r, IHttpContextAccessor httpContextAccessor)
        {
            this.dbContext = dbContext;
            this.logger = logger;
            this.httpContextAccessor = httpContextAccessor;
        }


        public async Task<Result<List<Expense>>> GetAllExpensesForTrainer()
        {
            try
            {
                User user = httpContextAccessor.HttpContext.Items["User"] as Entities.User
;

                if (user.Role == Role.Trainer)
                {
                    List<Expense> expenses = await dbContext.Expenses.Where(_ => _.Trainer
Id == user.Id).ToListAsync();

                    return Result<List<Expense>>.Success(expenses);
                }
                else
                {
                    string errorMessage = "Access Denied.";
                    logger.LogError(errorMessage);
                    return Result<List<Expense>>.Fail(StatusCodes.Status403Forbidden, erro
rMessage);
```

```
                }
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting all training sessions f
or trainer.";
                logger.LogError(ex, errorMessage);
                return Result<List<Expense>>.Fail(StatusCodes.Status500InternalServerError
, errorMessage);
            }
        }

        public async Task<Result<Expense>> Create(string name, float costPerOne, int amoun
t, string notes)
        {
            try
            {
                User user = httpContextAccessor.HttpContext.Items["User"] as User;

                if (user.Role == Role.Trainer)
                {
                    Expense expense = new Expense()
                    {
                        Id = new Guid(),
                        TrainerId = user.Id,
                        Name = name,
                        CostPerOne = costPerOne,
                        Amount = amount,
                        Notes = notes
                    };
                    await dbContext.Expenses.AddAsync(expense);
                    await dbContext.SaveChangesAsync();

