МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет ракетно – космічної техніки

Кафедра вищої математики та системного аналізу

Пояснювальна записка до дипломної роботи

магістра

(освітньо-кваліфікаційний рівень)

на тему «Фільтрація пробіжок у додатку «Фітнес асистент»»

ХАІ.405.463м.6.040303.зачетка.20О

Виконав: студент 6 курсу групи 463-м
спеціальності 124 «Системний аналіз»
освітньої програми « Системний аналіз і управління»
Тараненко В.С.
(прізвище та ініціали студента)
Керівник: Макарічев В.О.
(прізвище й ініціали)
Рецензент: Колодяжний В.М.
(прізвище й ініціали)

Харків – 2020

Міністерство освіти і науки України
Національний аерокосмічний університет ім. М.Є.Жуковського
«Харківський авіаційний інститут»

Факультет ракетно-космічної техніки
Кафедра вищої математики та системного аналізу
(повне найменування)
Рівень вищої освіти другий (магістерський)
Спеціальність 124 «Системний аналіз»
(код та найменування)
Освітня програма «Системний аналіз і управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Ніколаєв О.Г.
(підпис) (ініціали та прізвище)
«___» _____ 20___ р.

ЗАВДАННЯ
ДО ДИПЛОМНОЇ РОБОТИ СТУДЕНТа
Тараненку Вадиму Сергійовичу
(Прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Фільтрация пробіжок у додатку «Фітнес асистент»
керівник роботи к. ф.-м. наук, доцент Макарічев Віктор Олександрович
(Прізвище, ім'я, по батькові, науковий ступінь, Вчене звання)
затверджені наказом вищого навчального закладу від "13" листопада 2020 року № 1819 уч.
2. Термін подання студентом проекту (роботи) "___" _____ року
3. Вихідні дані до проекту (роботи) створення програмного продукту
4. Зміст пояснювальної записки (перелік завдань, які потрібно розв'язати):

1. Опис автоматизованої системи для розподілу завдань з точки зору системного аналізу.
2. Модельний вигляд об'єкта дослідження.
3. Вибір платформи та середи розробки програмного продукту.
4. Розробка програмного продукту.
5. Тестування та виправлення помилок у функціоналі програмного продукту.
6. Визначення розрахунку собівартості та ціни програмного продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

_____
_____
_____

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання приняв
Загальна частина			
Економічна частина			

Нормоконтроль _____ _____ «____» _____20___ р.
(підпис) (ініціали та прізвище)

7.Дата видачі завдання
_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Рядків виконання етапів проекту (роботи)	Примітка
1	Огляд апріорної інформації по заданій тематиці		
2	Визначення актуальності проблеми, цілей, завдань та методів дослідження		
3	Опис мобільного додатку з точки зору системного аналізу		
4	Модельний вигляд об'єкта дослідження		
5	Вибір платформи та середовища розробки програмного продукту		
6	Розробка програмного продукту, який фільтрує дані		
7	Тестування та виправлення помилок у функціоналі програмного продукту		
8	Визначення розрахунку собівартості та ціни програмного продукту		
9	Оформлення пояснювальної записки		

студент _____ _____
(Підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ _____
(Підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи містить: 83с., 36 рис.,14 табл., 23 джерела.

АНАЛІЗ, МОБІЛЬНИЙ ДОДАТОК, СИСТЕМА, СИСТЕМНИЙ АНАЛІЗ, ФІЛЬТРАЦІЯ.

Об’єктом дослідження є фільтр даних для мобільного додатку “Фітнес-асистент”.

Мета дослідження: розробити фільтр даних для мобільного додатку «Фітнес-асистент».

Предмет дослідження: процес фільтрації даних.

Задачі дослідження:

- аналіз інформаційних джерел по створенню фільтрів для мобільних додатків, формування основних вимог та рекомендацій для розробки фільтру для мобільного додатку;

- визначення та аналіз вхідних і вихідних параметрів;

Методи дослідження: методи та засоби комп'ютерних інформаційних джерел та технологій; методи системного аналізу.

Доводиться системність об'єкта дослідження, проводиться структурний, функціональний, інформаційний та класифікаційний опис. Визначаються основні вхідні та вихідні параметри. Розглядаються та аналізуються існуючі алгоритми фільтрації даних в аналогічних мобільних додатках.

РЕФЕРАТ

Пояснительная записка к дипломной работе содержит: 83 стр., 14 табл., 36 рис., 23 источников литературы.

АНАЛИЗ, ФИЛЬТРАЦИЯ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, СИСТЕМА, СИСТЕМНЫЙ АНАЛИЗ.

