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

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: «Мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики»

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

Виконавець роботи: студент групи ІТ.м-12 Малиновський Богдан Юрійович

Кваліфікаційну роботу захищено на засіданні ЕК з оцінкою _____ «__» грудня 2022 р.

Науковий керівник _____ к.т.н., доц., Антипенко В.П.
(підпис)

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

Студент _____
(підпис)

Суми-2022

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

ЗАТВЕРДЖУЮ
В.о. зав. кафедри ІТ

_____ С. М. Ващенко
«___» _____ 2022 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра студентові

Малиновський Богдан Юрійович
(прізвище, ім'я, по батькові)

1 Тема проекту *Мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики*

затверджена наказом по університету від «04» листопада 2022 р. № 01013-VI

2 Термін здачі студентом закінченого проекту «___» ___грудня___ 2022 р.

3 Вхідні дані до проекту _____

4 Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) аналіз предметної області, постановка задачі та методи дослідження, моделювання мобільного додатку, розробка мобільного додатку.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) актуальність роботи, постановка задачі, аналіз програмних продуктів – аналогів, IDEF0-модель, діаграма варіантів використання, демонстрація програми

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

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

Дата видачі завдання _____.

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів випускної проекту	Термін виконання етапів проекту	Примітка
1	Ознайомлення з предметною областю	22.08.22 – 22.08.22	
2	Визначення в потребі системи	23.08.22 – 29.08.22	
3	Ідентифікація ідеї проекту	30.08.22 – 31.08.22	
4	Аналіз документації	01.09.22 – 07.09.22	
5	Визначення вимог	08.09.22 – 09.09.22	
6	Визначення інструментарію	12.09.22 – 13.09.22	
7	Планування WBS, OBS	14.09.22 – 19.09.22	
8	Складання календарного плану	20.09.22 – 20.09.22	
9	Визначення бюджету	21.09.22 – 26.09.22	
10	Визначення ризиків	27.09.22 – 29.09.22	
11	Моделювання роботи додатку	30.09.22 – 04.10.22	
12	Розробка макету додатку	05.10.22 – 13.10.22	
13	Розробка функціоналу додатку	14.10.22 – 10.11.22	
14	Тестування	11.11.22 – 16.11.22	
15	Підготовка до завантаження	17.11.22 – 17.11.22	
16	Завантаження	18.11.22 – 18.11.22	

Магістрант _____ Малиновський Б.Ю.

Керівник роботи _____ к.т.н., доц. Антипенко В.П.

РЕФЕРАТ

Тема кваліфікаційної роботи магістра «Мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики».

Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел із 18 найменувань, додатків. Загальний обсяг роботи − 100 сторінок, у тому числі 39 сторінок основного тексту, 2 сторінки списку використаних джерел, 59 сторінок додатків.

Кваліфікаційну роботу магістра присвячено розробці мобільного додатку для аналізу результатів тренувань спортсменів з легкої атлетики.

У першому розділі наведено огляд останніх досліджень за темою роботи та проведено аналіз програмних продуктів-аналогів.

У другому розділі описано мету, задачі, метод дослідження та вибір технологій.

У третьому розділі представлено моделювання «Мобільного додатку для аналізу результатів тренувань спортсменів з легкої атлетики», розроблено контекстну діаграму IDEF0 та діаграму варіантів використання. Проведено проектування моделі бази даних та інформаційної бази.

У четвертому розділі детально описано етапи практичної реалізації проекту та наведено приклади використання створеного програмного додатку, які демонструють його працездатність.

Ключові слова: МОБІЛЬНИЙ ДОДАТОК, ЛЕГКА АТЛЕТИКА, РОЗРЯД, ПРОГНОЗУВАННЯ, ANDROID, KOTLIN.

ЗМІСТ

ВСТУП

Для кожного спортсмена важливо спостерігати за його прогресом у тренуваннях. У такому виді спорту, як легка атлетика, досить просто порівнювати проміжні результати. Але це варто робити постійно і, базуючись на отриманих результатах, розробляти програму подальших тренувань.

Актуальність. Професійним спортсменам набагато легше проводити заняття. Вони мають тренера, який складає їм програму та слідкує за їх роботою. Але є частка спортсменів, які з певних причин займаються та виступають на змаганнях самостійно. Це ускладнює відстеження та аналіз їх прогресу. Рішенням для таких спортсменів стане програмне забезпечення, яке буде в змозі замінити їм повністю або частково функції тренера. У такому випадку вони зможуть без допомоги сторонньої людини відслідковувати власний прогрес. Також доступною буде можливість прогнозувати подальші результати та, на базі цієї інформації, розробляти власну програму тренувань. Так як спортсменам важливо заносити результати занять одразу після їх виконання, доцільно розробити саме мобільний додаток. Доступ до нього буде як під час тренувань, так і до або після них.

Тому **мета даної роботи** – це розробка мобільного додатку для аналізу результатів тренувань спортсменів з легкої атлетики. Створюваний програмний продукт призначений для прогнозування можливих досягнень із інших нормативів.

Для досягнення мети даного проекту необхідно виконати такі задачі:

– проаналізувати предметну область;

– виконати огляд останніх досліджень і публікацій;

– провести дослідження програмних продуктів-аналогів;

– скласти сценарій роботи мобільного додатку для аналізу результатів тренувань спортсменів із легкої атлетики;

– провести моделювання роботи програмного забезпечення;
– виконати програмну реалізацію запропонованого мобільного додатку;
– здійснити тестування даного програмного продукту.

Об’єкт дослідження. Процес прогнозування результатів спортсменів із легкої атлетики.

Предмет дослідження. Інформаційні технології для прогнозування результатів спортсменів із легкої атлетики.

Практичне значення. Розроблений мобільний додаток допоможе користувачу передбачити, які він отримає результати з визначених нормативів із бігу на основі поточних даних, а також пропонуватиме йому програми тренувань для різних дистанцій та допомагатиме контролювати, при яких умовах буде досягнуто наступних розрядів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд останніх досліджень і публікацій

Із кожним роком інформаційні технології (ІТ) впроваджуються у все більшу кількість сфер діяльності. На сьогоднішній день важко представити лекцію без мультимедійної презентації або здачу іспиту з правил дорожнього руху без комп’ютера. Спорт не є виключенням. У сучасному суспільстві ІТ можуть відігравати також важливу роль і в даній сфері.

У [1] описуються напрямки використання інформаційно-комп’ютерних технологій (ІКТ) у спорті та фізичній культурі, що пов'язане з підвищенням рівня складності усіх тренувальних процесів. А, отже, можна зазначити, що використання ІТ є дійсно актуальним у зазначеній області. Також у статті наголошується на тому, що використання ІКТ, а саме автоматизованих систем і прикладних програмних продуктів, які спрощують керування тренувальним процесом, створює абсолютно нові можливості для розвитку спорту.

У [2] описано як саме сучасні комп’ютерні технології дозволяють коригувати тренувальний процес, що впливає на підвищення спортивних результатів. Також у статті відзначено, що науковці приділяють увагу підвищенню якості технічної підготовленості спортсменів, для чого створюються програмно-апаратні комплекси, які автоматизують введення даних у комп’ютер і обчислення необхідних біомеханічних параметрів, що дає змогу підвищити ефективність навчання рухових дій і не допускати помилок.

У результаті проведеного огляду можна зробити висновок, що , процес залучення інформаційних технологій у тренувальний процес досить активний. Це підтверджує актуальність розробки мобільного додатку для аналізу результатів занять спортсменів з легкої атлетики.

1.2 Аналіз програмних продуктів-аналогів

На сьогоднішній день усе частіше використовують різноманітні мобільні додатки для моніторингу та аналізу результатів із бігу. Основна їх частина представляє собою програмні засоби (ПЗ), які зберігають дані після забігів та пропонують різноманітні тренувальні програми. Прикладами таких застосунків є «Runkeeper» [3] та «C25K 5K Trainer» [4].

Окрім базових функцій додатків для бігу, таких як зберігання результатів, GPS моніторинг, відстеження темпу та часу, ПЗ «Runkeeper» також має персоналізовані програми тренувань із корисними нагадуваннями для спортсменів, які готуються до змагань.

Мобільний застосунок «C25K 5K Trainer» створений для легкоатлетів, які планують долати дистанцію 5000 метрів. Більше того, його використання може допомогти досягти позитивних результатів, починаючи з нульового рівня підготовки. Навіть за досить короткий проміжок часу, але у разі регулярних тренувань.

Досить багато існує додатків із переліченими вище функціями, але мало з них аналізують результати, порівнюючи їх з нормативами з кваліфікаційної таблиці. Це є значно важливим для спортсменів, які професійно займаються легкою атлетикою. Також такі стандартні застосунки зазвичай не можуть прогнозувати результат з нормативу, базуючись на даних на інших дистанціях. Тому доцільно створити мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики, направленого на прогнозування можливих досягнень із інших нормативів, у якому будуть реалізовані вище зазначені функції.

2 ПОСТАНОВКА ЗАДАЧІ ТА МЕТОДИ ДОСЛІДЖЕННЯ

2.1 Мета та задачі дослідження

Метою роботи є розробка мобільного додатку для аналізу результатів тренувань спортсменів з легкої атлетики. Створений програмний продукт буде мати попит серед легкоатлетів, які тренуються та виступають на змаганнях самостійно. Використовуючи даний додаток спортсмени зможуть зберігати результати після занять, робити їх аналіз та прогнозувати варіанти можливих досягнень для інших дистанцій, що зменшить витрачання їхнього часу та зусиль.

Розроблений мобільний додаток повинен містити наступні функції:

– робота секундоміра;

– збереження результатів забігів;

– перегляд результатів забігів;

– редагування результатів забігів;

– видалення результатів забігів;

– виведення інформації про встановлений час для досягнення наступного розряду для кожного з основних нормативів з кваліфікаційної таблиці;

– виведення програм тренувань для різних дистанцій;

– прогнозування результату нормативу, базуючись на результаті інших нормативів;

– виведення кваліфікаційної таблиці.

Для досягнення мети проекту потрібно виконати такі задачі:

– проаналізувати предметну область;

– виконати огляд останніх досліджень і публікацій;

– провести дослідження програмних продуктів-аналогів;

– скласти сценарій роботи мобільного додатку для аналізу результатів тренувань спортсменів із легкої атлетики;

– провести моделювання роботи програмного забезпечення;

– виконати програмну реалізацію запропонованого мобільного додатку;

– здійснити тестування даного програмного продукту.

Сценарій роботи користувача із запропонованим мобільним додатком є досить простим для розуміння. Початковою сторінкою є «Головна». Вона відображає секундомір. Під ним є кнопка «Зберегти забіг». Після натискання на неї відкривається сторінка «Новий забіг». На ній потрібно ввести всю необхідну інформацію про забіг та натиснути кнопку «Зберегти забіг». Після чого автоматично відкривається сторінка «Головна». Зліва мобільний додаток має меню. Використовуючи його, можна перейти на такі сторінки, як «Головна», «Мої результати», «Програми тренувань», «Прогноз подальших результатів» і «Кваліфікаційні нормативи». Перейшовши на «Мої результати», користувачу доступний список усіх збережених забігів, кожний з яких має функції редагування та видалення. Після натискання на кнопку редагування відкривається сторінка «Редагувати забіг». На ній в поля для вводу інформації введені поточні дані забігу, які можна змінити та зберегти їх, натиснувши на відповідну кнопку. Після цього користувач автоматично повертається на сторінку «Мої результати». Унизу списку результатів є кнопка для додавання забігу, на яку користувач має натиснути, щоб зберегти новий результат. Під списком результатів є секція, де відображається інформація про найближчий наступний розряд, під якою є кнопка «Переглянути для всіх розрядів», при натисненні на яку відкривається сторінка «Коли наступний розряд». На ній відображається таблиця з інформацією про наступний розряд для кожного нормативу. Після вибору пункту меню «Програми тренувань» користувачу відображається список дистанцій. При натисненні на кожну з них відображається відповідна програма тренувань. На сторінці «Прогнозування результатів» відображається список дистанцій. При натисненні на кожну з них відображається

спливаючий діалог з прогнозом результату, базуючись на результатах інших нормативів. На екрані «Кваліфікаційна таблиця» користувачу відображається таблиця з усіма значеннями нормативів.

Планування виконання робіт даного проекту представлено у додатку А.

2.2 Методи дослідження

Так як основною задачею мобільного додатку є аналіз результатів тренувань спортсменів з легкої атлетики для прогнозування можливих досягнень із інших нормативів, було досліджено наявні методи передбачення результатів із бігу. Було знайдено один, який відповідає поставленій задачі. Це метод прогнозування часу змагань для бігунів Пітера Рігеля, або як його частіше називають – формула Рігеля [5].

Формула Рігеля дозволяє передбачити час змагань для бігунів та інших спортсменів, які демонструють певні результати на іншій дистанції. Із роками вона отримала широке схвалення спортивної спільноти завдяки високій точності розрахунків і простоті застосування.

Формула Рігеля (2.1) має наступний вигляд:

$$T_2 = T_1(\frac{D_2}{D_1})^{1.06} \tag{2.1}$$

T_1 – нещодавно досягнутий час на відстані D_1

T_2 – прогнозований час для відстані D_2

D_1 – відстань, на якій було досягнуто часу T_1

D_2 – відстань, для якої передбачено час T_2

Перш ніж використовувати формулу Пітера Рігеля, варто звернути увагу на наступні моменти:

– припускається, що бігун пройшов відповідну підготовку для дистанції, яку він хоче пробігти;

– припускається, що спортсмен не має значної природної схильності ні до швидкості, ні до витривалості;

– обчислення менш точні для часового результату менше 3,5 хвилин і більше 4 годин.

Велика перевага цієї формули полягає в тому, що вона налаштована на відстань – вона не просто подвоює, наприклад, прогноз 6 км для прогнозу 12 км.

2.3 Вибір технологій

Після того, як був проведений аналіз актуальності роботи та визначений тип програмного забезпечення, потрібно визначити технології для реалізації даного продукту.

Так як мобільних пристроїв з операційною системою Android більше за рейтингом gs.statcounter.com [6], у порівнянні з іншими, тому було прийнято рішення розробити мобільний додаток саме під платформу Android.

Для реалізації даного проекту було обрано мову програмування Kotlin [7]. Вона придатна для розробки під платформу Android. Мова програмування Kotlin є нащадком стандартної мови для створення мобільних додатків Java. Вона має низку переваг [8] над своєю попередницею. Тому мова програмування Kotlin є вибором великої кількості професіоналів, працюючих із додатками даного типу.

Серед середовищ розробки було обрано Android Studio [9]. Дане програмне забезпечення вважається одним із найкращих серед конкуруючих аналогів, що використовуються для створення Android продуктів. У ньому підтримується розробка мобільних додатків мовою Kotlin, присутній емулятор для тестування останніх у процесі розробки та система контролю версій Git.

3 МОДЕЛЮВАННЯ ТА ПРОЕКТУВАННЯ

3.1 Моделювання «Мобільного додатку для аналізу результатів тренувань спортсменів з легкої атлетики»

Наступним кроком є виконання етапу моделювання та проектування мобільного додатку. Були розроблені діаграми нотації IDEF0 з описаними послідовностями процесу прогнозування нового результату для бажаної дистанції, та діаграма Use-Case з демонстрацією наявних функцій, які користувач може використовувати, застосовуючи даний мобільний додаток.

3.1.1 Діаграми нотації IDEF0

Нотація IDEF0 призначена для опису бізнес-процесів. Для кращого представлення останніх проект представляється у вигляді прямокутника [10]. Керуючі стрілки під’єднуються до його сторін. Кожна стрілка повинна відповідати за різний наступний тип даних:

– ліва стрілка – вхідні дані системи;

– права стрілка – вихідні дані системи;

– верхня стрілка – дані керування, тобто документ, що фіксує, як система повинна працювати та реалізовуватися;

– нижня стрілка – дані механізму, тобто персонал або програмне забезпечення, які залучаються для реалізації системи.

Процес «Отримання спрогнозованого результату забігу бажаного нормативу» містить такі дані:

– вхідні дані – результат забігу по одному з нормативів.

– вихідні дані – спрогнозований результат забігу бажаного нормативу.

– управління – кваліфікаційна таблиця [11], метод прогнозування часу змагань для бігунів Пітера Рігеля.

– механізми – мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики, виконавець та технічне забезпечення.

На рисунку 3.1 представлена контекстна діаграма процесу отримання спрогнозованого результату забігу бажаного нормативу.

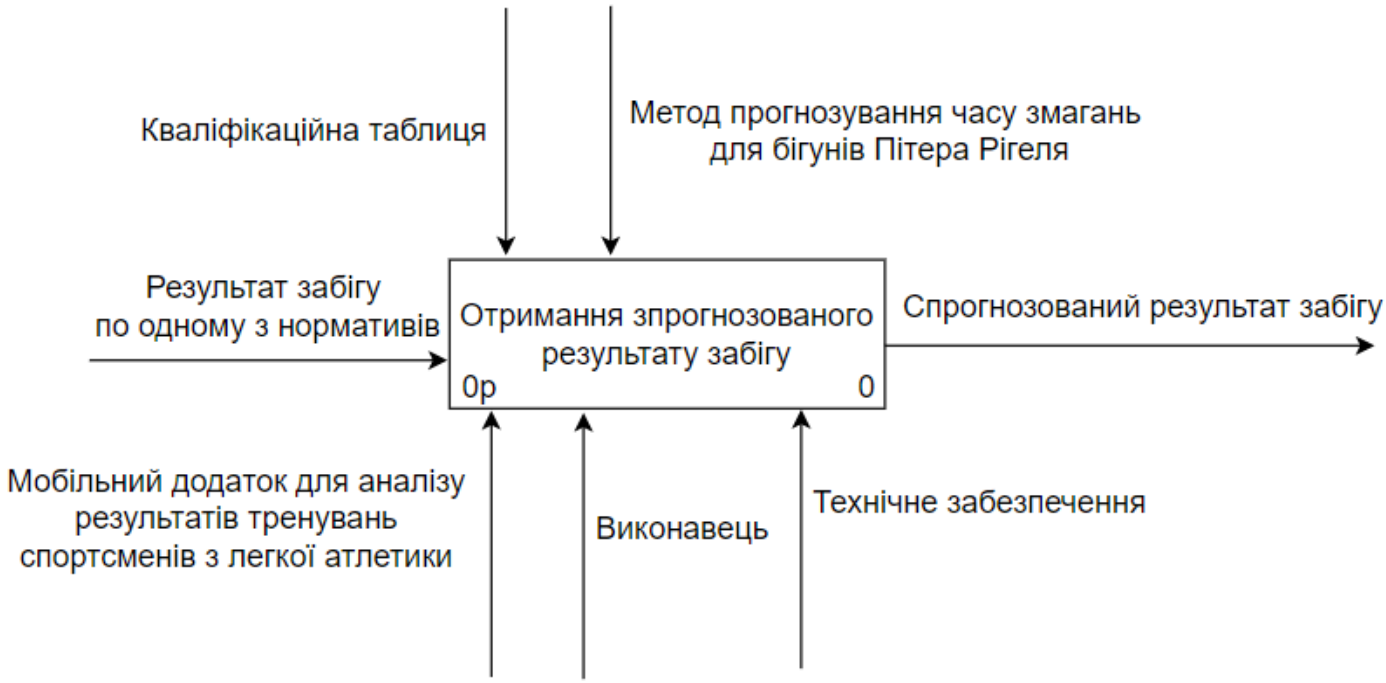


Рисунок 3.1 – Контекстна діаграма нотації IDEF0

Наступний крок – декомпозиція реалізації системи на такі два головних процеси:

– збереження отриманого результату забігу по одному з нормативів;

– прогнозування результату забігу бажаного нормативу.