                    return Result<Expense>.Success(expense);
                }
                else
                {
                    string errorMessage = "Access Denied.";
                    logger.LogError(errorMessage);
                    return Result<Expense>.Fail(StatusCodes.Status403Forbidden, errorMessa
ge);
                }
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting all training sessions f
or trainer.";
                logger.LogError(ex, errorMessage);
                return Result<Expense>.Fail(StatusCodes.Status500InternalServerError, erro
rMessage);
            }
        }
    }
}
```

TrainingSessionService.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.SignalR;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using Newtonsoft.Json;
using Newtonsoft.Json.Converters;
using Newtonsoft.Json.Serialization;
using Parashut.Api.Db;
using Parashut.Api.Entities;
using Parashut.Api.Hubs;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Transactions;
using Telegram.Bots.Types;

namespace Parashut.Api.Services
{
    public class TrainingSessionService : ITrainingSessionService
    {
        private readonly ITelegramBotService botService;
        private readonly ParashutDbContext dbContext;
        private readonly IGroupService groupService;
        private readonly IHubEventsService hubEventsService;
        private readonly ILogger<TrainingSessionService> logger;
        private readonly IHttpContextAccessor httpContextAccessor;
        private readonly IMembershipService membershipService;

        public TrainingSessionService(ITelegramBotService botService,
            ParashutDbContext dbContext,
            ILogger<TrainingSessionService> logger,
            IGroupService groupService,
            IHubEventsService hubEventsService,
            IHttpContextAccessor httpContextAccessor, IMembershipService membershipService
)
        {
            this.botService = botService;
            this.dbContext = dbContext;
            this.logger = logger;
            this.groupService = groupService;
            this.hubEventsService = hubEventsService;
            this.httpContextAccessor = httpContextAccessor;
            this.membershipService = membershipService;
        }

        public async Task<Result<List<TrainingSession>>> GetAllSessionsForUser()
        {
            try
            {
                Entities.User user = httpContextAccessor.HttpContext.Items["User"] as Enti
ties.User;

                if (user.Role == Role.Trainer) {
                    List<Guid> groupIds = (await groupService.GetAllGroupIdsForTrainer(use
r.Id)).Item;
```

```
                    List<TrainingSession> trainingSessions = await dbContext.TrainingSessi
ons
                        .Where(_ => groupIds.Contains(_.GroupId))
                        .ToListAsync();

                    return Result<List<TrainingSession>>.Success(trainingSessions);
                }
                else
                {
                    List<TrainingSession> trainingSessions = await dbContext.TrainingSessi
ons
                        .Where(_ => _.GroupId == user.GroupId)
                        .ToListAsync();

                    return Result<List<TrainingSession>>.Success(trainingSessions);
                }
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting all training sessions f
or trainer.";
                logger.LogError(ex, errorMessage);
                return Result<List<TrainingSession>>.Fail(StatusCodes.Status500InternalSer
verError, errorMessage);
            }
        }

        public async Task<Result<List<TrainingSession>>> Create(Guid groupId, List<DateTim
e> dates, string textMessage)
        {
            try
            {
                if (!dates.Any())
                {
                    string errorMessage = "Dates field can not be empty.";
                    logger.LogError(errorMessage);
                    return Result<List<TrainingSession>>.Fail(StatusCodes.Status400BadRequ
est, errorMessage);
                }
                dates.Sort();

                using TransactionScope trScope = new TransactionScope(TransactionScopeAsyn
cFlowOption.Enabled);
                Group group = (await groupService.GetById(groupId)).Item;
                DateTime firstDate = dates.First();
                string question = firstDate.ToString("dddd, MMMM dd");


                List<string> options = dates.Select(date => date.ToString("t")).ToList();
                if (options.Count == 1)
                {
                    options.Add(")");
                }

                if (!string.IsNullOrWhiteSpace(textMessage))
                {
                    await botService.SendMessageToGroup(group.TelegramChatId, textMessage)
;
```

```
                }

                PollMessage pollMessage =
                    (await botService.SendPollToGroup(group.TelegramChatId, question, opti
ons)).Item;

                List<TrainingSession> sessions = new List<TrainingSession>();

                for (int i = 0; i < dates.Count; i++)
                {
                    sessions.Add(new TrainingSession()
                    {
                        GroupId = groupId,
                        Date = dates[i],
                        PollId = pollMessage.Poll.Id,
                        PollMessageId = pollMessage.Id,
                        PollOptionId = (uint)i,
                        Status = TrainingSessionStatus.Ongoing
                    });
                }

                await dbContext.AddRangeAsync(sessions);
                await dbContext.SaveChangesAsync();

                await hubEventsService.SendTrainingSessionsCreated(group.TrainerId.ToStrin
g(), sessions);

                trScope.Complete();

                return Result<List<TrainingSession>>.Success(sessions);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on training session creation.";
                logger.LogError(ex, errorMessage);
                return Result<List<TrainingSession>>.Fail(StatusCodes.Status500InternalSer
verError, errorMessage);
            }
        }

        public async Task<Result<TrainingSession>> GetByPoll(string pollId, uint optionId)
        {
            try
            {
                TrainingSession session = await dbContext.TrainingSessions
                    .FirstOrDefaultAsync(_ => _.PollId == pollId && _.PollOptionId == opti
onId);

                return Result<TrainingSession>.Success(session);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting session by pollId and o
ption.";
                logger.LogError(ex, errorMessage);
                return Result<TrainingSession>.Fail(500, errorMessage);
            }
        }
```

```
        public async Task<Result<List<TrainingSession>>> GetByPoll(string pollId)
        {
            try
            {
                List<TrainingSession> sessions = await dbContext.TrainingSessions
                    .Where(_ => _.PollId == pollId).ToListAsync();

                return Result<List<TrainingSession>>.Success(sessions);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting sessions by pollId.";
                logger.LogError(ex, errorMessage);
                return Result<List<TrainingSession>>.Fail(500, errorMessage);
            }
        }

        public async Task<Result<TrainingSession>> CancelSession(Guid sessionId)
        {
            try
            {
                TrainingSession session = await dbContext.TrainingSessions.Include(_ => _.
Group)
                    .FirstOrDefaultAsync(_ => _.Id == sessionId);

                session.Status = TrainingSessionStatus.Canceled;

                await dbContext.SaveChangesAsync();
                await botService.StopPoll(session.Group.TelegramChatId, session.PollMessag
eId);

                await hubEventsService.SendTrainingSessionUpdated(session.Group.TrainerId.
ToString(), session);

                return Result<TrainingSession>.Success(session);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting group by id.";
                logger.LogError(ex, errorMessage);
                return Result<TrainingSession>.Fail(500, errorMessage);
            }
        }

        public async Task<Result<TrainingSession>> FinishSession(Guid sessionId, Dictionar
y<Guid, bool> didAttendMap)
        {
            try
            {
                TrainingSession session = await dbContext.TrainingSessions.Include(_ => _.
Group)
                    .FirstOrDefaultAsync(_ => _.Id == sessionId);

                if(session == null)
                {
                    string errorMessage = "Session not found.";
                    logger.LogError(errorMessage);
```

```
                    return Result<TrainingSession>.Fail(500, errorMessage);
                }
                using TransactionScope trScope = new TransactionScope(TransactionScopeAsyn
cFlowOption.Enabled);

                session.Status = TrainingSessionStatus.Finished;

                List<TrainingSessionAttendee> attendees = await dbContext.TrainingSessionA
ttendees
                    .Where(tsa => tsa.SessionId == sessionId).ToListAsync();

                foreach (var item in attendees)
                {
                    item.DidAttend = didAttendMap[item.Id];
                    if (item.DidAttend)
                    {
                        Membership membership = (await membershipService.GetLastMembership
ForUser(item.UserId)).Item;
                        if(membership == null)
                        {
                            membership = (await membershipService.Create(item.UserId, null
, session.Date)).Item;
                        }
                        else if (!(await membership.GetIsActiveAsync(dbContext.MembershipT
ypes)))
                        {
                            membership = (await membershipService.Create(item.UserId, memb
ership.MembershipTypeId, session.Date)).Item;
                        }
                        item.MembershipId = membership.Id;
                        membership.AttendedSessionsCount++;
                    }
                }

                await dbContext.SaveChangesAsync();
                trScope.Complete();
                await hubEventsService.SendTrainingSessionUpdated(session.Group.TrainerId.
ToString(), session);

                return Result<TrainingSession>.Success(session);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting group by id.";
                logger.LogError(ex, errorMessage);
                return Result<TrainingSession>.Fail(500, errorMessage);
            }
        }
    }
}
```