Объектом исследования является фильтр данных для мобильного приложения "Фитнес-ассистент».

Цель исследования: разработать фильтр данных для мобильного приложения «Фитнес-ассистент».

Предмет исследования: процесс фильтрации данных.

Основные задачи:

- анализ информационных источников по созданию фильтров для мобильных приложений, формирование основных требований и рекомендаций для разработки фильтра для мобильного приложения;

- определение и анализ входных и выходных параметров;

Методы исследования: методы и средства компьютерных информационных источников и технологий; методы системного анализа.

Доказывается системность объекта исследования, проводится структурное, функциональное, информационное и классификационное описание. Определяются основные входные и выходные параметры. Рассматриваются и анализируются существующие аналогичные алгоритмы фильтрации данных в мобильном приложении.

PAPER

Explanatory note to the thesis work contains: 83p., 36pic., 14 tab., 23 sources of literature.

ANALYSIS, FILTRATION, MOBILE APP, SYSTEM, SYSTEM ANALYSIS.

The object of the study is a data filter for the mobile application "Fitness Assistant".

The purpose of the study: to develop a data filter for the mobile application "Fitness Assistant".

Subject of research: the process of data filtering.

Main tasks:

- analysis of information sources on the creation of filters for mobile applications, the formation of basic requirements and recommendations for the development of a filter for a mobile application;

- definition and analysis of input and output parameters;

Investigation methods: methods and means of computer information sources and technologies; methods of system analysis.

Proved systematic of the project, structure, functional, informational and classification description is carried out. The main input and output parameters are determined. Existing similar filtering for mobile applications are considered and analyzed.

ЗМІСТ

ВСТУП

В мобільному додатку “Фітнес-асистент” немає інструменту для фільтрації даних маршруту. Це дуже важливо, тому що наприклад: людині із недостатньою фізичною підготовкою не потрібні довгі та важкі маршрути, хтось буде обирати шлях поважче а хтось покоротше. Комусь треба важкі фізичні навантаження, а комусь треба просто підтримувати свою фізичну форму. Саме тому було прийнято рішення розробити систему фільтрації маршрутів для мобільного додатку “Фітнес-асистент”.

У сучасному світі існує багато різноманітних спортивних мобільних додатків із своїм функціоналом. Саме функціонал та швидко працьовитість програми відрізняє один мобільний додаток від іншого, а програма з фільтрації маршрутів є ключовою, тому вона і відрізняє цей мобільний додаток від інших.

В даній роботі основний акцент є на створення комфорту в користуванні мобільним додатком. Це все для того, аби користувачі відчули швидку роботу із обробкою їх запитів.

У роботі наведені результати аналізу аналогічних засобів для фільтрації даних, показані їх переваги та недоліки.

У першому розділі даної дипломної роботи представлена інформація про актуальність даної теми, порівняння недоліків і переваг аналогічних систем. На основі розглянутих інформаційних джерел були складені об'єкт, мета, предмет, завдання і методи дослідження.

У другому розділі було проведено морфологічний, функціональний і інформаційний опис системи. Описано структуру предмета дослідження. Також було представлено доказ того, що об'єкт дослідження є об'єктом з точки зору системного аналізу.

У третьому розділі описана модель об'єкта дослідження, описані елементи моделі і визначені вхідні та вихідні величини даного об'єкта.

У четвертому розділі обґрунтований вибір середовища розробки програмного забезпечення, описано процес розробки.

У п'ятому розділі проводиться розрахунок економічної складової програмного продукту.

У висновку описані результати наведеної роботи.

1 АНАЛІЗ ФІЛЬТРУ ДАНИХ В МОБІЛЬНОМУ ДОДАТКУ

1.1 Визначення та основні поняття

Фільтр (алгоритм сортування) - це алгоритм, який розміщує елементи списку в певному порядку. Найчастіше вживані замовлення - це числовий порядок та лексикографічний порядок. Ефективне сортування важливо для оптимізації ефективності інших алгоритмів (таких як алгоритми пошуку та злиття), які вимагають, щоб вхідні дані знаходились у відсортованих списках.

Сортування також часто корисно для канонізації даних та для отримання зручного для читання результату. Більш формально результат будь-якого алгоритму сортування повинен відповідати двом умовам:

Аналіз — розчленування предмету пізнання, абстрагування його окремих сторін чи аспектів.

1. Метод дослідження, який вивчає предмет, уявно чи реально розчленовуючи його на складові елементи, як-от частини об'єкта, його ознаки, властивості, відношення, відтак розглядає кожен з виділених елементів окремо в межах єдиного цілого; *протилежний метод* — синтез.
2. Уточнення логічної форми (побудови, структури) міркування засобами формальної логіки.
3. В широкому розумінні — наукове дослідження взагалі.
4. Визначення складу і властивостей якої-небудь речовини, дослідження їх.