Після декомпозиції, розроблено діаграму першого рівня. Дані для діаграми наступні:

– вхідні дані – результат забігу по одному з нормативів;

– вихідні дані – спрогнозований результат забігу бажаного нормативу;

– управління – кваліфікаційна таблиця, метод прогнозування часу змагань для бігунів Пітера Рігеля;

– механізми – мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики, виконавець та технічне забезпечення.

На рисунку 3.2 представлена діаграма першого рівня.

Рисунок 3.2 – Діаграма першого рівня

3.1.2 Діаграма Use Case

У діаграмі Use Case проводиться опис взаємодії користувача з системою [12]. Було визначено одного актора – користувач. Нижче відображено список сценаріїв використання ним системи:

– зберігати результати;

– працювати зі збереженими результатами;

– отримувати прогноз подальших результатів;

– переглядати скільки залишилося до наступного розряду;

– переглядати програми тренувань для різних дистанцій;

– переглядати кваліфікаційну таблицю.

Також було визначено 2 артефакти:

– інформаційна база;

– база даних.

Розроблена діаграма Use Case на основі даних сценаріїв представлена на рисунку 3.3.

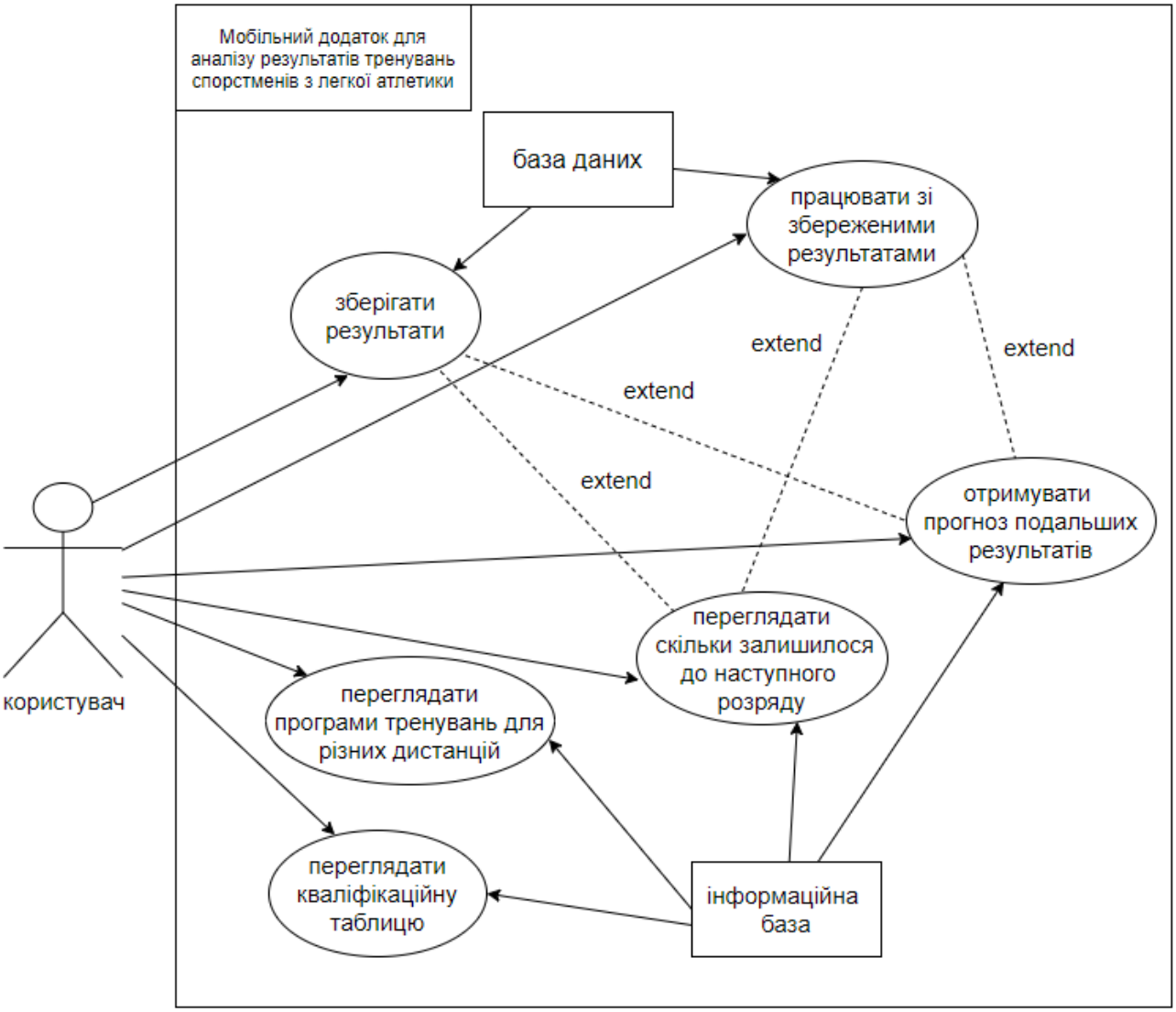


Рисунок 3.3 – Діаграма Use Case

3.2 Проектування моделі бази даних

Модель даних – це система представлення для моделювання структури фізичної системи, яка застосовується для створення баз даних (БД) [13].

БД даного мобільного додатку має лише одну модель даних – забіг, атрибутами якого є дистанція, часовий результат, дата та час забігу, нотатка.

На рисунку 3.4 представлена ER-діаграма бази даних.

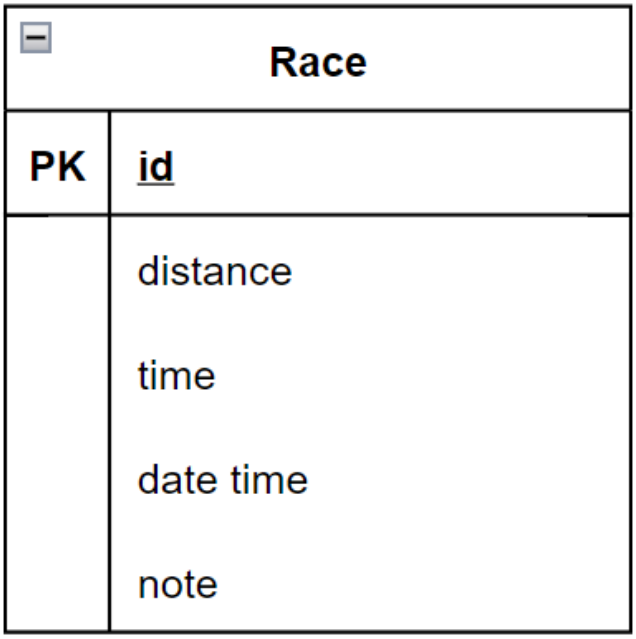


Рисунок 3.4 – Структура бази даних

3.3 Проектування моделі інформаційної бази

Окрім даних, які можуть редагуватися, запропонований мобільний додаток має також такі, які не повинні змінюватися або видалятися. Це значення кваліфікаційної

таблиці з усіма значеннями нормативів, які доступні у даному програмного продукту. Тому було вирішено додати інформаційну базу у вигляді класу-репозиторію, де всі дані представлені у вигляді статичного списку.

Аналіз предметної області інформаційної бази є наступним:

– об’єкт – норматив кваліфікаційної таблиці;

– атрибути об’єкта: дистанція, результат для третього юнацького розряду, результат для другого юнацького розряду, результат для першого юнацького розряду, результат для третього розряду, результат для другого розряду, результат для першого розряду, результат для розряду кандидата у майстра спорту, результат для розряду майстра спорту, результат для розряду майстра спорту міжнародного класу, тип нормативу за гендерним фактором.

На рисунку 3.5 представлена ER-діаграма інформаційної бази.

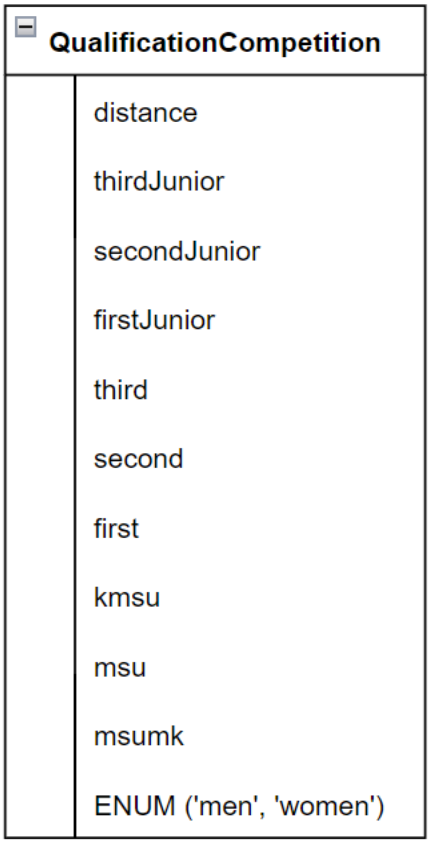


Рисунок 3.5 – Структура інформації інформаційної бази

4 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

4.1 Програмна реалізація

Програмна реалізація мобільного додатку виконана мовою програмування Kotlin у середовищі Android Studio. Навігація між фрагментами додатку реалізована за допомогою Navigation component (рис. 4.1) [14].

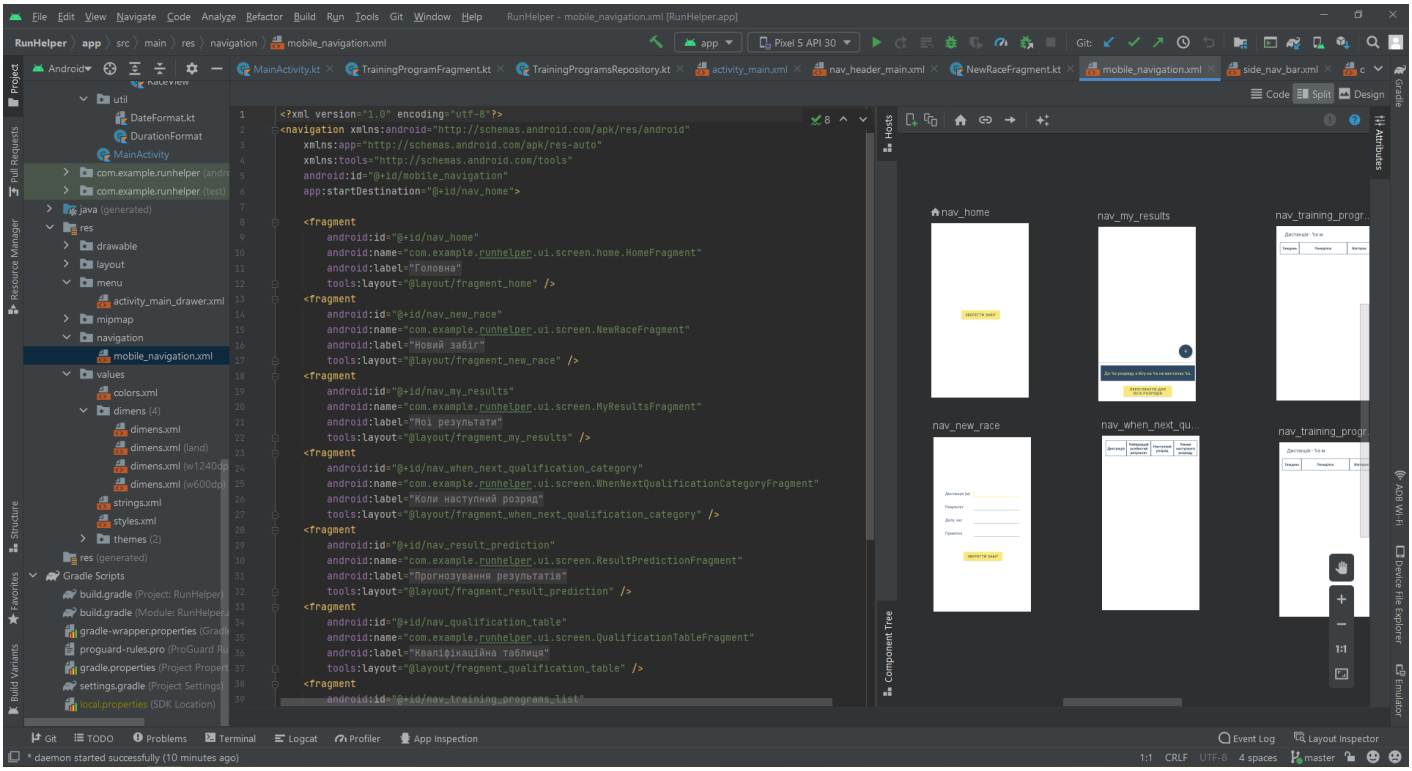


Рисунок 4.1 – Файл опису графу навігації

Для реалізації роботи з БД використано бібліотеку Room [15]. Виконання запитів у базу даних асинхронно реалізовано за допомогою kotlin Coroutines [16]. Таблиці у БД генеруються за допомогою Entity класів [15]. Для зберігання даних розроблена одна таблиця Race (рис. 4.2).

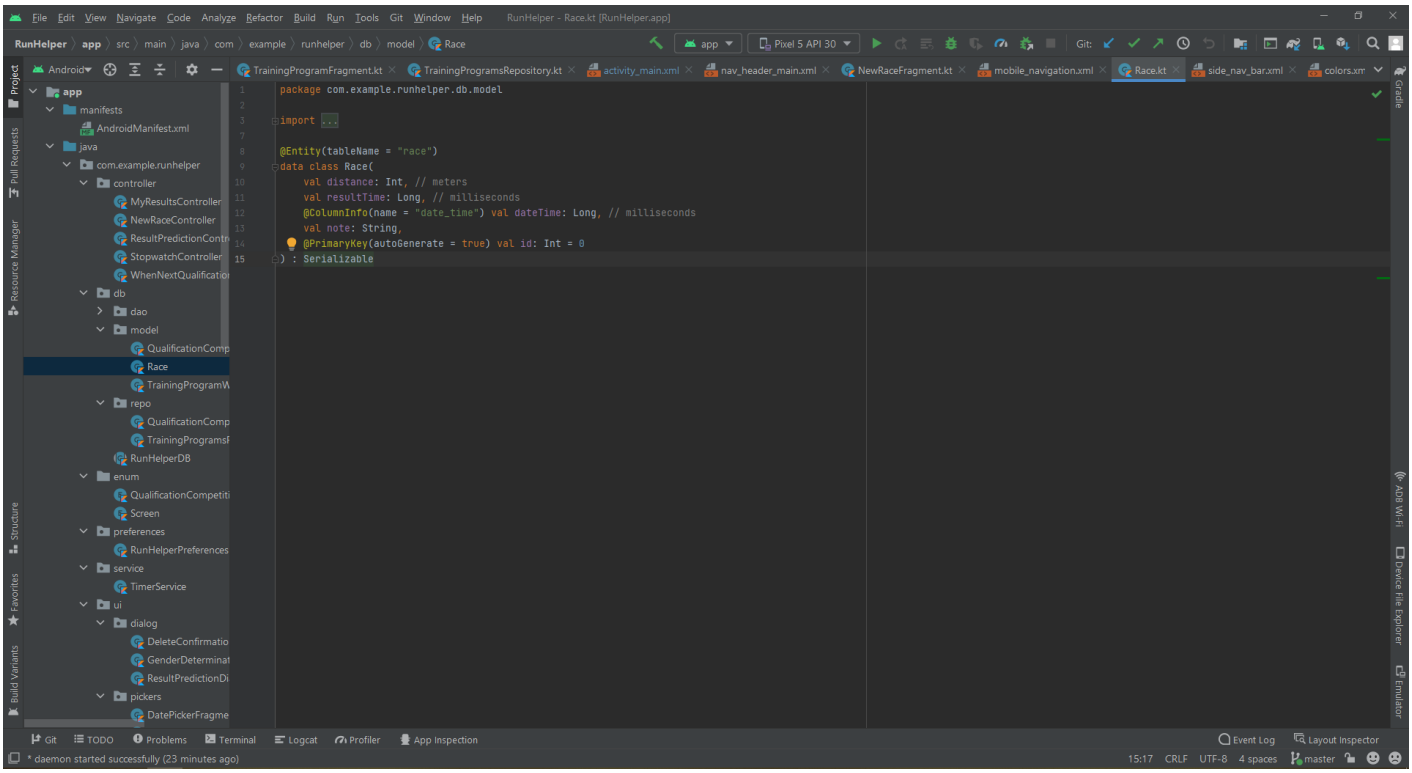


Рисунок 4.2 – Entity клас Race

Зчитування та запис даних до БД виконано за допомогою інтерфейсів типу Dao, в яких описано методи запитів до бази даних (рис. 4.3) [15].