TrainingSessionAttendeeService.cs

```
using Microsoft.AspNetCore.Http;
```

```
using Microsoft.AspNetCore.SignalR;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using Parashut.Api.Db;
using Parashut.Api.Entities;
using Parashut.Api.Hubs;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Transactions;
using Telegram.Bots.Types;

namespace Parashut.Api.Services
{
    public class TrainingSessionAttendeeService : ITrainingSessionAttendeeService
    {
        private readonly ITelegramBotService botService;
        private readonly ParashutDbContext db;
        private readonly ILogger<TrainingSessionAttendeeService> logger;
        private readonly ITrainingSessionService sessionService;
        private readonly IUserService userService;
        private readonly IHubEventsService hubEventsService;
        private readonly IGroupService groupService;
        private readonly IHttpContextAccessor httpContextAccessor;

        public TrainingSessionAttendeeService(ITelegramBotService botService,
            ParashutDbContext dbContext,
            ILogger<TrainingSessionAttendeeService> logger,
            ITrainingSessionService sessionService,
            IUserService userService,
            IHubEventsService hubEventsService,
            IGroupService groupService,
            IHttpContextAccessor httpContextAccessor)
        {
            this.botService = botService;
            this.db = dbContext;
            this.logger = logger;
            this.sessionService = sessionService;
            this.userService = userService;
            this.hubEventsService = hubEventsService;
            this.groupService = groupService;
            this.httpContextAccessor = httpContextAccessor;
        }

        public async Task<Result> HandleUserVote(PollAnswer pollAnswer)
        {
            try
            {
                Entities.User user = (await userService.GetByTelegramId(pollAnswer.User.Id
)).Item;

                if (pollAnswer.OptionIds.Any())
                {
                    TrainingSession session = (await sessionService
                        .GetByPoll(pollAnswer.PollId, pollAnswer.OptionIds[0])
```

```
                    ).Item;

                    if (session == null)
                    {
                        string errorMessage = "Session not chosen.";
                        logger.LogError(errorMessage);
                        return Result.Fail(StatusCodes.Status500InternalServerError, error
Message);
                    }

                    Group group = (await groupService.GetById(session.GroupId)).Item;

                    TrainingSessionAttendee attendee = await db.TrainingSessionAttendees
                        .FirstOrDefaultAsync(tsa => tsa.UserId == user.Id && tsa.SessionId
 == session.Id);
                    if (attendee == null)
                    {
                        attendee = new TrainingSessionAttendee()
                        {
                            SessionId = session.Id,
                            UserId = user.Id,
                            DidAttend = true,
                            DidCancel = false,
                        };
                        await db.TrainingSessionAttendees.AddAsync(attendee);
                        await db.SaveChangesAsync();

                        await hubEventsService.SendSessionAttendeeCreated(group.TrainerId.
ToString(), attendee);
                    }
                    else {
                        attendee.DidAttend = true;
                        attendee.DidCancel = false;
                        await hubEventsService.SendSessionAttendeeCreated(group.TrainerId.
ToString(), attendee);
                    }
                    #region Membership logic
                    //Membership membership = (await membershipService.GetLastMembershipFo
rUser(user.Id)).Item;

                    //// TODO: check out what the start date should be and refactor
                    //if (membership == null)
                    //{
                    //    MembershipType defaultMembershipType = (await membershipTypeServ
ice.GetDefaultMembershipType()).Item;
                    //    membership = (await membershipService.Create(user.Id, defaultMem
bershipType.Id, session.Date)).Item;
                    //    await db.Memberships.AddAsync(membership);
                    //    await db.SaveChangesAsync();
                    //}
                    //else if(!membership.IsActive)
                    //{
                    //    membership = (await membershipService.Create(user.Id, membership
.MembershipTypeId, session.Date)).Item;
                    //    await db.Memberships.AddAsync(membership);
                    //    await db.SaveChangesAsync();
                    //}
                    #endregion
```

```
                }
                else
                {
                    IEnumerable<TrainingSession> sessions = (await sessionService
                        .GetByPoll(pollAnswer.PollId)
                    ).Item;
                    var sessionIds = sessions.Select(s => s.Id);
                    TrainingSessionAttendee attendee = await db.TrainingSessionAttendees
                        .FirstOrDefaultAsync(tsa => tsa.UserId == user.Id && sessionIds.Co
ntains(tsa.SessionId));

                    if(attendee == null)
                    {
                        string errorMessage = "Attendee not found.";
                        logger.LogError(errorMessage);
                        return Result.Fail(StatusCodes.Status500InternalServerError, error
Message);
                    }

                    attendee.DidCancel = true;
                    attendee.DidAttend = false;

                    Group group = (await groupService.GetById(sessions.First().GroupId)).I
tem;
                    await hubEventsService.SendSessionAttendeeCreated(group.TrainerId.ToSt
ring(), attendee);
                }
                await db.SaveChangesAsync();

                return Result.Success();
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on training session creation.";
                logger.LogError(ex, errorMessage);
                return Result.Fail(StatusCodes.Status500InternalServerError, errorMessage)
;
            }
        }

        public async Task<Result<List<TrainingSessionAttendee>>> GetAllAttendeesForUser()
        {
            try
            {
                Entities.User user = httpContextAccessor.HttpContext.Items["User"] as Enti
ties.User;
                if(user.Role == Role.Trainer)
                {
                    //TODO: figure out a way to exclude outright
                    List<TrainingSessionAttendee> attendees = await (from tGroup in db.Gro
ups
                        join session in db.TrainingSessions on tGroup.Id equals session.Gr
oupId
                        join attendee in db.TrainingSessionAttendees on session.Id equals
attendee.SessionId
                        where tGroup.TrainerId == user.Id
                        select attendee).ToListAsync();
```

```
                    attendees = attendees.Select(a => { a.Session = null; return a; }).ToL
ist();

                    return Result<List<TrainingSessionAttendee>>.Success(attendees);
                }
                else
                {
                    List<TrainingSessionAttendee> attendees = await (from session in db.Tr
ainingSessions
                        join attendee in db.TrainingSessionAttendees on session.Id equals
attendee.SessionId
                        where session.GroupId == user.GroupId
                        select attendee).ToListAsync();
                    attendees = attendees.Select(a => { a.Session = null; return a; }).ToL
ist();

                    return Result<List<TrainingSessionAttendee>>.Success(attendees);
                }
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting all training sessions f
or trainer.";
                logger.LogError(ex, errorMessage);
                return Result<List<TrainingSessionAttendee>>.Fail(StatusCodes.Status500Int
ernalServerError, errorMessage);
            }
        }
    }
}
```