Мобільний додаток — програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях. Багато мобільних застосунків встановлені на самому пристрої або можуть бути завантажені на нього з онлайн магазинів мобільних застосунків, таких як app Store, Google Play, Windows Phone Store та інших, безкоштовно або за плату.

Системний аналіз — науковий метод пізнання, що являє собою послідовність дій з встановлення структурних зв’язків між змінними або елементами досліджуваної системи. Спирається на комплекс загальнонаукових, експериментальних, природничих, статистичних, математичних методів.

Систе́ма — множина взаємопов'язаних елементів, що утворюють єдине ціле, взаємодіють із середовищем та між собою, і мають мету.

1.2 Огляд літератури та інших інформаційних джерел

Огляд досліджень і публікацій виявив, існування аналогів, але вони мають недолік, такий як: відсутність фільтрів для саме цього додатку, та з даними саме цього застосунку.

Наприклад:

- фільтрація даних в, що працює оффлайн – «Maps.Me» (безкоштовне);
- фільтрація даних в «Google Maps» (безкоштовне);
- фільтрація даних в «CityMaps 2Go» (платне);
- фільтрація даних в «Map My Ride»(безкоштовне);
- фільтрація даних в – «Rome2Rio»(безкоштовне);
- фільтрація даних в трекері для їзди на велосипеді та бігу – «Strava»(безкоштовне);

У результаті виявлення проблем, які представлені у вигляді дерева проблем на рисунку 1.1, що виникають при аналізі фільтрів мобільних додатків, можна виділити такі цілі:

- сформулювати основні вимоги та рекомендації для розробки мобільного додатку;

− визначити вхідні та вихідні параметри;

− проаналізувати вхідні та вихідні параметри;

− оптимізувати принцип роботи існуючих аналогічних мобільних додатків.

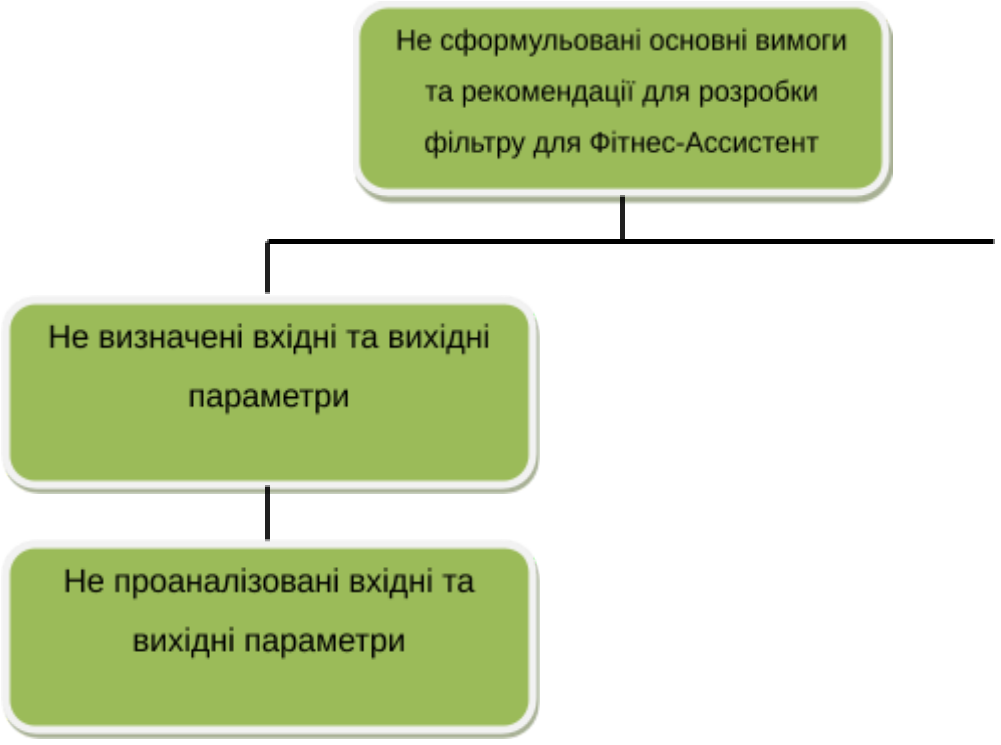

Рисунок 1.1 – Дерево проблем

Такі цілі можна уявити як дерево цілей, наданого на малюнку 1.2.