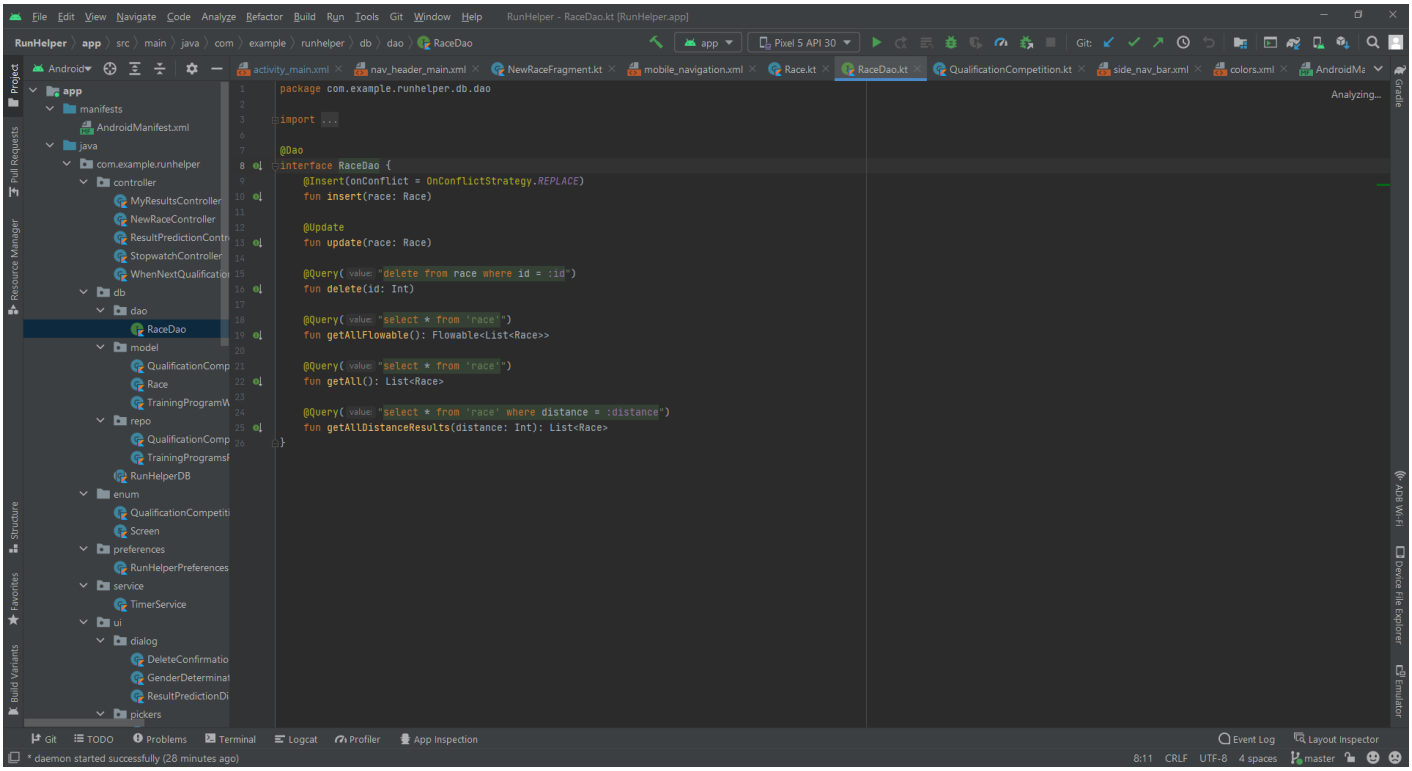


Рисунок 4.3 – Dao інтерфейс Race

Інтерфейс даного мобільного додатку розроблено, використовуючи мову розмітки XML (рис. 4.4).

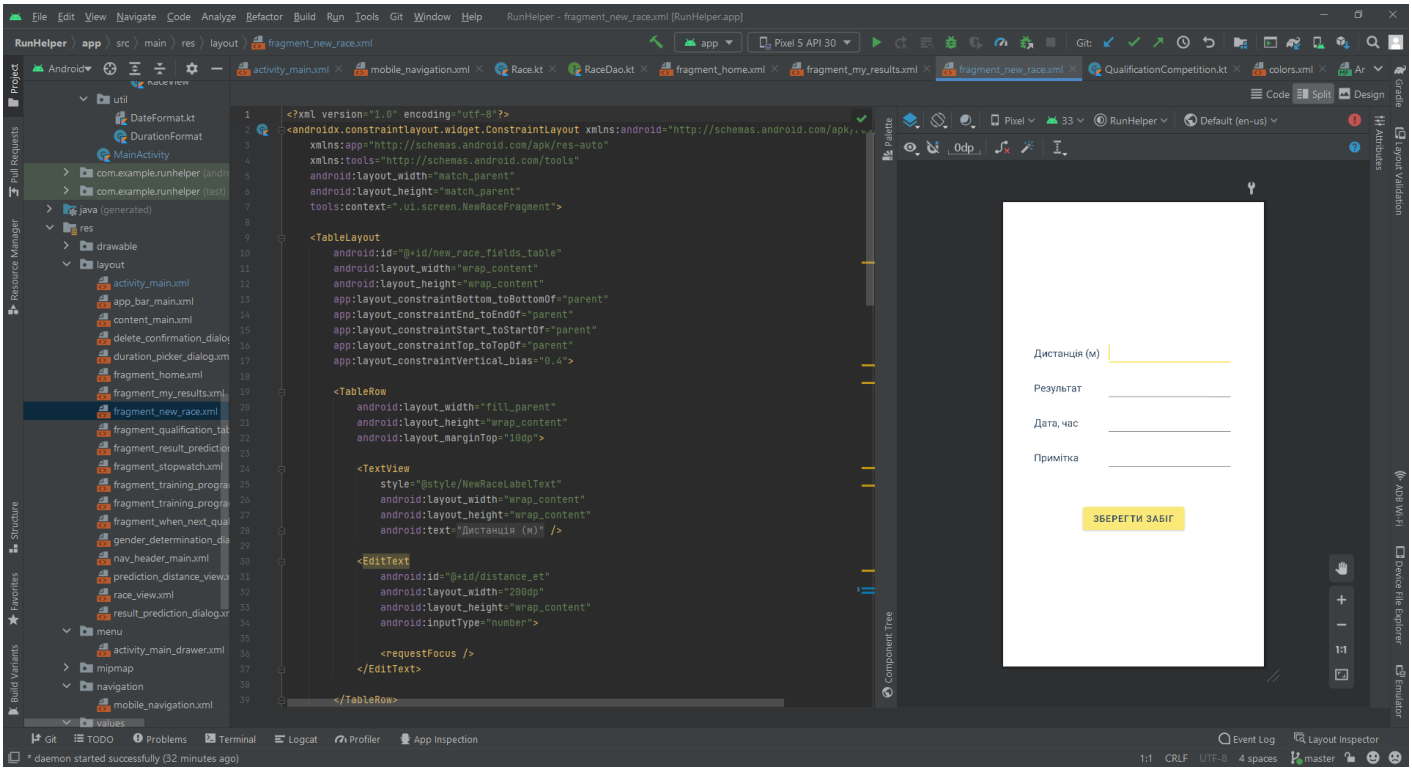


Рисунок 4.4 – Розмітка екрану «Новий забіг»

Лістинг основних програмних модулів наведено у додатку Б.

4.2 Використання додатку

Після першого завантаження мобільного додатку, користувачу відображається спливаючий діалог для вибору категорії. Саме по ній для спортсмена будуть визначатися кваліфікаційні нормативи з легкої атлетики (рис. 4.5). Вибір категорії зберігається, тому користувачу не потрібно буде знову її обирати при наступному завантаженні даного мобільного додатку. Після обрання відкривається початкова сторінка –

«Головна» (рис. 4.6). Вона відображає секундомір. Під ним є кнопка «Зберегти забіг». Після натискання на неї відкривається екран «Новий забіг».

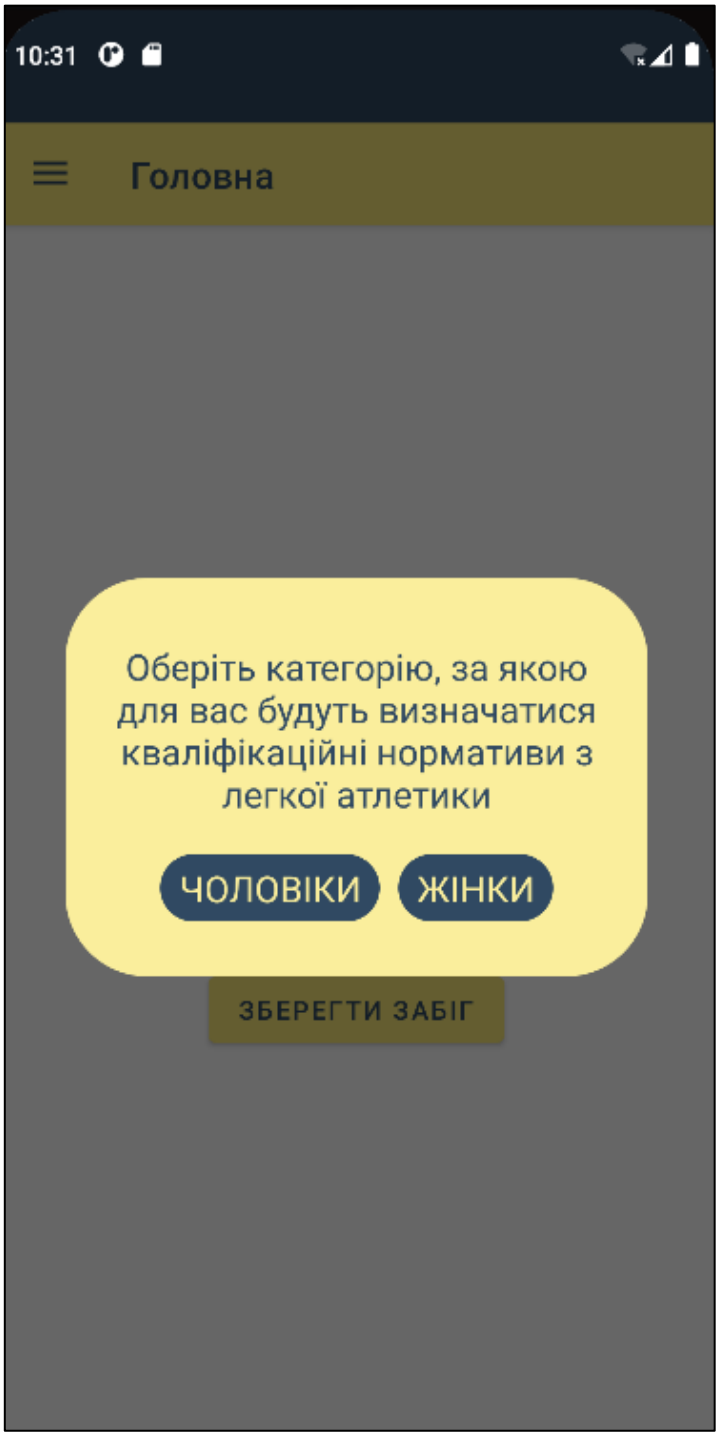


Рисунок 4.5 – Діалог для вибору категорії

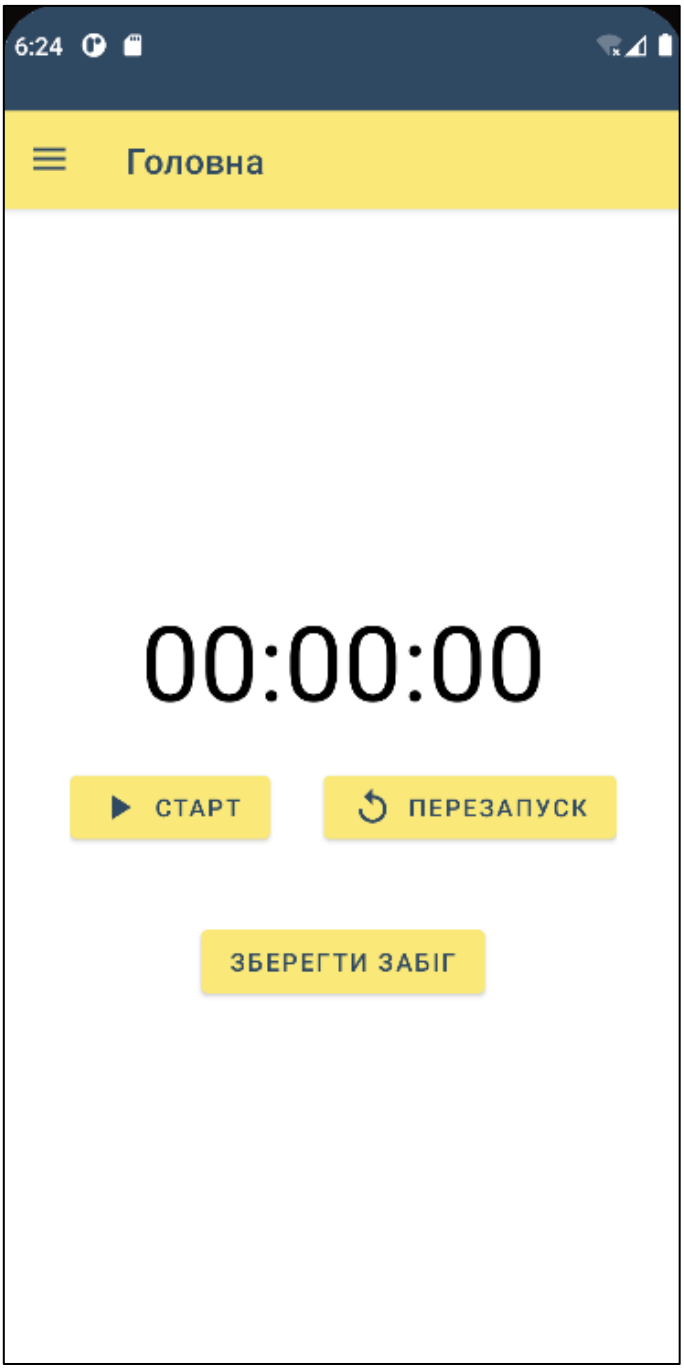


Рисунок 4.6 – Екран «Головна»

Створений мобільний додаток має бокове меню (рис. 4.7), з якого можна перейти на екрани «Головна», «Мої результати», «Прогнозування результатів», «Кваліфікаційна таблиця» та «Програми тренувань».

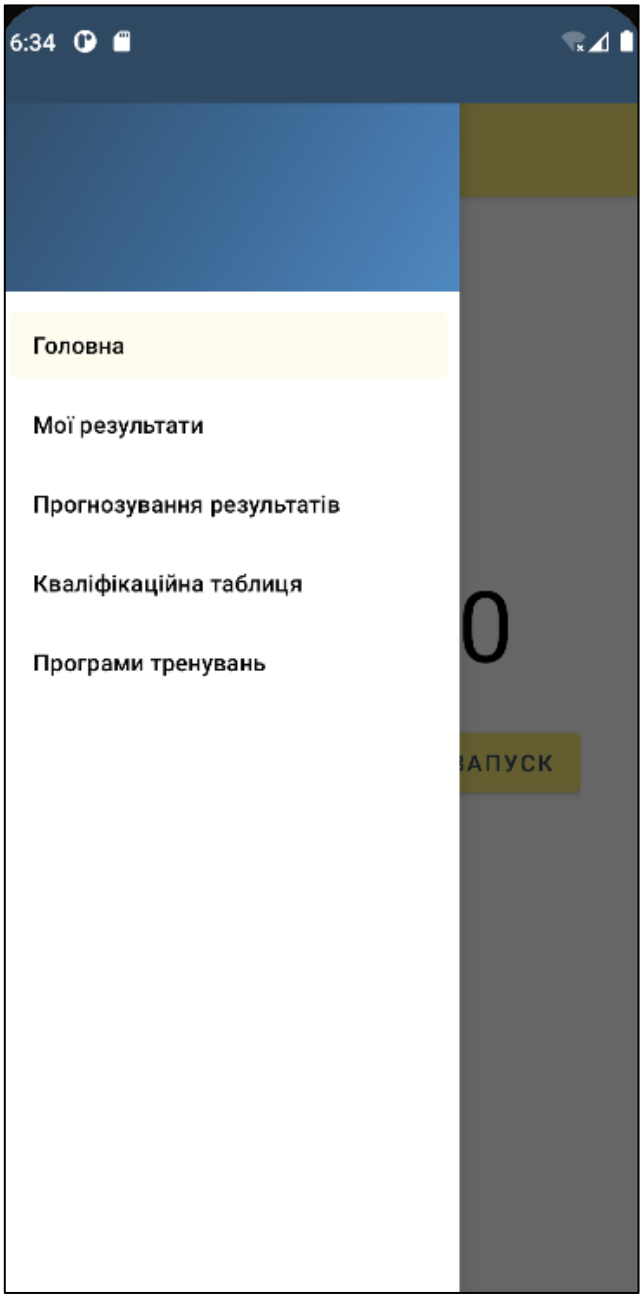


Рисунок 4.7 – Бокове меню мобільного додатку

Після натискання на кнопку «Зберегти забіг» відкривається екран «Новий забіг» (рис. 4.8). На ньому потрібно ввести всю необхідну інформацію про забіг. Це дистанцію, результат, дату й час тренування та примітку. Це необхідно для отримання подальшого прогнозування. При натисненні на поле вводу результату відобразиться спливаючий діалог для введення часу забігу (рис. 4.9). Це можна зробити в різних величинах. Дата й час тренування обираються також за допомогою DatePickerDialog (рис. 4.10) та TimePickerDialog (рис. 4.11). Вони по замовчуванню визначаються поточні. Якщо екран відкривається під час роботи секундоміра, то його дані автоматично відображається у полі вводу результату. Для збеоеження інформації забігу треба натиснути кнопку «Зберегти забіг». Після чого автоматично відкривається сторінка «Головна».