MembershipService.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Logging;
using Parashut.Api.Db;
using Parashut.Api.Entities;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Parashut.Api.Services
{
    public class MembershipService : IMembershipService
    {
        private readonly ParashutDbContext dbContext;
        private readonly ILogger<MembershipService> logger;
        private readonly IGroupService groupService;
        private readonly IHttpContextAccessor httpContextAccessor;
        private readonly IMembershipTypeService membershipTypeService;

        public MembershipService(
            ParashutDbContext dbContext,
            ILogger<MembershipService> logger,
            IGroupService groupService,
```

```
            IHttpContextAccessor httpContextAccessor,
            IMembershipTypeService membershipTypeService)
        {
            this.dbContext = dbContext;
            this.logger = logger;
            this.groupService = groupService;
            this.httpContextAccessor = httpContextAccessor;
            this.membershipTypeService = membershipTypeService;
        }

        public async Task<Result<Membership>> Create(Guid userId, Guid? membershipTypeId,
DateTime? startDate, bool wasPaidFor = false)
        {
            try
            {
                DateTime membershipStartDate = startDate ?? DateTime.Now;
                Guid typeId = membershipTypeId ?? (await membershipTypeService.GetDefaultM
embershipType()).Item.Id;

                Membership membership = new Membership()
                {
                    UserId = userId,
                    MembershipTypeId = typeId,
                    StartDate = membershipStartDate,
                    EndDate = membershipStartDate.AddMonths(1),
                    WasPaidFor = wasPaidFor,
                    AttendedSessionsCount = 0,
                };

                await dbContext.Memberships.AddAsync(membership);
                await dbContext.SaveChangesAsync();
                return Result<Membership>.Success(membership);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on membership creation.";
                logger.LogError(ex, errorMessage);
                return Result<Membership>.Fail(StatusCodes.Status500InternalServerError, e
rrorMessage);
            }
        }

        public async Task<Result<List<Membership>>> GetAllMembershipsForUser()
        {
            try
            {
                Entities.User user = httpContextAccessor.HttpContext.Items["User"] as User
;

                if(user.Role == Role.Trainer)
                {
                    List<Guid> groupIds = (await groupService.GetAllGroupIdsForTrainer(use
r.Id)).Item;
                    List<Guid> userIds = await dbContext.Users
                        .Where(_ => _.GroupId.HasValue && groupIds.Contains((Guid)_.GroupI
d))
                        .Select(_ => _.Id)
                        .ToListAsync();
```