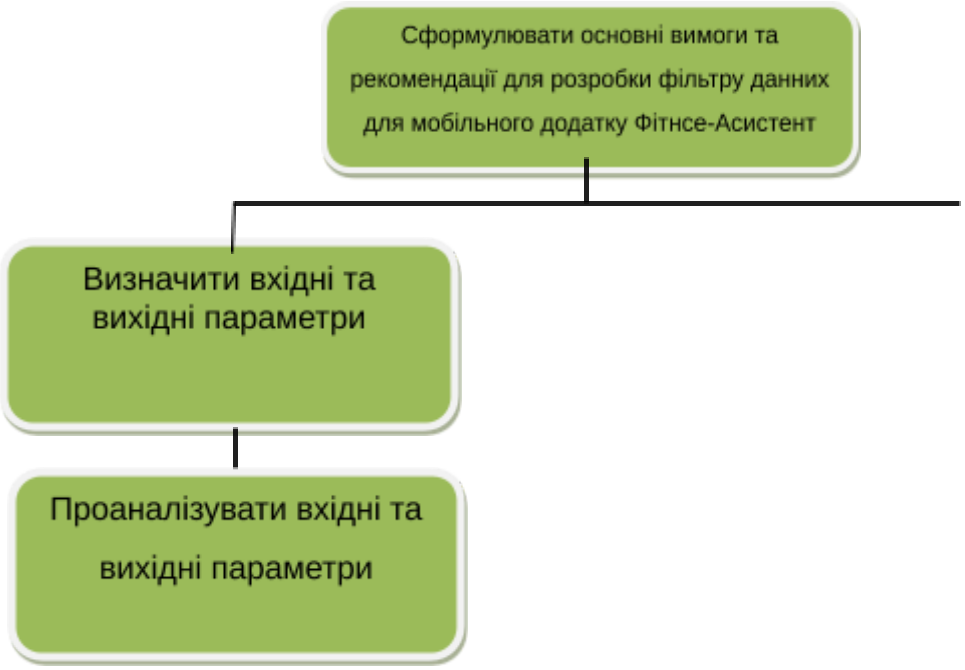

Рисунок 1.2 – Дерево цілей

1.3 Актуальність та особливості дослідження

Актуальність задачі можливостей створення фільтрів для мобільного застосування «Фітнес-асистент» визначається тим, що розробка такої системи робить фактично зручним підбір оптимальних маршрутів для заданого користувача. а це впливає на такі важливі чинники як:

1. Зручність.
2. Популярність мобільного додатку.
3. Індивідуальний підхід до кожного користувача.
4. Надання більш точної інформації.

Все це сприяє покращенню життя користувачів мобільного додатку і дозволяє їм знаходити маршрути індивідуально під свої вимоги.

Особливостями фільтра мобільного додатку повинно бути те, що він:

- розрахований на будь-якого користувача
- дозволяє самостійно за допомогою фільтру обирати найбільш відповідний маршрут;
- дозволить скоротити час пошуку свого маршруту;
- дозволить підвищити ефективність додатку;

1.4 Мети й завдання дослідження

На основі розглянутих інформаційних джерел були складені об'єкт, предмет, мета і завдання дослідження,а також методи дослідження.

Об'єкт дослідження*:* фільтр даних для мобільного додатку “Фітнес-асистент”.

Предмет дослідження*:* процес фільтрації даних.

Мета дослідження: розробити фільтр даних для мобільного додатку «Фітнес-асистент».

Завдання дослідження:

Аналіз інформаційних джерел по створенню фільтрів для мобільних додатків, формування основних вимог та рекомендацій для розробки фільтру для мобільного додатку;

визначення та аналіз вхідних і вихідних параметрів;

Методи дослідження:

- методи системного аналізу
- методи та засоби комп'ютерних інформаційних технологій

2 ОПИС СИСТЕМИ МОБІЛЬНОГО ДОДАТКУ З ТОЧКИ ЗОРУ СИСТЕМНОГО АНАЛІЗУ

2.1 Морфологічний опис

Структурний опис фільтру даних для мобільного додатку, виконано на підставі аналізу інформаційних джерел [4 - 8].

З цих джерел слідує:

1) робота фільтру для мобільного додатку «Фітнес-асистент» починається з формування вхідних параметрів;

2) на підставі вхідних параметрів виділяються підсистеми визначення поточного місця розташування та вивчення інформації користувача;

3) аналіз фільтру для мобільного додатку працює на основі двох модулів - обробки та аналізу;

4) параметри на виході формуються за допомогою модуля видачі результатів.

Дана схема роботи дозволила уявити систему в морфологічному вигляді.