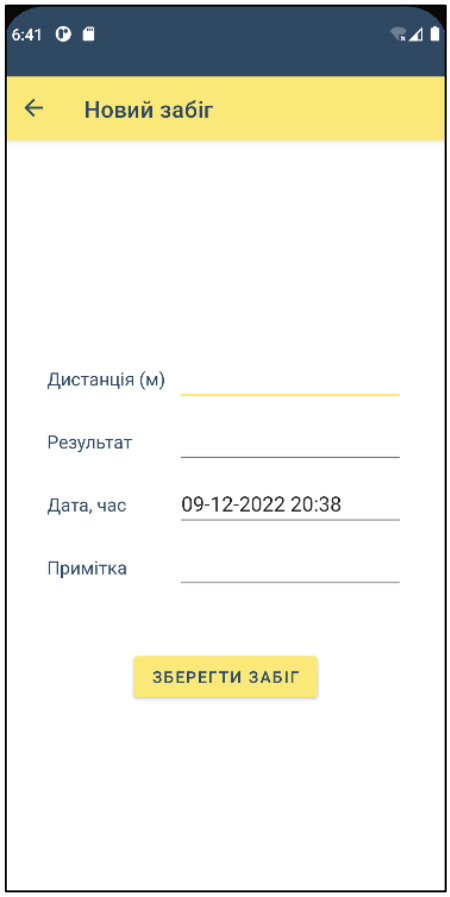


Рисунок 4.8 – Екран «Новий забіг»

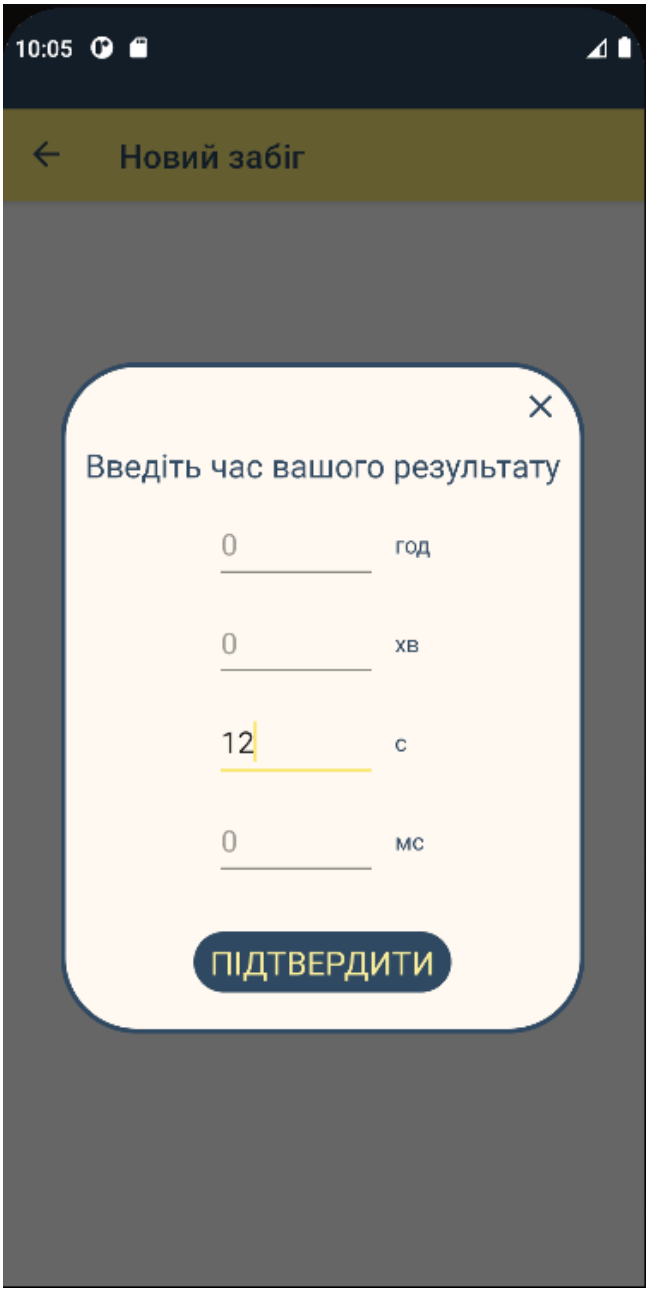


Рисунок 4.9 – Спливаючий діалог для введення часу забігу

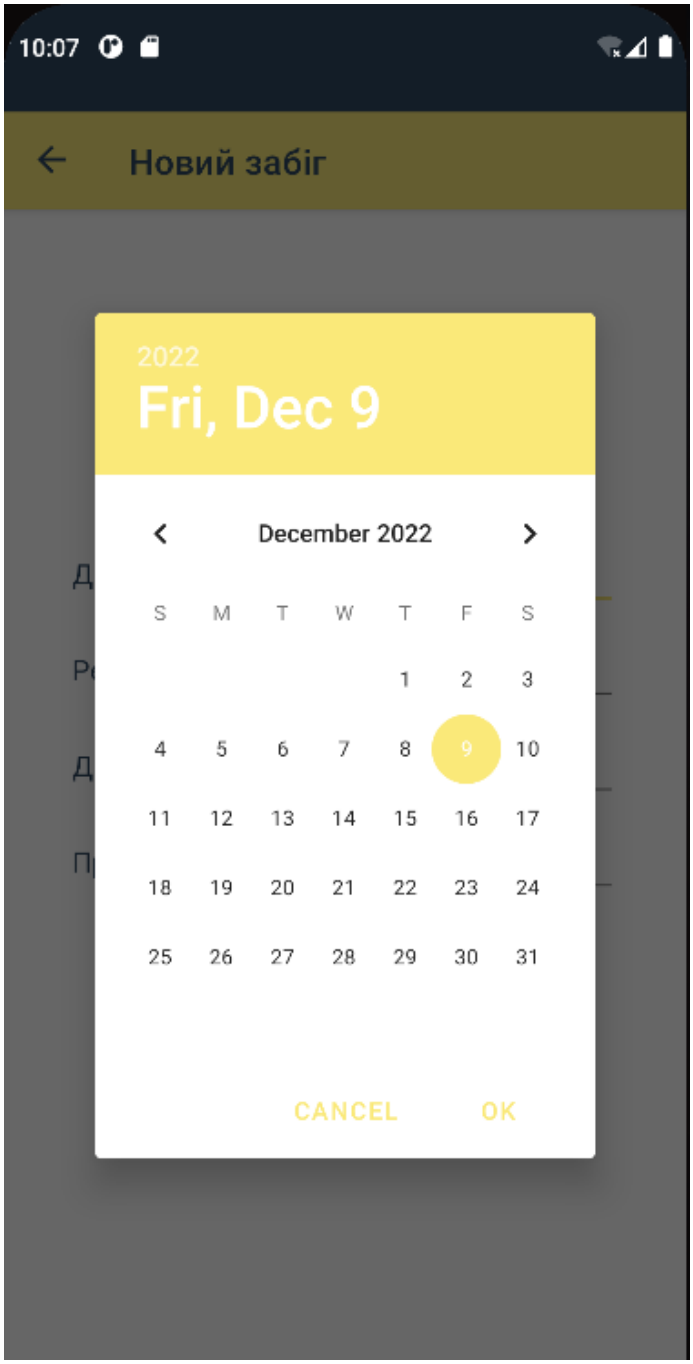


Рисунок 4.10 – DatePickerDialog

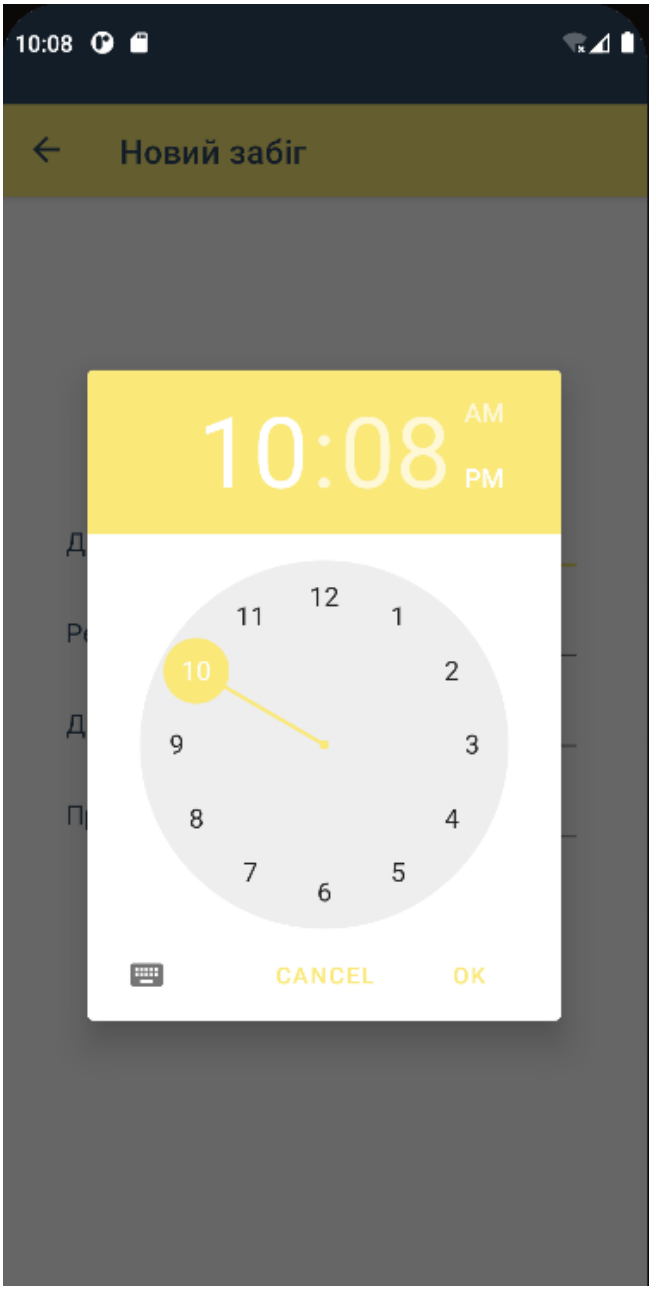


Рисунок 4.11 – TimePickerDialog

Перейшовши на екран «Мої результати», користувачу доступний список усіх збережених забігів, кожний з яких має функції редагування та видалення (рис. 4.12). Після натискання на кнопку редагування відкривається сторінка «Редагувати забіг». На ній в поля для вводу інформації внесено поточні дані забігу. Їх можна змінити та зберегти, натиснувши на відповідну кнопку. Після цього користувач автоматично

повертається на сторінку «Мої результати». Унизу списку результатів є кнопка для додавання забігу. Користувач має натиснути на неї, щоб зберегти новий результат. Також нижче там є наявною секція, де відображається інформація про найближчий наступний розряд. Під нею є кнопка «Переглянути для всіх розрядів», при натисненні на яку відкривається сторінка «Коли наступний розряд».

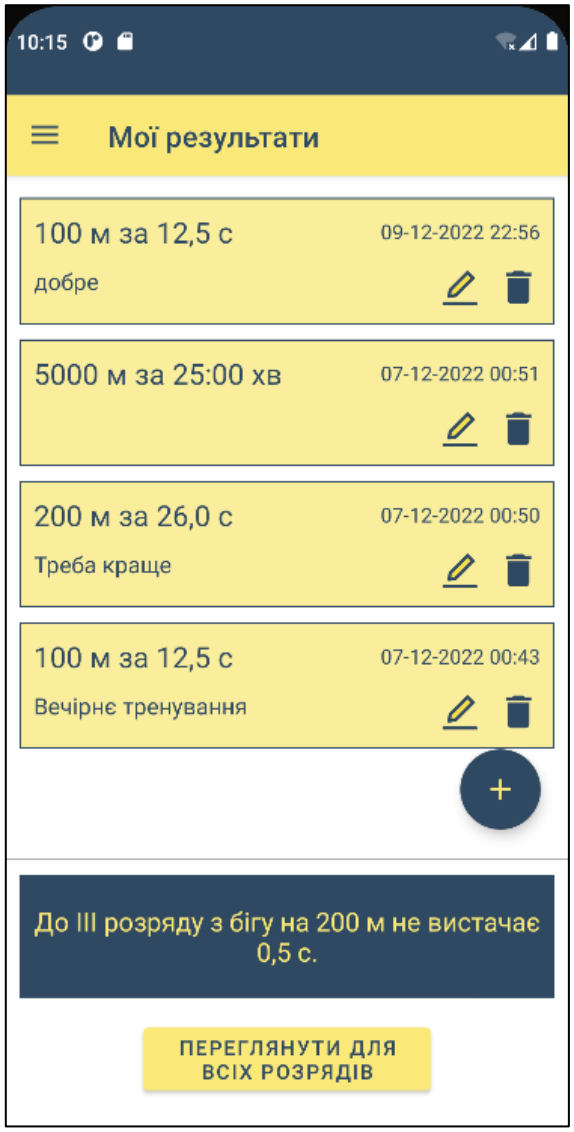


Рисунок 4.12 – Екран «Мої результати»

На сторінці «Коли наступний розряд» відображається таблиця з інформацією про наступний розряд для кожного нормативу (рис. 4.13).

Рисунок 4.13 – Екран «Коли наступний розряд»

Після вибору пункту меню «Програми тренувань» користувачу відображається список дистанцій (рис. 4.14). При натисненні на кожну з них надається відповідна програма тренувань (рис. 4.15). Останні взяті з таких джерел [17] і [18].

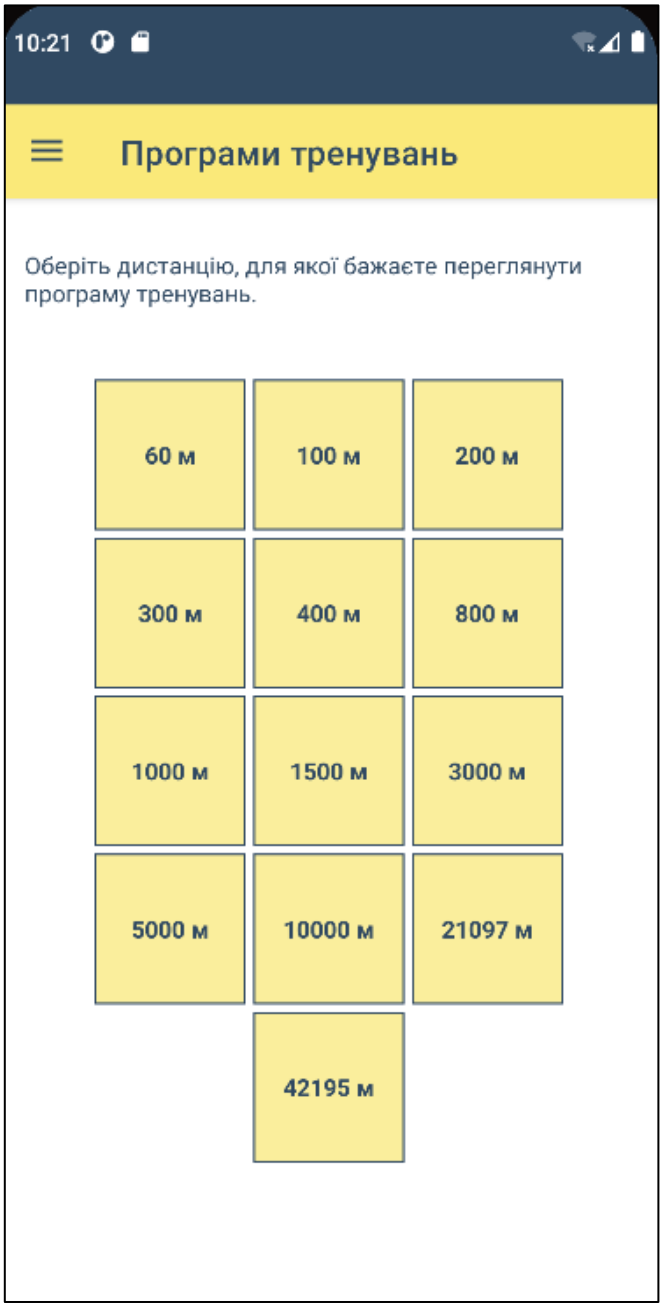


Рисунок 4.14 – Екран «Програми тренувань»

Рисунок 4.15 – Програма тренувань для визначеної дистанції

На сторінці «Прогнозування результатів» відображається список дистанцій (рис. 4.16). При натисненні на кожну з них стає доступним спливаючий діалог із прогнозом результату (рис. 4.17). Він базується на даних інших нормативів.

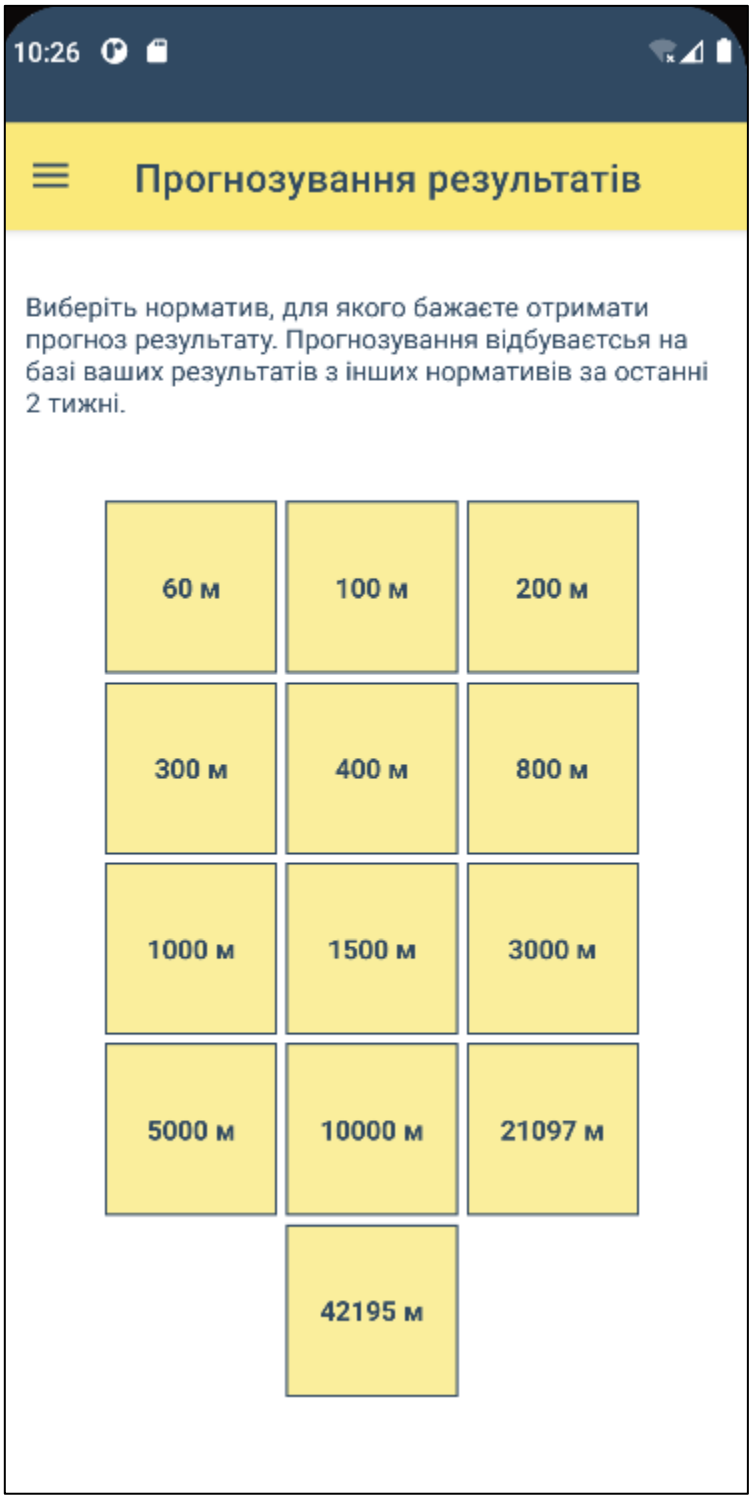


Рисунок 4.16 – Екран «Прогнозування результатів»