```
                    List<Membership> memberships = await dbContext.Memberships
                        .Where(_ => userIds.Contains(_.UserId))
                        .ToListAsync();

                    return Result<List<Membership>>.Success(memberships);
                }
                else
                {
                    List<Membership> memberships = await dbContext.Memberships
                        .Where(_ => _.UserId == user.Id)
                        .ToListAsync();

                    return Result<List<Membership>>.Success(memberships);
                }
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting all users.";
                logger.LogError(ex, errorMessage);
                return Result<List<Membership>>.Fail(StatusCodes.Status500InternalServerEr
ror, errorMessage);
            }
        }

        public async Task<Result<Membership>> GetLastMembershipForUser(Guid userId)
        {
            try
            {
                Membership membership = await dbContext.Memberships
                    .OrderByDescending(_ => _.CreatedAt)
                    .FirstOrDefaultAsync(_ => _.UserId == userId);

                return Result<Membership>.Success(membership);
            }
            catch (Exception ex)
            {
                string errorMessage = "Unexpected error on getting all users.";
                logger.LogError(ex, errorMessage);
                return Result<Membership>.Fail(StatusCodes.Status500InternalServerError, e
rrorMessage);
            }
        }
    }
}
```

ExpensesController.cs

```
using Microsoft.AspNetCore.Mvc;
using Parashut.Api.Entities;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using Parashut.Api.ViewModels;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Parashut.Api.Controllers
{
```

```
    [Route("api/[controller]")]
    [ApiController]
    public class ExpensesController : ControllerBase
    {
        private readonly IExpensesService expensesService;

        public ExpensesController(IExpensesService expensesService)
        {
            this.expensesService = expensesService;
        }

        [HttpPost]
        public async Task<IActionResult> Create([FromBody] CreateExpenseRequest request)
        {
            Result<Expense> result = await expensesService.Create(request.Name,request.Cos
tPerOne, request.Amount, request.Notes);

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }

        [HttpGet("TrainerExpenses")]
        public async Task<IActionResult> GetAllSessionsForUser()
        {
            Result<List<Expense>> result = await expensesService.GetAllExpensesForTrainer(
);

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }
    }
}
```

TrainingSessionAttendeeController.cs

```
using Microsoft.AspNetCore.Mvc;
using Parashut.Api.Entities;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Parashut.Api.Controllers
{
    [Route("api/[controller]")]
```

```
    [ApiController]
    public class TrainingSessionAttendeeController : ControllerBase
    {
        private readonly ITrainingSessionAttendeeService trainingSessionAttendeeService;

        public TrainingSessionAttendeeController(ITrainingSessionAttendeeService trainingS
essionAttendeeService)
        {
            this.trainingSessionAttendeeService = trainingSessionAttendeeService;
        }

        [HttpGet("CurrentUserTrainingAttendees")]
        public async Task<IActionResult> GetAllAttendeesForUser()
        {
            Result<List<TrainingSessionAttendee>> result = await trainingSessionAttendeeSe
rvice.GetAllAttendeesForUser();

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }
    }
}
```

TrainingSessionController.cs

```
using Microsoft.AspNetCore.Mvc;
using Parashut.Api.Entities;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using Parashut.Api.ViewModels;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Parashut.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class TrainingSessionController : ControllerBase
    {
        private readonly ITrainingSessionService trainingSessionService;

        public TrainingSessionController(ITrainingSessionService trainingSessionService)
        {
            this.trainingSessionService = trainingSessionService;
        }

        [HttpPost]
        public async Task<IActionResult> Create([FromBody]CreateTrainingSessionRequest req
uest)
        {
            Result<List<TrainingSession>> result = await trainingSessionService.Create(req
uest.GroupId, request.Dates, request.TextMessage);
```

```
            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }

        [HttpGet("CurrentUserTrainingSessions")]
        public async Task<IActionResult> GetAllSessionsForUser()
        {
            Result<List<TrainingSession>> result = await trainingSessionService.GetAllSess
ionsForUser();

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }

        [HttpPut("Cancel")]
        public async Task<IActionResult> CancelSession(Guid id)
        {
            Result<TrainingSession> result = await trainingSessionService.CancelSession(id
);

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }

        [HttpPut("Finish")]
        public async Task<IActionResult> FinishSession(FinishTrainingSessionRequest reques
t)
        {
            Result<TrainingSession> result = await trainingSessionService
                .FinishSession(request.SessionId, request.AttendeeDidAttendMap);

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }
    }
}
```

MembershipController.cs