Склад системи наступний:

1. Фільтр для мобільного додатку

1.1. Система формування вхідних параметрів

1.1.1. Алгоритм сортування даних

1.1.2. Підсистема обробки даних

1.2. Система аналізу вхідних даних

1.2.1.Модуль обробки вхідних даних

1.2.2.Модуль аналізу оброблених даних

1.3. Модуль видачі результатів

При цьому представлена система є системою з точки зору системного аналізу, так як вона характеризується такими властивостями:

Емерджентність: розглянута система має властивість емерджентності,

оскільки в цілому система виробляє збір параметрів, аналіз і обробку даних. Таким властивістю не володіє будь-який окремий елемент цієї системи.

Цілісність: система є цілісною. Наприклад, при відсутності модуля обробки буде порушений аналіз системи і це призведе до непрацездатності всієї системи.

Адитивність: система має властивість адитивності, оскільки властивість системи мобільного додатку представляється як сума властивостей елементів системи мобільного додатку.

Прогресуюча факторизація: характеризується тим, що всі елементи залежать один від одного, так як отримати оптимальний результат можна тільки якщо об'єднати всі елементи системи.

Ізоморфізм: дана система має властивості ізоморфізму, так як елементи системи мають подібні властивості.

Тип елементного складу: змішаний тип.

Тип зв'язків: інформаційний, так як зв'язку переносять інформацію між елементами.

Тип структури: дана система має ієрархічною структурою.

2.2 Функціональний аналіз досліджуваного об’єкта

Дана система є багатофункціональною. Функції першого рівня показані в таблиці 2.1. Функції підсистем другого рівня представлені в таблиці 2.2. Функції третього рівня представлені в таблиці 2.3.

Параметри елементів першого рівня представлені в таблиці 2.4.

Параметри елементів другого рівня представлені в таблиці 2.5. Параметри підсистем третього рівня показані в таблиці 2.6

Таблиця 2.1 – Функції системи першого рівня

Код	Елемент	Функція
1	Фільтр для мобільного	Забезпечення формування вихідних

Продовження таблиці 2.1

	додатку	даних

Таблиця 2.2 - Функції підсистем другого рівня

Код	Елемент	Функція
1.1	Система формування вхідних параметрів	Приймає у себе вхідні параметри.
1.2	Система аналізу вхідних даних	Аналіз і обробка вхідних даних
1.3	Модуль видачі результатів	Видає отсортированы дані

Таблиця 2.3 – Функції підсистем третього рівня

Код	Елемент	Функція
1.1.1	Алгоритм сортування даних	Сортує отримані дані
1.2.1	Модуль обробки вхідних даних	Перевіряє вхідні дані на правильність
1.2.2	Модуль аналізу оброблених даних	Аналіз отриманих даних

Таблиця 2.4 – Параметри систем першого рівня

Код	Елемент	Параметри
1	Фільтр даних	Довжина, важкість маршруту

Таблиця 2.5 – Параметри підсистем другого рівня

Код	Елемент	Параметри
1.1	Система формування вхідних параметрів	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту
1.2	Система аналізу вхідних даних	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту
1.3	Модуль видачі результатів	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту Картинка маршруту

Таблиця 2.6. – Параметри підсистем третього рівня

Код	Елемент	Параметри
1.1.1	Алгоритм сортування даних	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту
1.1.2	Підсистема обробки даних	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту

Продовження таблиці 2.6

1.2.1	Модуль обробки вхідних даних	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту Локація маршруту Картинка маршруту
1.2.2.	Модуль аналізу оброблених даних	Номер маршруту Довжина маршруту Важкість маршруту Чистота маршруту

Підсумкове і сумарна кількість функцій:

- функції першого рівня –1;
- функції другого рівня – 3;
- функції третього рівня –4;

Загальні характеристики системи:

Загальні характеристики заданої системи наведені в таблиці 2.7.

Таблиця 2.7. – Загальні характеристики системи

Функціональність	Ранг	Фактори	
		Системаруйнівні	Системоутворюючі
Багатофункціональна	Обслуговування	Неправильні вхідні дані	Правильні вхідні дані

2.3 Інформаційний опис системи

Елементи системи:

1) Система формування вхідних параметрів

2) Алгоритм сортування даних

3) Підсистема обробки даних

4) Система аналізу вхідних даних

5) Модуль обробки вхідних даних

6) Модуль аналізу оброблених даних

7) Модуль видачі результатів

2. Властивості системи

Загальні властивості заданої системи наведені в таблиці 2.8.