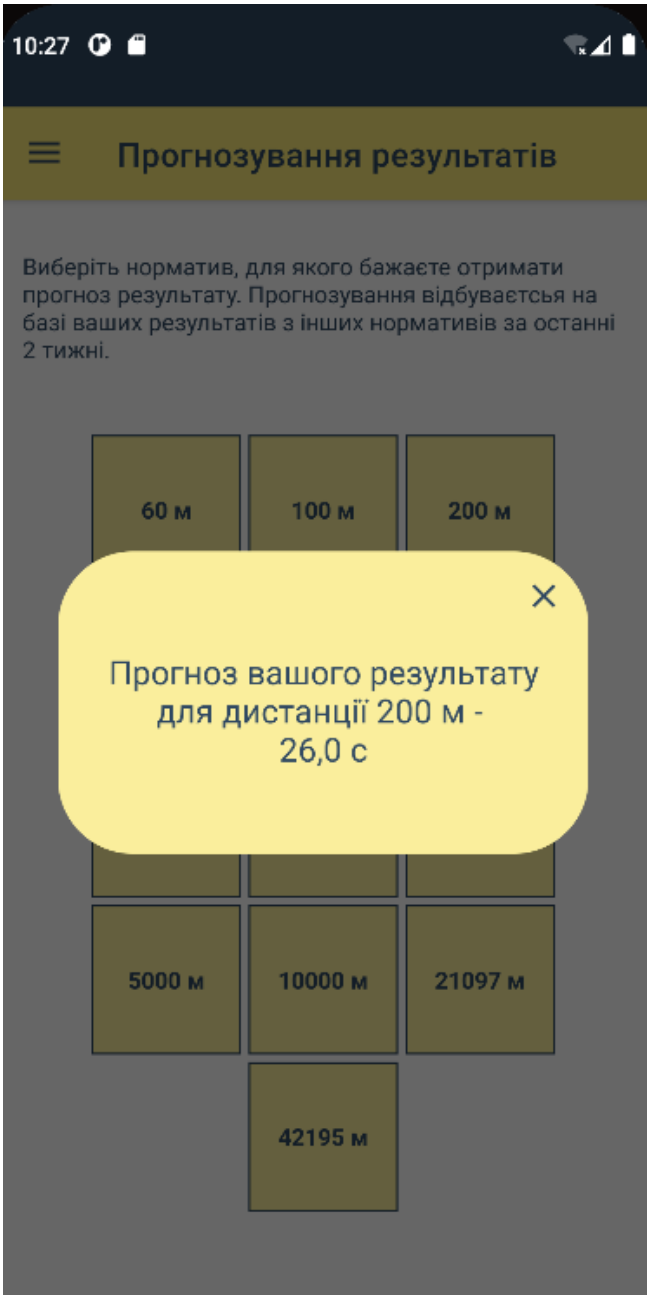


Рисунок 4.17 – Спрогнозований результат для визначеної дистанції

На екрані «Кваліфікаційна таблиця» користувачу відображається таблиця з усіма значеннями нормативів (рис. 4.18).

Вид змагань			
	МСУМК	МСУ	КМСУ
60 м	-	-	-
100 м	10,2 с	10,7 с	10,8 с
200 м	20,6 с	21,4 с	22,0 с
300 м	-	-	-
400 м	45,9 с	47,5 с	49,0 с
800 м	1:46,5 хв	1:49,5 хв	1:54 хв
1000 м	-	2:21 хв	2:28 хв
1500 м	3:38 хв	3:46 хв	3:54 хв
3000 м	7:52 хв	8:08 хв	8:30 хв
5000 м	13:26 хв	14:05 хв	14:40 хв

Рисунок 4.18 – Екран «Кваліфікаційна таблиця»

ВИСНОВКИ

Усім спортсменам важливо моніторити їх прогрес у тренуваннях. І легкоатлети не є виключенням. Так як значна частка спортсменів займаються без тренера, вони мають складнощі у відстеженні та аналізі власних результатів. Тому обрана тема кваліфікаційної роботи є актуальною.

Під час виконання дипломного проекту був розроблений мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики.

У ході виконання кваліфікаційної роботи було проведено аналіз предметної області та огляд продуктів-аналогів. Також було створено перелік вимог до майбутнього програмного продукту, а саме:

– робота секундоміра

– збереження, перегляд, редагування та видалення результатів забігів;

– виведення інформації про те, скільки залишилося до наступного розряду для кожного з основних змагань з кваліфікаційної таблиці;

– виведення програм тренувань для різних дистанцій;

– прогнозування результату з інших нормативів, базуючись на даних одного визначеного нормативу та виведення кваліфікаційної таблиці.

За вимогами стандарту IDEF0 створено контекстну діаграму та діаграму першого рівня декомпозиції моделі, яка описує структурні етапи процесу прогнозування можливого результату для бажаної дистанції та взаємозв’язки між ними. Можливі варіанти роботи з мобільним додатком описано у вигляді створеної UML діаграми варіантів використання.

Реалізацію представленого програмного продукту виконано мовою програмування Kotlin. Створений мобільний додаток повністю реалізує всі визначені вимоги. Проведено його успішне тестування, оскільки жодних багів не було виявлено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Чухланцева Н. Застосування інформаційних технологій у галузі фізичної культури і спорту // *Спортивна наука України.* 2016. № 3 (73). С. 21-25.

2. Молчанюк В. А. Основні напрямки використання сучасних комп'ютерних технологій у фізичній культурі і спорті // Сорок третя всеукраїнська практично-пізнавальна інтернет-конференція: збірник тез та доповідей.

3. Best Running Apps. Runkeeper [Електронний ресурс] – Режим доступу до ресурсу: https://www.verywellfit.com/best-running-apps-4165816#toc-best-overall-runkeeper

4. Best Running Apps. C25K 5K Trainer [Електронний ресурс] – Режим доступу до ресурсу: https://www.verywellfit.com/best-running-apps-4165816#toc-best-for-beginners-c25k-5k-trainer

5. Race Predictor [Електронний ресурс] – Режим доступу до ресурсу: https://www.omnicalculator.com/sports/race-predictor

6. StatCounter. Mobile Operating System Market Share Worldwide [Електронний ресурс] – Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide

7. Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/

8. Which is better, Kotlin vs. java? [Електронний ресурс] – Режим доступу до ресурсу: https://dac.digital/kotlin-vs-java/#kotlin-vs-java-which-is-better

9. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio

10. The Complete Guide to Understand IDEF Diagram [Електронний ресурс] – Режим доступу до ресурсу: https://www.edrawmax.com/article/the-complete-guide-to-understand-idef-diagram.html

11. Про затвердження Кваліфікаційних норм та вимог Єдиної спортивної класифікації України з олімпійських видів спорту: наказ Міністерства молоді та спорту України від 17.04.2014 № 1258. Додаток 22.

12. What is Use Case Diagram? [Електронний ресурс] – Режим доступу до ресурсу: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

13. Зарицька О.Л. Бази даних та інформаційні системи: Методичний посібник. – Житомир: Вид-во ЖДУ ім. І. Франка, 2009. – 132 с.

14. Get started with the Navigation component [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/guide/navigation/navigation-getting-started

15. Room Database with Kotlin [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/huawei-developers/room-database-with-kotlin-mvvm-architecture-477c3ad3c264

16. Coroutines guide [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/docs/coroutines-guide.html

17. Hal Higdon Training Programs [Електронний ресурс] – Режим доступу до ресурсу: https://www.halhigdon.com/training/

18. Sprint Training [Електронний ресурс] – Режим доступу до ресурсу: https://www.teachpe.com/sports-coaching/athletics/sprints

ДОДАТОК А

Планування робіт

Деталізація мети проекту методом SMART. Продуктом дипломного проекту є мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики.

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

Таблиця А.1 − Деталізація мети методом SMART

Specific (конкретна)	Створення мобільногододатку для аналізу результатів тренувань спортсменів з легкої атлетики, направленого на спрогнозування можливих досягнень із інших нормативів.
Measurable (вимірювана)	Результатом роботи проекту є спрогнозований результат для бажаної дистанції.
Achievable (досяжна)	Реалізація додатку здійснюється за допомогою середовища розробки Android Studio мовою програмування Kotlin і є чітко поставлені задачі, розробник достатньо кваліфікований для їх виконання.
Relevant (реалістична)	Використання створюваного мобільного додатку дозволить легкоатлетам економити час і зусилля, щоб дізнатися результат забігу на інші дистанції на основі поточних даних.
Time-framed (обмежена у часі)	Ціль має часове обмеження. Робота повинна бути виконана у терміни, що були оговорені замовником проекту. Проект повинен бути виконаний згідно з календарним планом.

Планування змісту структури робіт. Основним інструментом для планування змісту структури робіт служить WBS (Work Break Structure) – представлення проекту, виконане у вигляді ієрархічної структури робіт, що досягається за допомогою послідовної декомпозиції. Інструмент спрямований на детальне планування, оцінку вартості, визначення та розподіл персональної відповідальності виконавців та інші - тобто, на основні роботи і результати, що визначають зміст проекту.

Як правило, на верхньому рівні вказується сам проект, під ним (на першому рівні) – основні результати, кожен з яких, в свою чергу, деталізується, тобто наступний рівень завжди менше попереднього за обсягом робіт і, як правило, включає 2 і більше пакетів робіт. При цьому в різних гілках WBS може бути різна кількість рівнів в залежності від потрібного ступеня деталізації.

У ході виконання даного проекту перший рівень структури називається «Мобільний додаток для аналізу результатів тренувань спортсменів з легкої атлетики», який в свою чергу декомпозується на чотири рівні: ініціалізація, планування, реалізація та завершення.

Ініціалізація додатку складається з таких етапів: ознайомлення з предметною областю, визначення в потребі додатку та ідентифікація ідеї проекту.

Наступний рівень – планування, який розбивається на такі два рівні:

1. Аналіз документації.
2. Визначення вимог

– визначення інструментарію;
– планування WBS;
– планування OBS;
– складання календарного плану;
– визначення бюджету;
– визначення ризиків;

Третій рівень діаграми – реалізація, який розбивається на такі чотири рівні:

1. Моделювання роботи додатку.
2. Розробка макету додатку.
3. Розробка функціоналу додатку.
4. Тестування:

– тестування розробником;

– тестування незалежною особою.

Останній етап створення проекту є завершення, що містить у собі такі процеси: підготовка до завантаження, завантаження.

На рисунку А.1 приведена WBS-структура даного проекту.

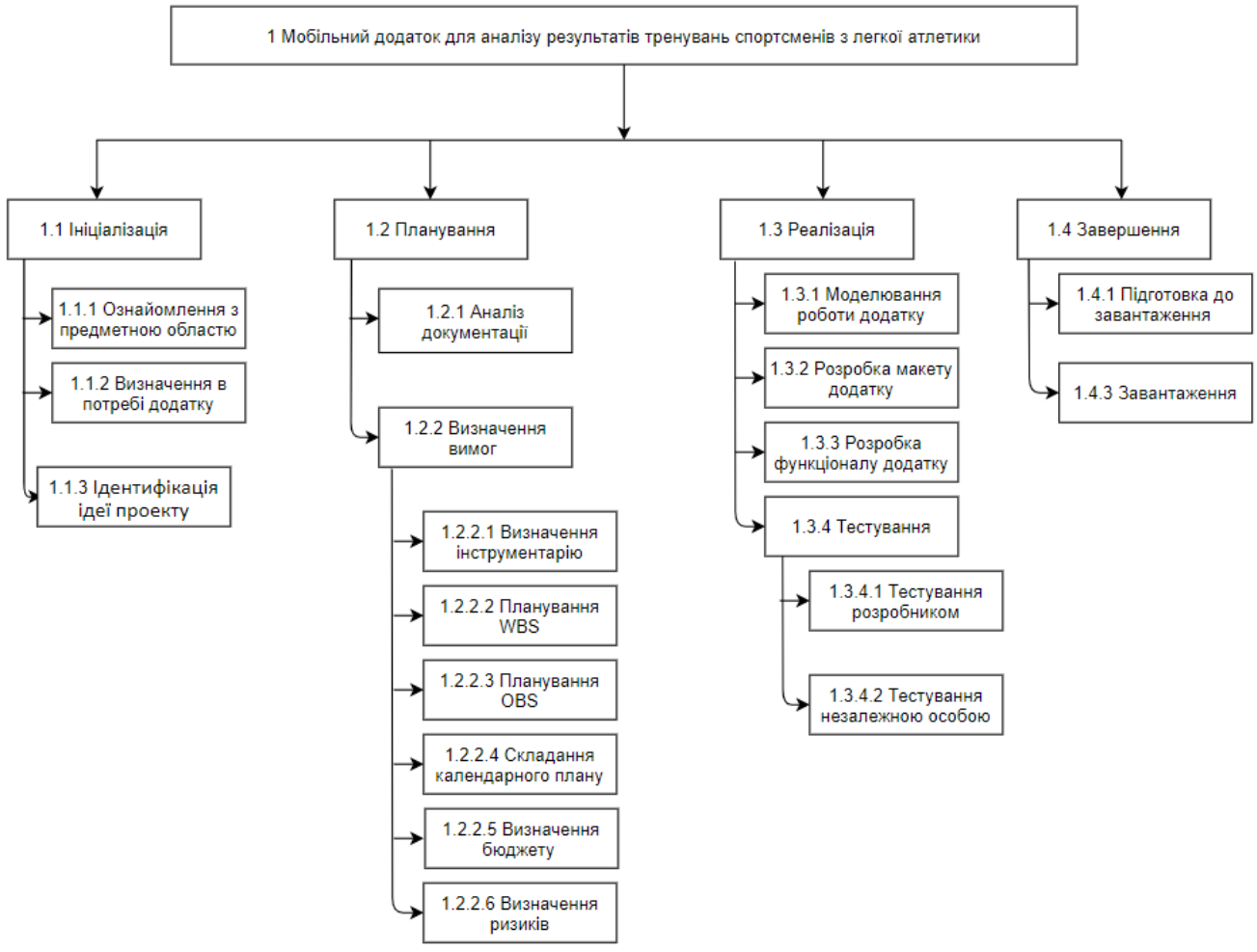


Рисунок А.1 – WBS-структура проекту

Планування структури організації, для впровадження готового проекту (OBS). Після того, як була побудована WBS структура проекту наступним етапом є розроблення OBS (Organization Break structure) - склад, підпорядкованість, взаємодія і розподіл робіт по підрозділах і органам управління, між якими встановлюються певні відносини з приводу реалізації владних повноважень, потоків команд і інформації. Організаційна структура проекту стосується тільки внутрішньої організаційної структури проекту і не стосується відносин проектних груп чи учасників з батьківськими організаціями. Список виконавців, що функціонують в проекті представлений в таблиці А.2. Організаційна структура проекту зображена на рисунку А.2.

Таблиця А.2 – Виконавці проекту

Роль	Ім’я	Проектна роль
Розробник	Малиновський Б.Ю.	Виконує розробку основного функціоналу проекту.
Менеджер проекту	Антипенко В.П.	Відповідає за виконання термінів, виконує збір та аналіз даних.
Тестувальник	Черняк О.С.	Відповідає за тестування функціоналу проекту