```
using Microsoft.AspNetCore.Mvc;
```

```
using Parashut.Api.Entities;
using Parashut.Api.Services.Abstraction;
using Parashut.Api.Utils;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Parashut.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MembershipController : ControllerBase
    {
        private readonly IMembershipService membershipService;

        public MembershipController(IMembershipService membershipService)
        {
            this.membershipService = membershipService;
        }


        [HttpGet("CurrentUserMemberships")]
        public async Task<IActionResult> GetAllMembershipsForUser()
        {
            Result<List<Membership>> result = await membershipService.GetAllMembershipsFor
User();

            if (result.IsSuccess)
            {
                return Ok(result.Item);
            }

            return StatusCode(result.StatusCode, result.ErrorMessage);
        }
    }
}
```

ParashutDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using Parashut.Api.Db.ModelConfigurations;
using Parashut.Api.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Parashut.Api.Db
{
    public class ParashutDbContext : DbContext
    {
        public ParashutDbContext(DbContextOptions<ParashutDbContext> options) : base(optio
ns)
        {
        }

        public DbSet<User> Users { get; set; }
```

```
        public DbSet<Group> Groups { get; set; }
        public DbSet<MembershipType> MembershipTypes { get; set; }
        public DbSet<Membership> Memberships { get; set; }
        public DbSet<TrainingSession> TrainingSessions { get; set; }
        public DbSet<TrainingSessionAttendee> TrainingSessionAttendees { get; set; }
        public DbSet<Expense> Expenses { get; set; }
        public DbSet <UserStats> UserStats { get; set; }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.ApplyConfiguration(new UserConfiguration());
            modelBuilder.ApplyConfiguration(new GroupConfiguration());
            modelBuilder.ApplyConfiguration(new MembershipTypeConfiguration());
            modelBuilder.ApplyConfiguration(new MembershipConfiguration());
            modelBuilder.ApplyConfiguration(new TrainingSessionConfiguration());
            modelBuilder.ApplyConfiguration(new TrainingSessionAttendeeConfiguration());
            modelBuilder.ApplyConfiguration(new ExpenseConfiguration());
            modelBuilder.ApplyConfiguration(new UserStatsConfig());
        }
    }
}
```

ExpenseConfiguration.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Parashut.Api.Entities;

namespace Parashut.Api.Db.ModelConfigurations
{
    public class ExpenseConfiguration : BaseEntityConfiguration<Expense>
    {
        public override void Configure(EntityTypeBuilder<Expense> builder)
        {
            base.Configure(builder);

            builder.ToTable("Expenses");

            // Configure columns
            builder.Property(_ => _.TrainerId).HasColumnType("CHAR(36)").IsRequired();
            builder.Property(_ => _.Name).HasColumnType("text").IsRequired();
            builder.Property(_ => _.CostPerOne).HasColumnType("float").IsRequired();
            builder.Property(_ => _.Amount).HasColumnType("int").IsRequired();
            builder.Property(_ => _.Notes).HasColumnType("text");

            builder.HasOne(e => e.Trainer).WithMany(u => u.Expenses).HasForeignKey(m => m.
TrainerId);
        }
    }
}
```

MembershipConfiguration.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
```

```
using Parashut.Api.Entities;

namespace Parashut.Api.Db.ModelConfigurations
{
    public class MembershipConfiguration : BaseEntityConfiguration<Membership>
    {
        public override void Configure(EntityTypeBuilder<Membership> builder)
        {
            base.Configure(builder);

            builder.ToTable("Memberships");

            // Configure columns
            builder.Property(_ => _.UserId).HasColumnType("CHAR(36)").IsRequired();
            builder.Property(_ => _.MembershipTypeId).HasColumnType("CHAR(36)").IsRequired
();
            builder.Property(_ => _.StartDate).HasColumnType("date").IsRequired();
            builder.Property(_ => _.EndDate).HasColumnType("date").IsRequired();
            builder.Property(_ => _.WasExtended).HasDefaultValue(false).IsRequired();
            builder.Property(_ => _.AttendedSessionsCount).HasColumnType("int").HasDefault
Value(0).IsRequired();
            builder.Property(_ => _.WasPaidFor).HasDefaultValue(false).IsRequired();

            builder.HasOne(m => m.MembershipType).WithMany(mt => mt.Memberships).HasForeig
nKey(m => m.MembershipTypeId);
            builder.HasOne(m => m.User).WithMany(u => u.Memberships).HasForeignKey(m => m.
UserId);
        }
    }
}
```