1. Середньогеометричне число властивостей на один елемент:

$$a = \sqrt[7]{a_1 \cdot \dots \cdot a_7} = \sqrt[7]{1 \cdot 1 \cdot 2 \cdot 2 \cdot 1 \cdot 1 \cdot 2} = \sqrt[7]{8} \approx 1,5$$

2. Структура об'єкта мережева

Структура системи у вигляді графа представлена на рисунку 2.1

Таблиця 2.8 – Властивості системи

№ п/п	Найменування	Позначення	Кількість властивостей	Примітка
1	Система формування вхідних параметрів	a_1	1	1 (1) забезпечення визначення вхідних даних
2	Алгоритм сортування даних	a_2	1	1 (2) сортування даних
3	Підсистема обробки даних	a_3	2	1 (3) введені користувачем параметрів 1 (4) вивчення параметрів користувача

Продовження таблиці 2.8

4	Система аналізу вхідних даних	a_4	2	1 (5) забезпечення аналізу даних 1 (6) забезпечення обробки даних
5	Модуль обробки вхідних даних	a_5	1	1 (7) обробка даних
6	Модуль аналізу оброблених даних	a_6	1	1(8) аналіз даних
7	Модуль видачі результатів	a_7	2	1 (9) забезпечення виведення оброблених і проаналізованих даних 1 (10) висновок оброблених і проаналізованих даних

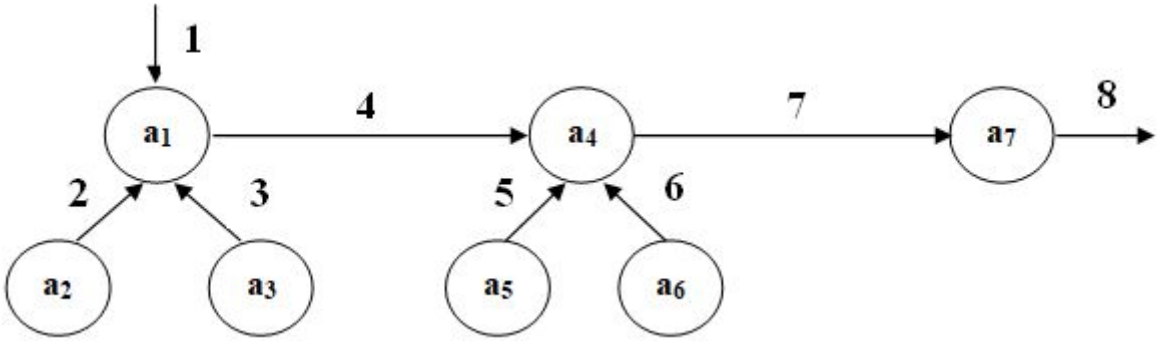

Рисунок 2.1 – Граф

3. Зв'язки системи між елементами:

а) Сполучні

2- алгоритм сортування даних – система формування вхідних параметрів

3- підсистема вивчення інформації користувача – система формування вхідних параметрів

4-Підсистема обробки даних – система аналізу вхідних даних

5- модуль обробки вхідних даних – система аналізу вхідних даних

6- модуль аналізу оброблених даних – система аналізу вхідних даних

7- система аналізу вхідних даних – модуль видачі результатів

Б) Організуючі

1- зовнішнє середовище – система створення вхідних параметрів

8- модуль видачі результатів – зовнішнє середовище

Зв'язки системи представлені на таблиці 2.9.

Середньогеометричне число зв'язків на один елемент:

$$Y = \sqrt[7]{4 \cdot 1 \cdot 1 \cdot 4 \cdot 1 \cdot 1 \cdot 2} = \sqrt[7]{32} \approx 1,643$$

Таблиця 2.9 - Зв’язки системи

№ п/п	Найменування	Кількість зв’язків
1	Система формування вхідних параметрів	4
2	Система формування вхідних параметрів	1
3	Алгоритм сортування даних	1
4	Підсистема обробки даних	4
5	Система аналізу вхідних даних	1

Продовження таблиці 2.9

6	Модуль обробки вхідних даних	1
7	Модуль видачі результатів	2
2	Система формування вхідних параметрів	1

2.4 Класифікаційний опис системи

Таблиця 2.10 – Класифікаційний опис системи

№ п / п	Ознака класифікації	Тип системи за ознакою	Визначення
1	по зв'язку системи з навколишнім середовищем	закрита	не взаємодіє з навколишнім середовищем
2	за походженням	штучна	створена людиною
3	за об'єктивністю існування	реальна	складається з штучних (НЕ живих) об'єктів; застосовується людиною в реальному житті