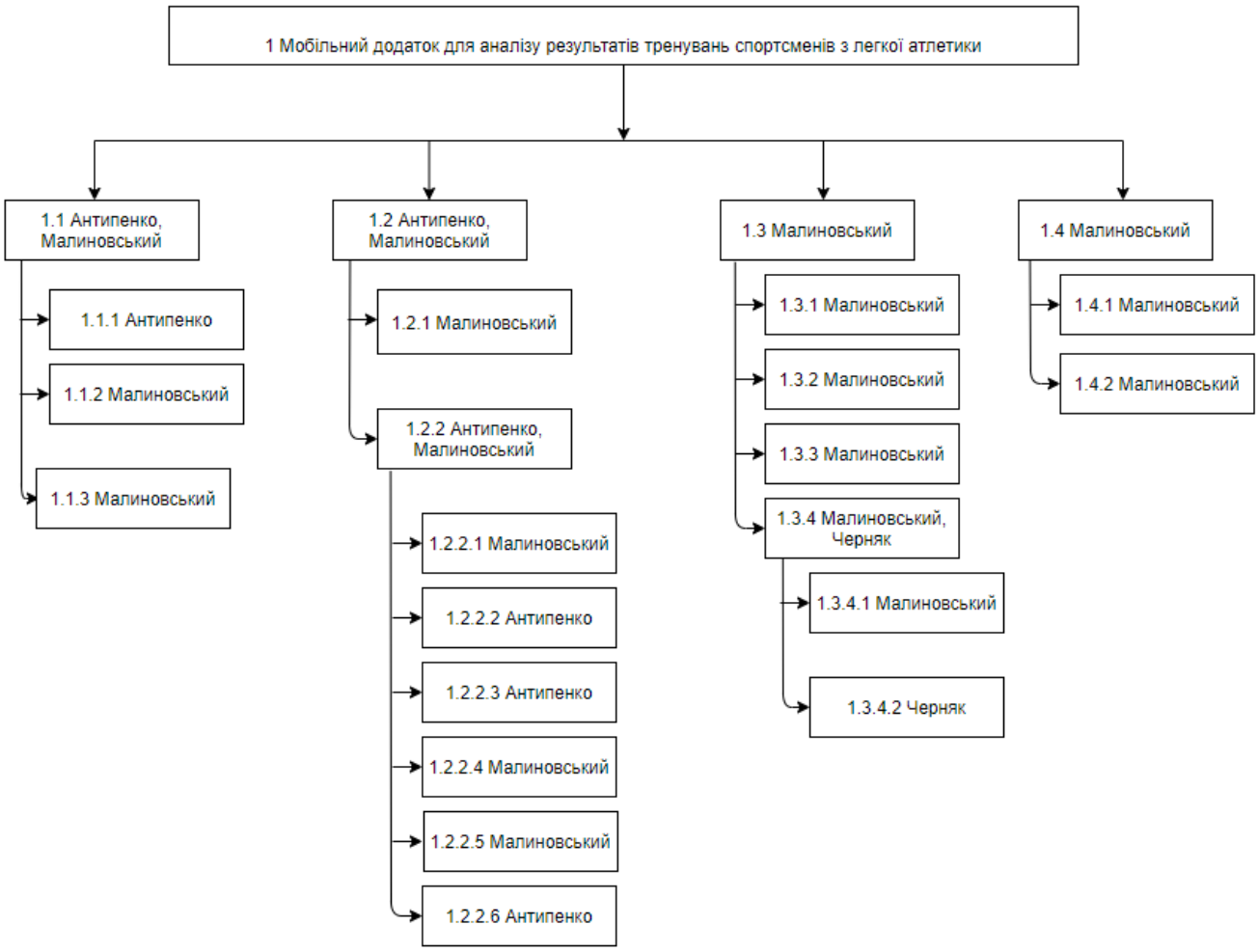
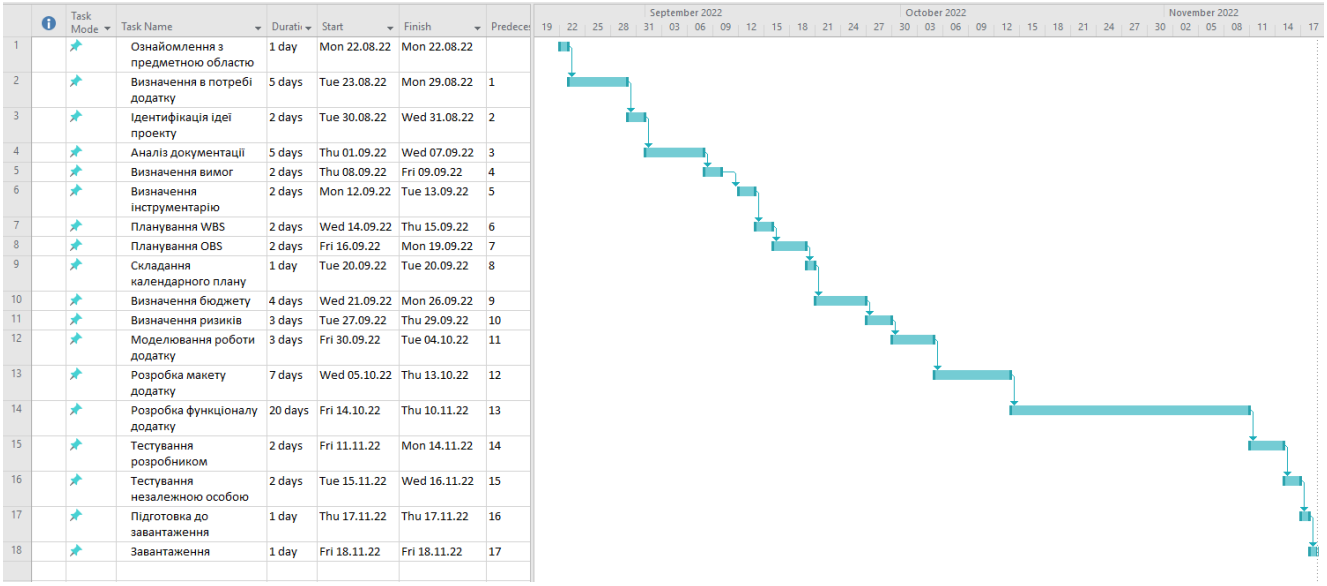


Рисунок А.2 – OBS-структура проекту

Діаграма Ганта. Далі побудуємо календарний план виконання дипломного проекту. Найпоширеніший формат графіка в будь-якій галузі — діаграма Ганта. Управління проектами з діаграмами Ганта засноване на форматі гістограм. Це допомагає відслідковувати відсоток робіт, виконаних по кожному завданню. Керівникам проектів дуже важливо правильно розподілити завдання і бути впевненими в тому, що проект буде завершений вчасно. Основна увага діаграм Ганта зосереджено на процентному завершенні кожного завдання. Крім того, діаграми Ганта краще для проектів з невеликою кількістю взаємопов'язаних завдань. Завдяки засобам програмного продукту

MS Project була розроблена діаграма Ганта, яка у вигляді гістограми відображає тривалість кожного процесу, що був визначений на етапі формування WBS. Діаграма Ганта представлена на рисунку А.3.



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Ознайомлення з предметною областю	1 day	Mon 22.08.22	Mon 22.08.22	
2		Визначення в потребі додатку	5 days	Tue 23.08.22	Mon 29.08.22	1
3		Ідентифікація ідеї проекту	2 days	Tue 30.08.22	Wed 31.08.22	2
4		Аналіз документації	5 days	Thu 01.09.22	Wed 07.09.22	3
5		Визначення вимог	2 days	Thu 08.09.22	Fri 09.09.22	4
6		Визначення інструментарію	2 days	Mon 12.09.22	Tue 13.09.22	5
7		Планування WBS	2 days	Wed 14.09.22	Thu 15.09.22	6
8		Планування OBS	2 days	Fri 16.09.22	Mon 19.09.22	7
9		Складання календарного плану	1 day	Tue 20.09.22	Tue 20.09.22	8
10		Визначення бюджету	4 days	Wed 21.09.22	Mon 26.09.22	9
11		Визначення ризиків	3 days	Tue 27.09.22	Thu 29.09.22	10
12		Моделювання роботи додатку	3 days	Fri 30.09.22	Tue 04.10.22	11
13		Розробка макету додатку	7 days	Wed 05.10.22	Thu 13.10.22	12
14		Розробка функціоналу додатку	20 days	Fri 14.10.22	Thu 10.11.22	13
15		Тестування розробником	2 days	Fri 11.11.22	Mon 14.11.22	14
16		Тестування незалежною особою	2 days	Tue 15.11.22	Wed 16.11.22	15
17		Підготовка до завантаження	1 day	Thu 17.11.22	Thu 17.11.22	16
18		Завантаження	1 day	Fri 18.11.22	Fri 18.11.22	17

Рисунок А.3 – Діаграма Ганта проекту

GANTT CHART

	Task Mode	Task Name	Durati	Start	Finish	Predece
1		Ознайомлення з предметною областю	1 day	Mon 22.08.22	Mon 22.08.22	
2		Визначення в потребі додатку	5 days	Tue 23.08.22	Mon 29.08.22	1
3		Ідентифікація ідеї проекту	2 days	Tue 30.08.22	Wed 31.08.22	2
4		Аналіз документації	5 days	Thu 01.09.22	Wed 07.09.22	3
5		Визначення вимог	2 days	Thu 08.09.22	Fri 09.09.22	4
6		Визначення інструментарію	2 days	Mon 12.09.22	Tue 13.09.22	5
7		Планування WBS	2 days	Wed 14.09.22	Thu 15.09.22	6
8		Планування OBS	2 days	Fri 16.09.22	Mon 19.09.22	7
9		Складання календарного плану	1 day	Tue 20.09.22	Tue 20.09.22	8
10		Визначення бюджету	4 days	Wed 21.09.22	Mon 26.09.22	9
11		Визначення ризиків	3 days	Tue 27.09.22	Thu 29.09.22	10
12		Моделювання роботи додатку	3 days	Fri 30.09.22	Tue 04.10.22	11
13		Розробка макету додатку	7 days	Wed 05.10.22	Thu 13.10.22	12
14		Розробка функціоналу додатку	20 days	Fri 14.10.22	Thu 10.11.22	13
15		Тестування розробником	2 days	Fri 11.11.22	Mon 14.11.22	14
16		Тестування незалежною особою	2 days	Tue 15.11.22	Wed 16.11.22	15
17		Підготовка до завантаження	1 day	Thu 17.11.22	Thu 17.11.22	16
18		Завантаження	1 day	Fri 18.11.22	Fri 18.11.22	17

Рисунок А.4 − Список робіт для побудови діаграми Ганта

Аналіз ризиків. Ризик – ймовірнісна подія, яка може позитивно чи негативно вплинути на проект. Причиною виникнення ризиків є невизначеності, існуючі в кожному проекті. Ризики можуть бути «відомі» – ті, які визначені, оцінені, для яких можливе планування. Ризики «невідомі» – ті, які не ідентифіковані і не можуть бути

прогнозовані. Хоча специфічні ризики і умови їх виникнення не визначені, але більшу частину ризиків можна передбачити.

Ідентифікація ризиків – визначення ризиків, здатних вплинути на проект, і документування їх характеристик.

Ідентифікація ризиків визначає, які ризики здатні вплинути на проект, і документує характеристики цих ризиків. Ідентифікація ризиків не буде ефективною, якщо вона не буде проводитися регулярно протягом реалізації проекту.

Ідентифікація ризиків повинна залучати якомога більше учасників: менеджерів проекту, користувачів, незалежних фахівців.

Класифікація ризиків є наступна:

1. За ймовірністю виникнення:

– слабо ймовірнісні;
– мало ймовірнісні;
– імовірні;
– досить імовірні;
– майже імовірні.

2. За величиною втрат:

– мінімальна;
– низька;
– середня;
– висока;
– максимальна.

На основі цих даних була проведена класифікація ризиків для даного проекту, що наведена в таблиці А.3.

Таблиця А.3 – Класифікація ризиків

№	Назва ризику	Ймовірність	Величина втрат
1	Недотримання календарного плану	1	3
2	Некоректна робота програмного забезпечення	4	4
3	Некоректна робота апаратного забезпечення	4	4
4	Хвороба розробника	2	2
5	Некоректне тестування	2	1

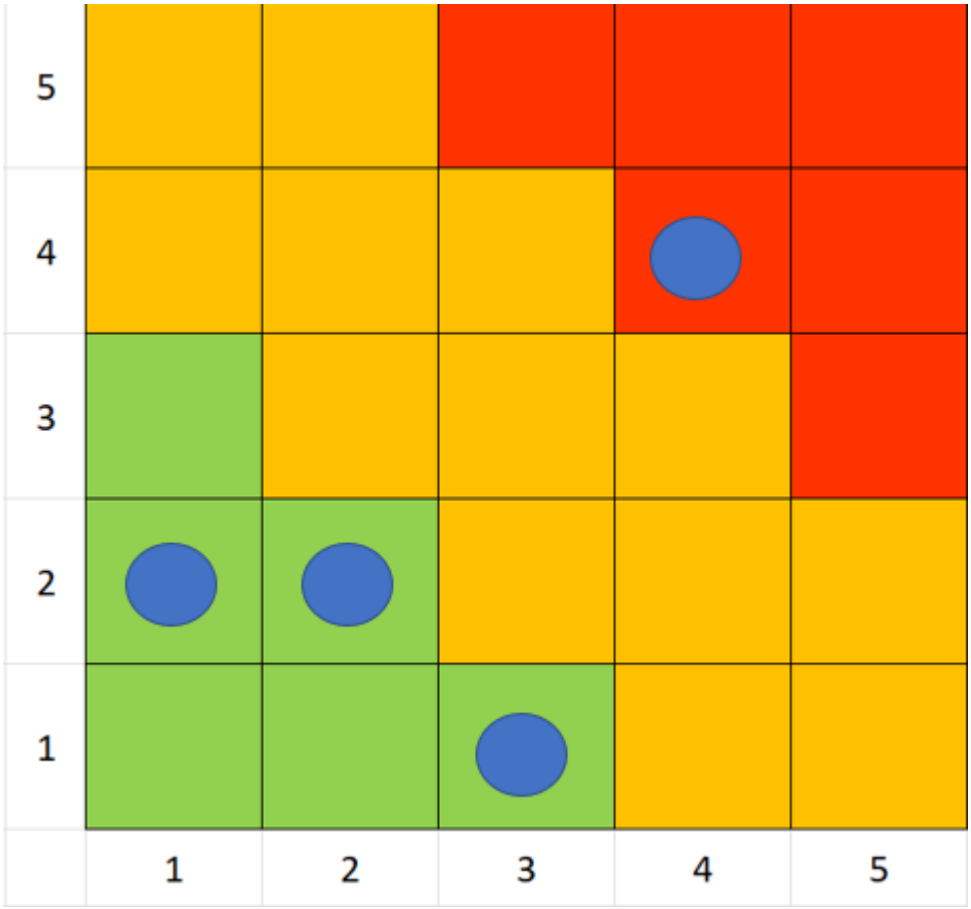


Рисунок А.5 – Матриця ризиків

Далі визначаємо рівні ризиків та ступінь їх дії.

Рівні можуть бути такі:

– допустимі 1<R<4;
– оправдані 5<R<10;
– недопустимі 13<R<25.

Ступінь дії ризиків є наступним:

– ті, що можна проігнорувати 1<R<4;
– незначні 5<R<8;
– помірні 9<R<10;
– істотні 11<R<16;
– критичні 17<R<25.

На основі цих даних була виконана оцінка ступенів та рівнів для кожного ризику в проекті. Результати роботи представлені в таблиці А.4.

Таблиця А.4 – Визначення ступенів та рівнів ризиків

№	Назва ризику	Ймовірність ризику	Величина втрат	Рівень ризику	Ступінь дії
1	Недотримання календарного плану	1	3	Допустимий	Проігнорувати
2	Некоректна робота програмного забезпечення	4	4	Недопустимий	Істотний

Продовження табл. А.4

№	Назва ризику	Ймовірність ризику	Величина втрат	Рівень ризику	Ступінь дії
3	Некоректна робота апаратного забезпечення	4	4	Недопустимий	Істотний
4	Хвороба розробника	2	2	Допустимий	Проігнорувати
5	Некоректне тестування	2	1	Допустимий	Проігнорувати

Після виконання прогнозування виникнення ризиків та їх ступеню впливу на результат реалізації проекту, були розроблені варіанти запобігання та реакції на кожний із них. Результати даного етапу представлені в таблиці А.5.

Таблиця А.5 – Варіанти запобігання та реакції на виявлені ризики

Ризики проекту	План запобігання ризику	План реакції на ризик
Недотримання календарного плану	Створення плану реалізації проекту на основі ретельного аналізу списку всіх робіт. Затвердження зазначених термінів із замовником. Командна робота над планом термінів виконання. (Можливість внесення правок перед затвердженням усіма членами команди).	1. Обговорення варіантів внесення правок до термінів реалізації із керівником та замовником. 2. Домовитися про умови зміни термінів із замовником. Якщо це недопустимо, тоді переорганізувати роботу таким чином, щоб в результаті терміни виконувалися.
Некоректна робота програмного забезпечення	1. Встановлення ліцензійного програмного забезпечення з перевірених джерел перед початком роботи. 2. Забезпечити наявність антивірусного програмного забезпечення.	Перезапуск або переустановлення програми, яка дала збій.

Продовження табл. А.5

Некоректна робота апаратного забезпечення	Раз на 4-6 місяців виконувати перевірку працездатності апаратного забезпечення	Виконати ремонт апаратного забезпечення, якщо терміни виконання завдань не дозволяють чекати, знайти тимчасову заміну.
Хвороба розробника	Виконувати певну частину роботи в команді для того, щоб члени проекту змогли замінити один одного при необхідності. При плануванні термінів залишити декілька резервних днів для таких випадків.	Передати повноваження робітника іншому члену команди, якщо цього вимагають терміни виконання.
Некоректне тестування	Виконати пошук кваліфікованого тестувальника в даній предметній області.	Передати проект на додаткове тестування кваліфікованому спеціалісту.

ДОДАТОК Б

Лістинг основних модулів

Файл HomeFragment.kt

```
package com.example.runhelper.ui.screen.home


import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.example.runhelper.MainActivity
import com.example.runhelper.databinding.FragmentHomeBinding
import com.example.runhelper.enum.Screen
import com.example.runhelper.preferences.RunHelperPreferences
import com.example.runhelper.ui.dialog.GenderDeterminationDialog
import com.example.runhelper.ui.screen.NewRaceFragment


class HomeFragment : Fragment() {


    private var _binding: FragmentHomeBinding? = null


    private val binding get() = _binding!!
    private val stopwatchFragment = StopwatchFragment()


    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
```

```
    _binding = FragmentHomeBinding.inflate(inflater, container, false)
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    showGenderDeterminationDialogIfNeeded()
    setupViews()
}

private fun setupViews() {
    setupStopwatchFragment()
    binding.saveRaceHomeButton.setOnClickListener {
        if (stopwatchFragment.controller.time > 0.0) {
            val bundle = Bundle()
            bundle.putSerializable(
                NewRaceFragment.STOPWATCH_TIME,
                stopwatchFragment.controller.time
            )
            (activity as MainActivity).navigateToScreen(Screen.NEW_RACE, bundle)
            stopwatchFragment.resetTimer()
        } else {
            (activity as MainActivity).navigateToScreen(Screen.NEW_RACE)
        }
    }
}

private fun showGenderDeterminationDialogIfNeeded() {
    val gender = RunHelperPreferences.getInstance(requireContext()).getGender()
    if (gender == null) {
        GenderDeterminationDialog(requireContext()).show()
    }
```

```
    }

    private fun setupStopwatchFragment() {
        val transaction = childFragmentManager
            .beginTransaction()
            .replace(binding.stopwatchFragmentContainer.id, stopwatchFragment)
        transaction.commit()
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Файл StopwatchFragment.kt

```
package com.example.runhelper.ui.screen.home

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.appcompat.content.res.AppCompatResources
import androidx.fragment.app.Fragment
import com.example.runhelper.R
import com.example.runhelper.controller.StopwatchController
import com.example.runhelper.databinding.FragmentStopwatchBinding

class StopwatchFragment : Fragment() {
    private var _binding: FragmentStopwatchBinding? = null
    private val binding get() = _binding!!
```

```
lateinit var controller: StopwatchController

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    controller = StopwatchController(requireActivity(), onUpdateTime = this::setTimeTV)
    controller.setService()
    controller.registerReceiver()
}

override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    _binding = FragmentStopwatchBinding.inflate(inflater, container, false)
    return binding.root
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    setTimeTV()
    setStartStopButton()
    setResetButton()
}

private fun setStartStopButton() {
    binding.startStopButton.apply {
        val startStopButtonStatusPair = controller.getStartStopButtonStatusPair()
        text = context.getString(startStopButtonStatusPair.first)
        icon =
            AppCompatResources.getDrawable(
                requireContext(),
```