TrainingSessionAttendeeConfiguration.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Parashut.Api.Entities;

namespace Parashut.Api.Db.ModelConfigurations
{
    public class TrainingSessionAttendeeConfiguration : BaseEntityConfiguration<TrainingSe
ssionAttendee>
    {
        public override void Configure(EntityTypeBuilder<TrainingSessionAttendee> builder)
        {
            base.Configure(builder);

            builder.ToTable("TrainingSessionAttendees");

            builder.HasIndex(_ => _.SessionId).HasDatabaseName("Idx_SessionId");
            builder.HasIndex(_ => _.UserId).HasDatabaseName("Idx_UserId");

            builder.Property(_ => _.SessionId).HasColumnType("CHAR(36)").IsRequired();
            builder.Property(_ => _.UserId).HasColumnType("CHAR(36)").IsRequired();
            builder.Property(_ => _.MembershipId).HasColumnType("CHAR(36)");
            builder.Property(_ => _.DidCancel).HasDefaultValue(false).IsRequired();
            builder.Property(_ => _.DidAttend).HasDefaultValue(true).IsRequired();
```

```
            builder.HasOne(tsa => tsa.Session).WithMany(ts => ts.Attendees).HasForeignKey(
m => m.SessionId);
            builder.HasOne(tsa => tsa.Membership).WithMany(m => m.AttendedSessions).HasFor
eignKey(m => m.MembershipId);
        }
    }
}
```

TrainingSessionConfiguration

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Parashut.Api.Entities;

namespace Parashut.Api.Db.ModelConfigurations
{
    public class TrainingSessionConfiguration : BaseEntityConfiguration<TrainingSession>
    {
        public override void Configure(EntityTypeBuilder<TrainingSession> builder)
        {
            base.Configure(builder);

            builder.ToTable("TrainingSessions");

            builder.HasIndex(_ => _.PollId).HasDatabaseName("Idx_PollId");
            builder.HasIndex(_ => _.PollMessageId).HasDatabaseName("Idx_PollMessageId");

            // Configure columns
            builder.Property(_ => _.GroupId).HasColumnType("CHAR(36)").IsRequired();
            builder.Property(_ => _.PollMessageId).HasColumnType("int").IsRequired();
            builder.Property(_ => _.PollId).HasColumnType("char(255)").IsRequired();
            builder.Property(_ => _.PollOptionId).HasColumnType("tinyint").IsRequired();
            builder.Property(_ => _.Date).HasColumnType("datetime").IsRequired();
            builder.Property(_ => _.Status)
                .HasColumnType("char(15)")
                .HasConversion<string>()
                .HasDefaultValue(TrainingSessionStatus.Ongoing)
                .IsRequired();

            builder.HasOne(ts => ts.Group).WithMany(g => g.TrainingSessions).HasForeignKey
(m => m.GroupId);
        }
    }
}
```

IExpense.ts

```
import { IBaseModel } from './IBaseModel'

export interface IExpense extends IBaseModel {
  id: string
  trainerId: string
  name: string
  costPerOne: number
  amount: number
  notes?: string
```

```
  createdAt: number
  updatedAt: number
}
```

IMembership.ts

```
import { IBaseModel } from './IBaseModel'

export interface IMembership extends IBaseModel {
  id: string
  userId: string
  membershipTypeId: string
  startDate: number
  endDate: number
  wasPaidFor: boolean
  wasExtended: boolean
  attendedSessionsCount: number
  createdAt: number
  updatedAt: number
}
```

ITrainingSession.ts

```
import { IBaseModel } from './IBaseModel'
import { TrainingSessionStatus } from './TrainingSessionStatus'

export interface ITrainingSession extends IBaseModel {
  id: string
  groupId: string
  date: number
  status: TrainingSessionStatus
  createdAt: number
  updatedAt: number
}
```

ITrainingSessionAttendee.ts

```
import { IBaseModel } from './IBaseModel'

export interface ITrainingSessionAttendee extends IBaseModel {
  id: string
  sessionId: string
  userId: string
  membershipId?: string
  didCancel: boolean
  didAttend: boolean
  createdAt: number
  updatedAt: number
}
```

TrainingCalendar.connect.tsx

```
import { connect } from 'react-redux'
import { Shim } from '../../store/shim'
import { Dispatch } from 'redux'
import usersSelectors from '../../store/users/usersSelectors'
```

```
import trainingSessionsSelectors from '../../store/trainingSessions/trainingSessionsSelect
ors'
import groupSelectors from '../../store/groups/groupsSelectors'
import selectors from '../../store/selectors'

const mapStateToProps = (state: Shim) => {
  return {
    users: usersSelectors.getUsersList(state),
    sessions: trainingSessionsSelectors.getSessionsList(state),
    groups: groupSelectors.getGroupsMap(state),
    isFetching: selectors.getIsFetching(state),
  }
}

const mapDispatchToProps = (dispatch: Dispatch) => {
  return {
    // getAllUsersForTrainer: bindActionCreators(getAllUsersForTrainer, dispatch),
  }
}

export const connectToState = (component: React.ComponentType<any>) =>
  connect(mapStateToProps, mapDispatchToProps)(component)
```