Продовження таблиці 2.10

4	за типом опису законів функціонування	система типу чорний ящик»	користувачу не потрібно знати закони функціонування
5	за способом управління системою	з комбіновани м керуванням	система підпорядковується блоку управління з системи, а так само зовнішнього управління
6	за дією	технічна	дана система - сукупність взаємопов'язаних фізичних елементів
7	по централізації	централізован а система	не існує елемента, який є головним у функціонуванні системи
8	по однорідності структури	різнорідна	елементи системи не взаємозамінні
9	за типом складності	складна	має багато елементів і складні зв'язки
10	по мірній	багатовимірн а	має багато входів і виходів
11	по організованості	добре організована	визначені всі елементи, зв'язки і залежності між елементами і цілями системи
12	по обумовленості дії	детермінован а	вхід об'єкта однозначно визначає його вихід
13	по лінійності	лінійна	закон функціонування системи можна описати лінійним рівнянням

Продовження таблиці 2.10

14	по безперервності	безперервна	безперервний процес управління даними

3 МОДЕЛЬНЕ УЯВЛЕННЯ ОБ'ЄКТА ТА ДОСЛІДЖЕННЯ ЙОГО ХАРАКТЕРИСТИК

3.1 Короткий огляд основних алгоритмів і розрахункових формул, що використовуються при роботі

Сортування включенням або сортування вставленням — простий алгоритм сортування на основі порівнянь. На великих масивах є значно менш ефективним за такі алгоритми, як швидке сортування, пірамідальне сортування та сортування злиттям. Однак, має цілу низку переваг:

- простота у реалізації
- ефективний (зазвичай) на маленьких масивах
- ефективний при сортуванні масивів, дані в яких вже непогано
- відсортовані: продуктивність рівна O(n + d), де d — кількість інверсій
- на практиці ефективніший за більшість інших квадратичних алгоритмів (O(n2)), як то сортування вибором та сортування бульбашкою: його швидкодія рівна n2/4, і в найкращому випадку є лінійною
- є стабільним алгоритмом

Більшість людей при сортуванні колоди гральних карт, використовують метод, схожий на алгоритм сортування включенням.

На кожному кроці алгоритму ми вибираємо один з елементів вхідних даних і вставляємо його на потрібну позицію у вже відсортованому списку доти, доки набір вхідних даних не буде вичерпано. Метод вибору чергового елементу з початкового масиву довільний; може використовуватися практично будь-який алгоритм вибору. Зазвичай (і з метою отримання стійкого алгоритму сортування), елементи вставляються за порядком їх появи у вхідному масиві.

Нижче на Рисунку 3.1 представлений алгоритм сортування вставками на Python

```
for i := 2 to arrayLength do
begin
  key := arr[i];
  j := i;
  while (j > 1) and (arr[j - 1] > key) do
      begin
        {обмін елементів}
        tempValue :=  arr[j];
        arr[j] := arr[j - 1];
        arr[j - 1] := tempValue;
        j := j - 1;
      end;
  arr[j] := key;
end;
```

Рисунок 3.1 – Алгоритм сортування вставками на Python

На відміну від сортування пузиркового методу і сортування вибором число операцій порівняння для сортування вставки залежить від вихідної впорядкованості масиву елементів. Если вихідний масив Вже впорядкованим, то число операцій порівняння Рівно n-1. Если масив впорядкування у прямому порядку, то число операцій порівняння Рівно 1/2 (n2 + n) -1, дана в Середньому значенні 1/4 (n2 + n2).

Число операцій обміну буде наступним:

- для найкращого випадку: 2 (n-1);
- в середньому: 1/4 (n2 + 9n-10);
- для найгіршого випадку: 1/2 (n2 + 3n-4);

Таким чином, число операцій обміну для жорсткого випадку буде стояти більшим, як і для сортировок пузиркового методу та сортировки вибором. Число операцій обміну для середнього випадку буде лише трохи меншим. Однак сортировка вставці має дві переваги. По-перше, вона володіє природним підведенням, тобто вона виконується швидше для впорядкованого масиву та дольше всього виконується, коли масив впорядкований в образовному складі.

Це робить сортировку виставкової польової для впорядкування майже відсортованих масивів. По-друге, елементи з однаковими ключами не переставляються: якщо список елементів сортується з використанням двох ключів, то після завершення сортування вставки в попередньому буде впорядковано за двома ключами.

У разі, якщо число порівнянь може бути незначним для певних наборів даних, постійні здвиги масиву вимагають виконання великого числа операційних пересилок. Однак, сортировка вставкою має естетичне введення, вимагає найменшого числа операцій обміну для майже впорядкованого списку та найбільшого числа операцій обмена, коли масив впорядкувати у обраному виробництві.