```
                startStopButtonStatusPair.second
            )
        setOnClickListener {
            controller.startStopTimer(
                this@StopwatchFragment::onStartTimer,
                this@StopwatchFragment::onStopTimer
            )
        }
    }
}

private fun setResetButton() {
    binding.resetButton.setOnClickListener {
        resetTimer()
    }
}

fun resetTimer() {
    controller.resetTimer(
        onResetTimer = this::setTimeTV,
        this::onStopTimer,
    )
}

private fun onStartTimer() {
    binding.startStopButton.text = context?.getString(R.string.stop)
    binding.startStopButton.icon =
        AppCompatResources.getDrawable(requireContext(), R.drawable.ic_pause)
}

private fun onStopTimer() {
```

```
        binding.startStopButton.text = context?.getString(R.string.start)
        binding.startStopButton.icon =
            AppCompatResources.getDrawable(requireContext(), R.drawable.ic_play)
    }

    private fun setTimeTV() {
        binding.timeTV.text = controller.getTimeStringFromDouble()
    }
}
```

Файл StopwatchController.kt

```
package com.example.runhelper.controller

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import androidx.fragment.app.FragmentActivity
import com.example.runhelper.R
import com.example.runhelper.service.TimerService
import kotlin.math.roundToInt

class StopwatchController(
    private val activity: FragmentActivity,
    private val onUpdateTime: () -> Unit,
) {

    private var timerStarted = false
    private lateinit var serviceIntent: Intent
    var time = 0.0
```

```
fun setService() {
    serviceIntent = Intent(activity.applicationContext, TimerService::class.java)
}

fun registerReceiver() {
    activity.registerReceiver(updateTime, IntentFilter(TimerService.TIMER_UPDATED))
}

fun startStopTimer(onStartTimer: () -> Unit, onStopTimer: () -> Unit) {
    if (timerStarted)
        stopTimer(onStopTimer)
    else
        startTimer(onStartTimer)
}

private fun startTimer(onStartTimer: () -> Unit) {
    serviceIntent.putExtra(TimerService.TIME_EXTRA, time)
    activity.startService(serviceIntent)
    onStartTimer.invoke()
    timerStarted = true
}

private fun stopTimer(onStopTimer: () -> Unit) {
    activity.stopService(serviceIntent)
    onStopTimer.invoke()
    timerStarted = false
}

fun resetTimer(onResetTimer: () -> Unit, onStopTimer: () -> Unit) {
    stopTimer(onStopTimer)
    time = 0.0
```

```
        onResetTimer.invoke()
    }

    fun getStartStopButtonStatusPair(): Pair<Int, Int> {
        return if (timerStarted) {
            Pair(R.string.stop, R.drawable.ic_pause)
        } else {
            Pair(R.string.start, R.drawable.ic_play)
        }
    }

    fun getTimeStringFromDouble(): String {
        val resultInt = time.roundToInt()
        val hours = resultInt % 86400 / 3600
        val minutes = resultInt % 86400 % 3600 / 60
        val seconds = resultInt % 86400 % 3600 % 60

        return makeTimeString(hours, minutes, seconds)
    }

    private fun makeTimeString(hour: Int, min: Int, sec: Int): String =
        String.format("%02d:%02d:%02d", hour, min, sec)

    private val updateTime: BroadcastReceiver = object : BroadcastReceiver() {
        override fun onReceive(context: Context, intent: Intent) {
            time = intent.getDoubleExtra(TimerService.TIME_EXTRA, 0.0)
            onUpdateTime.invoke()
        }
    }
}
```

Файл NewRaceFragment.kt

```
package com.example.runhelper.ui.screen


import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.fragment.app.Fragment
import com.example.runhelper.MainActivity
import com.example.runhelper.R
import com.example.runhelper.controller.NewRaceController
import com.example.runhelper.databinding.FragmentNewRaceBinding
import com.example.runhelper.db.model.Race
import com.example.runhelper.util.DateFormats
import com.example.runhelper.util.DurationFormat
import kotlinx.android.synthetic.main.fragment_my_results.*
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.time.Duration
import java.util.*


class NewRaceFragment : Fragment() {

    companion object {

        const val IS_EDIT_MODE = "IS_EDIT_MODE"

        const val RACE_FOR_EDIT = "RACE_FOR_EDIT"

        const val STOPWATCH_TIME = "STOPWATCH_TIME"

    }
```

```
private var _binding: FragmentNewRaceBinding? = null

private val binding get() = _binding!!


private lateinit var controller: NewRaceController


override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    _binding = FragmentNewRaceBinding.inflate(inflater, container, false)
    return binding.root
}


override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    controller = NewRaceController(view.context)
    controller.getFragmentArguments(arguments)
    setupEditTexts()
    setupSaveEditButton()
}


private fun setupEditTexts() {
    binding.apply {
        resultTimeEt.setOnClickListener {
            controller.showDurationPicker(this@NewRaceFragment::saveResultDuration)
        }
        dateTimeEt.apply {
            val dateFormat = SimpleDateFormat(DateFormats.DMY_HM)
            val now = Calendar.getInstance().timeInMillis
            val nowLocal = Duration.ofMillis(now).plusHours(2)
            setText(dateFormat.format(nowLocal.toMillis()))
```

```
                setOnClickListener {
                    controller.showDatePickerFragment(
                        activity!!,
                        this@NewRaceFragment::saveDateTime
                    )
                }
            }
            if (controller.stopwatchTime != null) {
                val resultTime = controller.stopwatchTime!!
                controller.resultDuration = resultTime
                resultTimeEt.setText(DurationFormat.getDurationString(resultTime))
            }
            if (controller.isEditMode && controller.raceForEdit != null) {
                distanceEt.setText(controller.raceForEdit!!.distance.toString())
                val resultTime = controller.raceForEdit!!.resultTime
                controller.resultDuration = Duration.ofMillis(resultTime)
resultTimeEt.setText(DurationFormat.getDurationString(Duration.ofMillis(resultTime)))
dateTimeEt.setText(SimpleDateFormat(DateFormats.DMY_HM).format(controller.raceForEdit!!.dateTime)
)
                noteEt.setText(controller.raceForEdit!!.note)
            }
        }
    }

    private fun setupSaveEditButton() {
        binding.saveEditRaceButton.apply {
            text =
                activity?.getString(if (controller.isEditMode) R.string.edit_race else
R.string.save_race)
            setOnClickListener(this@NewRaceFragment::onSaveEditButtonClick)
```

```
        }
    }

    private fun onSaveEditButtonClick(v: View) {
        binding.apply {
            if (canRaceBeSaved()) {
                GlobalScope.launch {
                    val distance = distanceEt.editableText.toString().toInt()
                    val resultTime = controller.resultDuration!!.toMillis()
                    val dateTime =

SimpleDateFormat(DateFormats.DMY_HM).parse(dateTimeEt.editableText.toString())
                    val note = noteEt.editableText.toString()

                    if (controller.isEditMode && controller.raceForEdit != null) {
                        val race = controller.raceForEdit!!.copy(
                            distance = distance,
                            resultTime = resultTime,
                            dateTime = dateTime.time,
                            note = note
                        )
                        controller.updateRace(race)
                    } else {
                        val race = Race(distance, resultTime, dateTime.time, note)
                        controller.insertRace(race)
                    }
                    activity?.runOnUiThread {
                        (activity as MainActivity).goBack()
                    }
                }
            } else {
```

```
            Toast.makeText(
                context,
                requireContext().getString(R.string.all_fields_except_note_should_be_filled),
                Toast.LENGTH_LONG
            ).show()
        }
    }
}

private fun canRaceBeSaved(): Boolean {
    binding.apply {
        return distanceEt.editableText.isNotBlank()
                && resultTimeEt.editableText.isNotBlank()
                && dateTimeEt.editableText.isNotBlank()
    }
}

private fun saveDateTime(calendar: Calendar) {
    val dateFormat = SimpleDateFormat(DateFormats.DMY_HM)
    binding.dateTimeEt.setText(dateFormat.format(calendar.time))
}

private fun saveResultDuration(duration: Duration) {
    val resultTime =
        DurationFormat.getDurationString(duration)
    binding.resultTimeEt.setText(resultTime)
}

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
```

```
    }
}
```

Файл NewRaceController.kt

```
package com.example.runhelper.controller

import android.content.Context
import android.os.Bundle
import androidx.fragment.app.FragmentActivity
import com.example.runhelper.db.RunHelperDB
import com.example.runhelper.db.model.Race
import com.example.runhelper.ui.pickers.DatePickerFragment
import com.example.runhelper.ui.pickers.DurationPicker
import com.example.runhelper.ui.pickers.TimePickerFragment
import com.example.runhelper.ui.screen.NewRaceFragment
import java.time.Duration
import java.util.*

class NewRaceController(private val context: Context) {

    var isEditMode = false
    var raceForEdit: Race? = null
    var resultDuration: Duration? = null
    var stopwatchTime: Duration? = null

    fun getFragmentArguments(arguments: Bundle?) {
        if (arguments != null) {
            if (arguments.containsKey(NewRaceFragment.IS_EDIT_MODE) &&
                arguments.containsKey(NewRaceFragment.RACE_FOR_EDIT)
            ) {
                isEditMode = arguments.getSerializable(NewRaceFragment.IS_EDIT_MODE) as Boolean
```

```
            raceForEdit = arguments.getSerializable(NewRaceFragment.RACE_FOR_EDIT) as Race
        }
        if (arguments.containsKey(NewRaceFragment.STOPWATCH_TIME)) {
            val stopwatchTimeSeconds =
                arguments.getSerializable(NewRaceFragment.STOPWATCH_TIME) as Double
            stopwatchTime = Duration.ofSeconds(stopwatchTimeSeconds.toLong())
        }
    }
}

fun showDurationPicker(saveResultDuration: (duration: Duration) -> Unit) {
    DurationPicker(context) { hours, minutes, seconds, millis ->
        val duration = Duration.ofHours(hours.toLong())
            .plusMinutes(minutes.toLong())
            .plusSeconds(seconds.toLong())
            .plusMillis(millis.toLong())
        saveResultDuration(duration)
        resultDuration = duration
    }.show()
}

fun showDatePickerFragment(
    activity: FragmentActivity,
    saveDateTime: (calendar: Calendar) -> Unit
) {
    DatePickerFragment { year: Int, month: Int, day: Int ->
        onDateSetListener(activity, year, month, day, saveDateTime)
    }.show(
        activity.supportFragmentManager,
        "datePicker"
    )
```

```
    }

    private fun onDateSetListener(
        activity: FragmentActivity,
        year: Int,
        month: Int,
        day: Int,
        saveDateTime: (calendar: Calendar) -> Unit
    ) {
        TimePickerFragment { hourOfDay: Int, minute: Int ->
            val calendar = Calendar.getInstance()
            calendar.set(year, month, day, hourOfDay, minute)
            saveDateTime(calendar)
        }.show(activity.supportFragmentManager, "timePicker")
    }

    fun updateRace(race: Race) {
        RunHelperDB.getInstance(context)
            .raceDao()
            .update(race)
    }

    fun insertRace(race: Race) {
        RunHelperDB.getInstance(context)
            .raceDao()
            .insert(race)
    }
}
```

Файл MyResultsFragment.kt

```
package com.example.runhelper.ui.screen
```

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.example.runhelper.MainActivity
import com.example.runhelper.R
import com.example.runhelper.controller.MyResultsController
import com.example.runhelper.controller.WhenNextQualificationCategoryController
import com.example.runhelper.databinding.FragmentMyResultsBinding
import com.example.runhelper.db.model.Race
import com.example.runhelper.enum.Screen
import com.example.runhelper.ui.view.RaceView
import com.example.runhelper.util.DurationFormat
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch


class MyResultsFragment : Fragment() {

    private var _binding: FragmentMyResultsBinding? = null
    private val binding get() = _binding!!

    private lateinit var myResultsController: MyResultsController
    private lateinit var nextCategoryController: WhenNextQualificationCategoryController

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
```

```
    ): View {
        _binding = FragmentMyResultsBinding.inflate(inflater, container, false)
        myResultsController = MyResultsController(binding.root.context)
        nextCategoryController = WhenNextQualificationCategoryController(binding.root.context)
        myResultsController.setRaceListObserver(this::refreshRaceList)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        setupButtons()
    }

    private fun setupButtons() {
        binding.apply {
            addButton.setOnClickListener {
                (activity as MainActivity).navigateToScreen(Screen.NEW_RACE)
            }
            viewForAllWhenNewCategoryBtn.setOnClickListener {
                (activity as
MainActivity).navigateToScreen(Screen.WHEN_NEXT_QUALIFICATION_CATEGORY)
            }
        }
    }

    private fun setupNearestNextCategoryView() {
        GlobalScope.launch {
            val nearestNextCategory = nextCategoryController.getNearestNextCategory()
            val competition = nearestNextCategory.first
            val category = nearestNextCategory.second
            val difference = nearestNextCategory.third
            activity?.runOnUiThread {
```

```
            if (competition != null) {
                val differenceString = DurationFormat.getDurationString(difference!!)
                binding.nearestNextQualificationCategoryTv.text = requireContext().getString(
                    R.string.nearest_next_qualification_category_placeholder,
                    category,
                    "${competition.distance} м",
                    differenceString
                )
            } else {
                binding.nextQualificationCategoryLayout.visibility = View.GONE
            }
        }
    }
}

private fun refreshRaceList(raceList: List<Race>) {
    composeRaceList(raceList)
    setupNearestNextCategoryView()
}

private fun composeRaceList(raceList: List<Race>) {
    GlobalScope.launch {
        val sortedRaceList = raceList.sortedByDescending { it.dateTime }

        activity?.runOnUiThread {
            binding.myResultsListLayout.removeAllViews()
            for (race in sortedRaceList) {
                val raceView = RaceView(
                    requireContext(),
                    race
                ) {
```

```
                        myResultsController.onDeleteRaceTap(race)
                    }
                    binding.myResultsListLayout.addView(raceView)
                }
            }
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Файл MyResultsController.kt

```
package com.example.runhelper.controller

import android.annotation.SuppressLint
import android.content.Context
import com.example.runhelper.db.RunHelperDB
import com.example.runhelper.db.model.Race
import com.example.runhelper.ui.dialog.DeleteConfirmationDialog
import io.reactivex.android.schedulers.AndroidSchedulers
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch

class MyResultsController(private val context: Context) {

    fun onDeleteRaceTap(race: Race) {
        val dialog = DeleteConfirmationDialog(context) {
            GlobalScope.launch {
```

```
                RunHelperDB.getInstance(context)
                    .raceDao()
                    .delete(race.id)
            }
        }
        dialog.show()
    }

    @SuppressLint("CheckResult")
    fun setRaceListObserver(composeRaceList: (raceList: List<Race>) -> Unit) {
        RunHelperDB.getInstance(context).raceDao().getAllFlowable()
            .observeOn(AndroidSchedulers.mainThread())
            ?.subscribe(composeRaceList)
    }
}
```

Файл WhenNextQualificationCategoryFragment.kt

```
package com.example.runhelper.ui.screen

import android.os.Bundle
import android.view.ContextThemeWrapper
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TableLayout
import android.widget.TableRow
import android.widget.TextView
import androidx.fragment.app.Fragment
import com.example.runhelper.R
import com.example.runhelper.controller.WhenNextQualificationCategoryController
import com.example.runhelper.databinding.FragmentWhenNextQualificationCategoryBinding
```

```
import com.example.runhelper.db.model.QualificationCompetition
import com.example.runhelper.util.DurationFormat
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import java.time.Duration


class WhenNextQualificationCategoryFragment : Fragment() {

    private var _binding: FragmentWhenNextQualificationCategoryBinding? = null
    private val binding get() = _binding!!

    private lateinit var controller: WhenNextQualificationCategoryController

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentWhenNextQualificationCategoryBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        controller = WhenNextQualificationCategoryController(view.context)
        composeTable()
    }

    private fun composeTable() {
        GlobalScope.launch {
```

```
        for (competition in controller.competitionList) {
            val bestResult = controller.getBestPersonalResult(competition)
            val nextCategory =
                controller.getNextCategory(competition, bestResult)

            activity?.runOnUiThread {
                val row = composeRow(competition, nextCategory, bestResult)
                binding.whenNextQualificationCategoryTable.addView(
                    row,
                    TableLayout.LayoutParams(
                        TableLayout.LayoutParams.MATCH_PARENT,
                        TableLayout.LayoutParams.WRAP_CONTENT
                    )
                )
            }
        }
    }
}

private fun composeRow(
    competition: QualificationCompetition,
    nextCategory: Triple<String, Duration?, Int?>,
    bestResult: Duration?
): TableRow {
    val row =
        TableRow(ContextThemeWrapper(activity, R.style.TableRow))
    row.layoutParams = TableRow.LayoutParams(0, TableRow.LayoutParams.WRAP_CONTENT)

    val tvParams = TableRow.LayoutParams(0, 100)

    val distanceTv =
```

```
    TextView(ContextThemeWrapper(activity, R.style.TableText))
distanceTv.apply {
    text = "${competition.distance} м"
    layoutParams = tvParams
    row.addView(this)
}

val bestResultTv =
    TextView(ContextThemeWrapper(activity, R.style.TableText))
bestResultTv.apply {
    text = getDurationFormatString(bestResult)
    layoutParams = tvParams
    row.addView(this)
}

val nextCategoryTv =
    TextView(ContextThemeWrapper(activity, R.style.TableText))
nextCategoryTv.apply {
    text = nextCategory.first
    layoutParams = tvParams
    row.addView(this)
}

val nextCategoryConditionTv =
    TextView(ContextThemeWrapper(activity, R.style.TableText))
nextCategoryConditionTv.apply {
    text = getDurationFormatString(nextCategory.second)
    layoutParams = tvParams
    row.addView(this)
}
```