TrainingCalendar.styles.ts

```
import { createStyles, Theme } from '@material-ui/core'

export const styles = (theme: Theme) =>
  createStyles({
    calendarWrapper: {
      '& .fc': {
        maxWidth: '1100px',
        margin: '0 auto',
        padding: '10px',
        height: 'calc(100vh - 64px)',
      },
    },
  })
```

TrainingCalendar.tsx

```
import React from 'react'
import { withStyles } from '@material-ui/core'
import {
  TrainingCalendarProps,
  TrainingCalendarStateProps,
} from './TrainingCalendar.types'
import { styles } from './TrainingCalendar.styles'
import FullCalendar, {
  EventClickArg,
  EventContentArg,
  EventInput,
  EventSourceInput,
  FormatterInput,
} from '@fullcalendar/react'
import dayGridPlugin from '@fullcalendar/daygrid'
import interactionPlugin, { DateClickArg } from '@fullcalendar/interaction'
import timeGridPlugin from '@fullcalendar/timegrid'
import autobind from 'autobind-decorator'
```

```
import { connectToState } from './TrainingCalendar.connect'
import { compose } from 'redux'
import SessionCreationModal from './SessionCreationModal/SessionCreationModal'
import uaLocale from '@fullcalendar/core/locales/uk'
import SessionDetailsModal from './SessionDetailsModal/SessionDetailsModal'
import EventContent from './EventContent/EventContent'

export class TrainingCalendarBase extends React.Component<
  TrainingCalendarProps,
  TrainingCalendarStateProps
> {
  constructor(props: TrainingCalendarProps) {
    super(props)
    this.state = {}
  }
  render() {
    const { classes } = this.props
    const { clickedDate, selectedSession } = this.state
    const timeFormat: FormatterInput = {
      hour: '2-digit',
      minute: '2-digit',
      meridiem: false,
    }
    return (
      <>
        {clickedDate && (
          <SessionCreationModal
            closeModal={this.closeCreationModal}
            date={clickedDate}
          />
        )}
        {selectedSession && (
          <SessionDetailsModal
            closeModal={this.closeDetailsModal}
            session={selectedSession}
          />
        )}
        <div className={classes.calendarWrapper}>
          <FullCalendar
            plugins={[dayGridPlugin, interactionPlugin, timeGridPlugin]}
            headerToolbar={{
              left: 'prev,next today',
              center: 'title',
              right: 'dayGridMonth,timeGridWeek,timeGridDay',
            }}
            initialView="dayGridMonth"
            events={this.calendarEvents}
            editable={true}
            selectable={false}
            selectMirror={false}
            droppable={false}
            eventClick={this.openDetailsModal}
            dateClick={this.openCreationModal}
            dayMaxEvents={4}
            eventContent={this.renderEventContent}
            eventTimeFormat={timeFormat}
            locale={uaLocale}
          />
```

```
        </div>
      </>
    )
  }

  private get calendarEvents(): EventSourceInput {
    const { sessions, groups } = this.props

    return sessions.map((s) => {
      const startDate = new Date(s.date * 1000)
      return {
        title: 'Тренування',
        start: startDate,
        end: startDate.setHours(startDate.getHours() + 1),
        id: s.id,
        borderColor: groups.get(s.groupId)?.color,
        extendedProps: { status: s.status },
      } as EventInput
    })
  }

  private renderEventContent = (eventInfo: EventContentArg) => {
    return <EventContent eventInfo={eventInfo} />
  }

  @autobind
  private openCreationModal(arg: DateClickArg): void {
    this.setState({ clickedDate: arg.date })
  }

  @autobind
  private openDetailsModal(arg: EventClickArg): void {
    // TODO: pass only an id then get the session.
    const session = this.props.sessions.find((s) => s.id === arg.event.id)
    this.setState({ selectedSession: session })
  }

  @autobind
  private closeCreationModal(): void {
    this.setState({ clickedDate: undefined })
  }

  @autobind
  private closeDetailsModal(): void {
    this.setState({ selectedSession: undefined })
  }
}

export default compose(
  withStyles(styles),
  connectToState,
)(TrainingCalendarBase) as React.ComponentType
```

TrainingCalendar.types.ts

```
import { WithStyles } from '@material-ui/core'
import { IGroup } from '../../entities/IGroup'
import { ITrainingSession } from '../../entities/ITrainingSession'
import { IUser } from '../../entities/IUser'
```

```
import { styles } from './TrainingCalendar.styles'

type TrainingCalendarStyles = typeof styles

export interface TrainingCalendarBaseProps {
  users: IUser[]
  sessions: ITrainingSession[]
  groups: Map<string, IGroup>
  isFetching: boolean
}

export interface TrainingCalendarStateProps {
  clickedDate?: Date
  selectedSession?: ITrainingSession
}

export type TrainingCalendarProps = TrainingCalendarBaseProps &
  WithStyles<TrainingCalendarStyles>
```