```
        return row
    }

    private fun getDurationFormatString(duration: Duration?): String {
        return if (duration != null) {
            DurationFormat.getDurationString(duration)
        } else {
            "-"
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Файл WhenNextQualificationCategoryController.kt

```
package com.example.runhelper.controller

import android.content.Context
import com.example.runhelper.db.RunHelperDB
import com.example.runhelper.db.model.QualificationCompetition
import com.example.runhelper.preferences.RunHelperPreferences
import com.example.runhelper.db.repo.QualificationCompetitionRepository
import java.time.Duration

class WhenNextQualificationCategoryController(private val context: Context) {

    val competitionList = QualificationCompetitionRepository.getCompetitionList(
        RunHelperPreferences.getInstance(
```

```
        context
    ).getGender()!!
)

fun getBestPersonalResult(competition: QualificationCompetition): Duration? {
    val raceList = RunHelperDB.getInstance(context)
        .raceDao()
        .getAllDistanceResults(competition.distance)
    return if (raceList.isNotEmpty()) {
        val bestTime =
            raceList.minWithOrNull(Comparator.comparingLong { it.resultTime })!!.resultTime
        return Duration.ofMillis(bestTime)
    } else {
        null
    }
}

fun getNextCategory(
    competition: QualificationCompetition,
    bestResult: Duration?
): Triple<String, Duration?, Int?> {
    if (bestResult != null) {
        for (categoryCondition in competition.categoriesConditionsListAsc) {
            if (getNextCategoryCondition(categoryCondition, bestResult)) {
                return categoryCondition
            }
        }
        return nullCategory()
    } else {
        return nullCategory()
    }
```

```
    }

    private fun getNextCategoryCondition(
        category: Triple<String, Duration?, Int?>,
        bestResult: Duration
    ): Boolean {
        return category.second != null && bestResult.toMillis() > category.second!!.toMillis()
    }

    fun getNearestNextCategory(): Triple<QualificationCompetition?, String?, Duration?> { //
competition, category, difference
        var nearestNextCategoryCompetition: QualificationCompetition? = null
        var nearestNextCategoryCompetitionCategory: String? = null
        var nearestNextCategoryCompetitionDifference: Duration? = null
        var leastDifferenceConditionTimeDivision: Double? = null

        for (competition in competitionList) {
            val bestResult = getBestPersonalResult(competition)
            val nextCategory = getNextCategory(competition, bestResult)
            val bestTimeConditionTimeDifference =
                if (nextCategory.second != null) bestResult?.minus(nextCategory.second)
                else null
            val differenceConditionTimeDivision =
                if (bestTimeConditionTimeDifference != null) {
                    bestTimeConditionTimeDifference.toMillis()
                        .toDouble() / nextCategory.second?.toMillis()!!.toDouble()
                } else {
                    null
                }
            if (differenceConditionTimeDivision != null && (leastDifferenceConditionTimeDivision
== null || differenceConditionTimeDivision < leastDifferenceConditionTimeDivision)) {
```

```
                leastDifferenceConditionTimeDivision = differenceConditionTimeDivision
                nearestNextCategoryCompetition = competition
                nearestNextCategoryCompetitionCategory = nextCategory.first
                nearestNextCategoryCompetitionDifference = bestTimeConditionTimeDifference
            }
        }
        return Triple(
            nearestNextCategoryCompetition,
            nearestNextCategoryCompetitionCategory,
            nearestNextCategoryCompetitionDifference
        )
    }

    private fun nullCategory(): Triple<String, Duration?, Int?> {
        return Triple("-", null, 0)
    }
}
```

Файл ResultPredictionFragment.kt

```
package com.example.runhelper.ui.screen

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.example.runhelper.controller.ResultPredictionController
import com.example.runhelper.databinding.FragmentResultPredictionBinding
import com.example.runhelper.ui.dialog.ResultPredictionDialog
import com.example.runhelper.ui.view.DistanceView
import kotlinx.coroutines.GlobalScope
```

```
import kotlinx.coroutines.launch



class ResultPredictionFragment : Fragment() {

    private var _binding: FragmentResultPredictionBinding? = null
    private val binding get() = _binding!!

    private lateinit var controller: ResultPredictionController

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentResultPredictionBinding.inflate(inflater, container, false)
        controller = ResultPredictionController(binding.root.context)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        for (competition in controller.competitionList) {
            val predictionDistanceView =
                DistanceView(
                    requireContext(),
                    competition.distance,
                    clickHandler = this::onPredictionDistanceViewClick
                )
            binding.resultPredictionDistancesLayout.addView(predictionDistanceView)
        }
    }
```

```
    private fun onPredictionDistanceViewClick(distance: Int) {
        GlobalScope.launch {
            val predictedTime = controller.getPredictedDistanceResult(distance)
            activity?.runOnUiThread {
                val dialog = ResultPredictionDialog(requireContext(), distance, predictedTime)
                dialog.show()
            }
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Файл ResultPredictionController.kt

```
package com.example.runhelper.controller

import android.content.Context
import com.example.runhelper.db.RunHelperDB
import com.example.runhelper.db.model.QualificationCompetition
import com.example.runhelper.db.model.Race
import com.example.runhelper.preferences.RunHelperPreferences
import com.example.runhelper.db.repo.QualificationCompetitionRepository
import java.time.Duration
import java.util.function.Predicate
import kotlin.math.pow

class ResultPredictionController(private val context: Context) {
```

```
    val competitionList =
        QualificationCompetitionRepository.getCompetitionList(
            RunHelperPreferences.getInstance(
                context
            ).getGender()!!
        )

    fun getPredictedDistanceResult(predictedDistance: Int): Long? {
        val twoLastWeeksRaces = getTwoLastWeeksRaces()
        val bestRaceForPrediction = getBestRace(twoLastWeeksRaces)
        return if (bestRaceForPrediction != null) {
            val nearestDistance = bestRaceForPrediction.distance
            val nearestTime = bestRaceForPrediction.resultTime
            val predictedTime =
                (nearestTime * (predictedDistance.toDouble() /
nearestDistance.toDouble()).pow(1.06)).toLong()
            predictedTime
        } else {
            null
        }
    }

    private fun getBestPersonalCompetitionResultRace(
        competition: QualificationCompetition,
        raceList: List<Race>
    ): Race? {
        val competitionRaceList = raceList.filter { it.distance == competition.distance }
        return if (competitionRaceList.isNotEmpty()) {
            return competitionRaceList.minWithOrNull(Comparator.comparingLong { it.resultTime })
        } else {
```

```
            null
        }
    }

    private fun getBestRace(raceList: List<Race>): Race? {
        var bestRace: Race? = null
        var bestRaceCeff: Double? = null
        for (competition in competitionList) {
            val bestPersonalCompetitionResultRace =
                getBestPersonalCompetitionResultRace(competition, raceList)
            val bestPersonalCompetitionResultRaceDuration =
                getRaceDurationOrNull(bestPersonalCompetitionResultRace)
            val currentCategory = getCurrentCategory(
                competition,
                bestPersonalCompetitionResultRaceDuration
            )
            val currentCategoryNumber = currentCategory.third
            val nextCategory = getNextCategory(
                competition,
                bestPersonalCompetitionResultRaceDuration
            )
            val bestTimeConditionTimeDifference =
                if (nextCategory.second != null)
bestPersonalCompetitionResultRaceDuration?.minus(
                    nextCategory.second
                )
                else null
            val nextCategoryCeff =
                if (bestTimeConditionTimeDifference != null) {
                    bestTimeConditionTimeDifference.toMillis()
                        .toDouble() / nextCategory.second?.toMillis()!!.toDouble()
```

```
                } else {
                    null
                }
            val raceCeff =
                if (currentCategoryNumber != null && nextCategoryCeff != null)
currentCategoryNumber + nextCategoryCeff
                else null
            if (raceCeff != null && (bestRaceCeff == null || raceCeff > bestRaceCeff)) {
                bestRaceCeff = raceCeff
                bestRace = bestPersonalCompetitionResultRace
            }
        }
        return bestRace
    }

    private fun getRaceDurationOrNull(race: Race?): Duration? {
        return if (race != null)
            Duration.ofMillis(race.resultTime)
        else null
    }

    private fun getTwoLastWeeksRaces(): List<Race> {
        val raceList = RunHelperDB.getInstance(context).raceDao().getAll().toMutableList()
        val olderThan2WeeksPredicate = Predicate { race: Race ->
            val todayMillis = System.currentTimeMillis()
            val day2WeeksAgo = todayMillis - Duration.ofDays(14).toMillis()
            race.dateTime < day2WeeksAgo
        }
        raceList.removeIf(olderThan2WeeksPredicate)
        return raceList
    }
```

```
private fun getCurrentCategory(
    competition: QualificationCompetition,
    bestResult: Duration?
): Triple<String, Duration?, Int?> {
    if (bestResult != null) {
        for (categoryCondition in competition.categoriesConditionsListDesc) {
            if (getCurrentCategoryCondition(categoryCondition, bestResult)) {
                return categoryCondition
            }
        }
        return nullCategory()
    } else {
        return nullCategory()
    }
}

private fun getCurrentCategoryCondition(
    category: Triple<String, Duration?, Int?>,
    bestResult: Duration
): Boolean {
    return category.second != null && bestResult.toMillis() <= category.second!!.toMillis()
}

private fun getNextCategory(
    competition: QualificationCompetition,
    bestResult: Duration?
): Triple<String, Duration?, Int?> {
    if (bestResult != null) {
        for (categoryCondition in competition.categoriesConditionsListAsc) {
            if (getNextCategoryCondition(categoryCondition, bestResult)) {
```

```
                    return categoryCondition
                }
            }
            return nullCategory()
        } else {
            return nullCategory()
        }
    }

    private fun getNextCategoryCondition(
        category: Triple<String, Duration?, Int?>,
        bestResult: Duration
    ): Boolean {
        return category.second != null && bestResult.toMillis() > category.second!!.toMillis()
    }

    private fun nullCategory(): Triple<String, Duration?, Int?> {
        return Triple("-", null, 0)
    }
}
```

Файл TrainingProgramListFragment.kt

```
package com.example.runhelper.ui.screen.training_programs

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.example.runhelper.MainActivity
import com.example.runhelper.databinding.FragmentTrainingProgramListBinding
```

```
import com.example.runhelper.db.repo.QualificationCompetitionRepository
import com.example.runhelper.enum.Screen
import com.example.runhelper.preferences.RunHelperPreferences
import
com.example.runhelper.ui.screen.training_programs.TrainingProgramFragment.Companion.DISTANCE_KEY
import com.example.runhelper.ui.view.DistanceView

class TrainingProgramListFragment : Fragment() {

    private var _binding: FragmentTrainingProgramListBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentTrainingProgramListBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val gender = RunHelperPreferences.getInstance(requireContext()).getGender()!!
        val competitionList =
            QualificationCompetitionRepository.getCompetitionList(gender)
        for (competition in competitionList) {
            val distanceView =
                DistanceView(
                    requireContext(),
                    competition.distance,
                    clickHandler = this::onDistanceViewClick
```

```
                )
            binding.trainingProgramsDistancesLayout.addView(distanceView)
        }
    }

    private fun onDistanceViewClick(distance: Int) {
        val bundle = Bundle()
        bundle.putSerializable(DISTANCE_KEY, distance)
        (activity as MainActivity).navigateToScreen(Screen.TRAINING_PROGRAM, bundle)
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Файл TrainingProgramFragment.kt

```
package com.example.runhelper.ui.screen.training_programs

import android.os.Bundle
import android.view.ContextThemeWrapper
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TableRow
import android.widget.TextView
import androidx.fragment.app.Fragment
import com.example.runhelper.R
import com.example.runhelper.databinding.FragmentTrainingProgramBinding
import com.example.runhelper.db.model.TrainingProgramWeek
```

```
import com.example.runhelper.db.repo.TrainingProgramsRepository


class TrainingProgramFragment : Fragment() {
    companion object {
        const val DISTANCE_KEY = "distance"
    }

    private var _binding: FragmentTrainingProgramBinding? = null
    private val binding get() = _binding!!

    private var distance: Int? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentTrainingProgramBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        setDistance()
        composeTable()
    }

    private fun composeTable() {
        if (distance != null) {
            val program = getProgram(distance!!)
            for (week in program) {
                binding.trainingProgramLayout.addView(composeRow(week))
```

```
        }
    }
}

private fun setDistance() {
    distance = if (arguments != null && arguments!!.containsKey(DISTANCE_KEY)) {
        arguments!!.getSerializable(DISTANCE_KEY) as Int
    } else {
        null
    }
    binding.trainingProgramDistance.text =
        context?.getString(R.string.training_program_name_placeholder, distance.toString())
}

private fun getProgram(distance: Int): List<TrainingProgramWeek> {
    return when (distance) {
        60 -> TrainingProgramsRepository.trainingProgram60m
        100 -> TrainingProgramsRepository.trainingProgram100m
        200 -> TrainingProgramsRepository.trainingProgram200m
        300 -> TrainingProgramsRepository.trainingProgram300m
        400 -> TrainingProgramsRepository.trainingProgram400m
        800 -> TrainingProgramsRepository.trainingProgram800m
        1000 -> TrainingProgramsRepository.trainingProgram1000m
        1500 -> TrainingProgramsRepository.trainingProgram1500m
        3000 -> TrainingProgramsRepository.trainingProgram3K
        5000 -> TrainingProgramsRepository.trainingProgram5K
        10000 -> TrainingProgramsRepository.trainingProgram10K
        21097 -> TrainingProgramsRepository.trainingProgramHalfMarathon
        42195 -> TrainingProgramsRepository.trainingProgramMarathon
        else -> TrainingProgramsRepository.trainingProgram100m
    }
```

```
}

private fun composeRow(
    week: TrainingProgramWeek,
): TableRow {
    val row =
        TableRow(ContextThemeWrapper(activity, R.style.TableRow))
    row.layoutParams = TableRow.LayoutParams(
        TableRow.LayoutParams.WRAP_CONTENT,
        TableRow.LayoutParams.WRAP_CONTENT
    )

    getTableTv(text = week.number.toString(), row)
    getTableTv(text = week.monday, row)
    getTableTv(text = week.tuesday, row)
    getTableTv(text = week.wednesday, row)
    getTableTv(text = week.thursday, row)
    getTableTv(text = week.friday, row)
    getTableTv(text = week.saturday, row)
    getTableTv(text = week.sunday, row)

    return row
}

private fun getTableTv(
    text: String?,
    row: TableRow
) {
    val tvParams = TableRow.LayoutParams(
        resources.getDimension(R.dimen.qualification_table_column_width).toInt(),
        TableRow.LayoutParams.MATCH_PARENT
```

```
        )
        val tv =
            TextView(ContextThemeWrapper(activity, R.style.QualificationTableText))
        tv.apply {
            this.text = text
            layoutParams = tvParams
            row.addView(this)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Файл QualificationTableFragment.kt

```
package com.example.runhelper.ui.screen

import android.os.Bundle
import android.view.ContextThemeWrapper
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TableRow
import android.widget.TextView
import androidx.fragment.app.Fragment
import com.example.runhelper.R
import com.example.runhelper.databinding.FragmentQualificationTableBinding
import com.example.runhelper.db.model.QualificationCompetition
import com.example.runhelper.db.repo.QualificationCompetitionRepository
```

```
import com.example.runhelper.preferences.RunHelperPreferences
import com.example.runhelper.util.DurationFormat
import java.time.Duration

class QualificationTableFragment : Fragment() {

    private var _binding: FragmentQualificationTableBinding? = null
    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentQualificationTableBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        composeTable()
    }

    private fun composeTable() {
        val gender = RunHelperPreferences.getInstance(requireContext()).getGender()!!
        val competitionList =
            QualificationCompetitionRepository.getCompetitionList(gender)
        binding.qualificationTableLayout.addView(composeCategoryLabelsRow())
        for (competition in competitionList) {
            binding.qualificationTableLayout.addView(composeRow(competition))
        }
    }
```

```
private fun composeCategoryLabelsRow(): TableRow {
    val row =
        TableRow(ContextThemeWrapper(activity, R.style.TableRow))
    row.layoutParams = TableRow.LayoutParams(
        TableRow.LayoutParams.WRAP_CONTENT,
        TableRow.LayoutParams.WRAP_CONTENT
    )

    val emptyTvParams = TableRow.LayoutParams(
        TableRow.LayoutParams.WRAP_CONTENT,
        TableRow.LayoutParams.MATCH_PARENT
    )
    val emptyTv =
        TextView(ContextThemeWrapper(activity, R.style.QualificationTableLabelText))
    emptyTv.apply {
        text = ""
        layoutParams = emptyTvParams
        row.addView(this)
    }

    val tvParams = TableRow.LayoutParams(
        resources.getDimension(R.dimen.qualification_table_column_width).toInt(),
        TableRow.LayoutParams.MATCH_PARENT
    )
    for (category in QualificationCompetition.categoriesNames.reversed()) {
        val tv =
            TextView(ContextThemeWrapper(activity, R.style.QualificationTableLabelText))
        tv.apply {
            text = category
            layoutParams = tvParams
```

```
            row.addView(this)
        }
    }
    return row
}

private fun composeRow(
    competition: QualificationCompetition,
): TableRow {
    val row =
        TableRow(ContextThemeWrapper(activity, R.style.TableRow))
    row.layoutParams = TableRow.LayoutParams(
        TableRow.LayoutParams.WRAP_CONTENT,
        TableRow.LayoutParams.WRAP_CONTENT
    )

    val distanceTvParams = TableRow.LayoutParams(
        TableRow.LayoutParams.WRAP_CONTENT,
        TableRow.LayoutParams.MATCH_PARENT
    )

    val distanceTv =
        TextView(ContextThemeWrapper(activity, R.style.QualificationTableText))
    distanceTv.apply {
        text = "${competition.distance} м"
        layoutParams = distanceTvParams
        row.addView(this)
    }

    val tvParams = TableRow.LayoutParams(
        resources.getDimension(R.dimen.qualification_table_column_width).toInt(),
```

```
            TableRow.LayoutParams.MATCH_PARENT
        )

        getResultTableTv(tvParams, duration = competition.msumk.second, row)
        getResultTableTv(tvParams, duration = competition.msu.second, row)
        getResultTableTv(tvParams, duration = competition.kmsu.second, row)
        getResultTableTv(tvParams, duration = competition.first.second, row)
        getResultTableTv(tvParams, duration = competition.second.second, row)
        getResultTableTv(tvParams, duration = competition.third.second, row)
        getResultTableTv(tvParams, duration = competition.firstJunior.second, row)
        getResultTableTv(tvParams, duration = competition.secondJunior.second, row)
        getResultTableTv(tvParams, duration = competition.thirdJunior.second, row)

        return row
    }

    private fun getResultTableTv(
        tvParams: TableRow.LayoutParams,
        duration: Duration?,
        row: TableRow
    ) {
        val tv =
            TextView(ContextThemeWrapper(activity, R.style.QualificationTableText))
        tv.apply {
            text = getDurationFormatString(duration)
            layoutParams = tvParams
            row.addView(this)
        }
    }

    private fun getDurationFormatString(duration: Duration?): String {
```

```
        return if (duration != null) {
            DurationFormat.getDurationString(duration)
        } else {
            "-"
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```