Використання фільтру даних відбувається наступним чином:

1. Відкриття панелі.
2. Введення користувачем бажаних даних.
3. Обробка даних.
4. аналіз особистих даних користувача.
5. аналіз місця знаходження користувача.
6. Вибір найбільш оптимальних маршрутів для користувача.
7. Виведення підходящих маршрутів на екран.

3.2 Вхідні та вихідні параметри

Вхідні параметри:

1. Номер маршруту
2. Довжина маршруту
3. Важкість маршруту
4. Чистота маршруту

Вихідні параметри:

1. Номер маршруту

2. Довжина маршруту
3. Важкість маршруту
4. Чистота маршруту
5. Локація маршруту
6. Картинка маршруту

3.3 Специфікація вимог по створенню мобільного додатка "Фітнес-асистент"

На основі виявлених проблем, отриманої структури мобільного додатку і побудованої структури системи була укладена специфікація вимог для створення фільтру для мобільного додатку «Фітнес-асистент»:

1. Система повинна бути спрямована на виведення підходящих маршрутів.

2. Система повинна забезпечувати видання бажаних маршрутів.

3. У системи повинен бути вибраний швидкий алгоритм пошуку.

4. У системі має бути графічне представлення даних.

5. Система повинна фіксувати та аналізувати пройдені маршрути користувачів.

6. Система повинна по інформації користувача пропонувати йому найбільш оптимальний варіант маршруту.

7. Система повинна забезпечувати захист даних.

8. Система повинна забезпечувати зберігання даних певний час.

3.4 Модельне уявлення об’єкта дослідження

Головна мета системи фільтру для мобільного додатку в цій роботі - це надання маршрутів які найкраще для цього підходять. Для цього необхідно визначити основні параметри, за якими можна найбільш точніше

отсортурувати маршрут. На основі робіт було встановлено, що для мобільного додатку, що розглядається в цій роботі найбільш придатними для аналізу будуть дані, згадані вище в подглаві «3.2 Вхідні і вихідні величини» [6].

При введенні цих параметрів в системі починають роботу модулі обробки, аналізу і видачі результатів. Обробивши параметри, модуль обробки передає інформацію модулю аналізу, який аналізує її і передає інформацію модулю видачі результатів. Весь процесс представлений на рисунку 3.1.

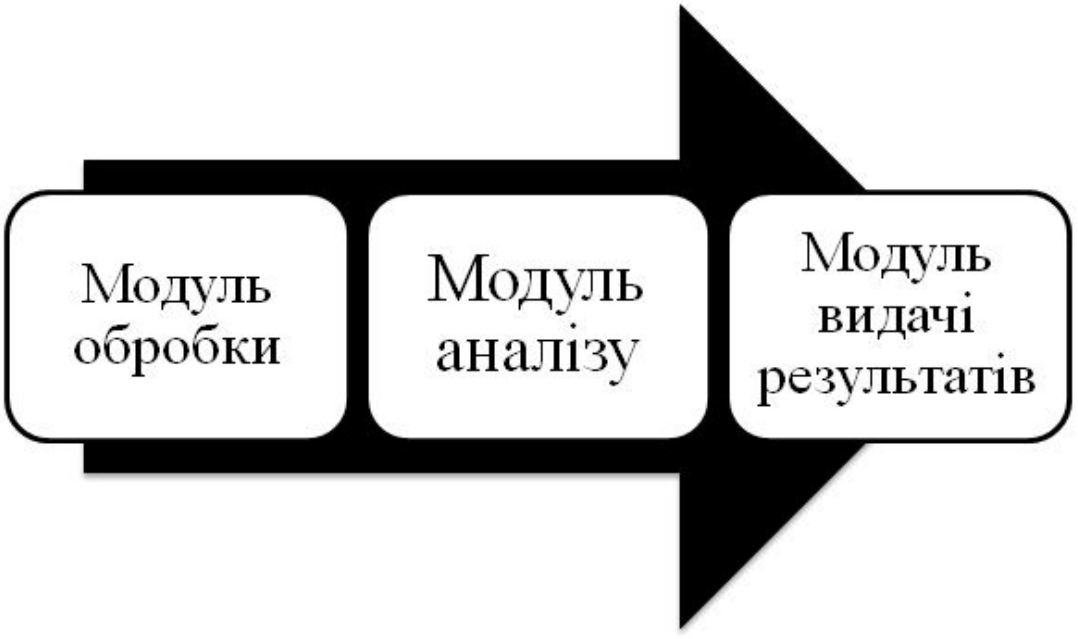

Рисунок 3.2 – Модель роботи системи

3.5 Можливі критерії оптимізації

До критеріїв оптимізації слід віднести:

- правильний підбір вхідних даних;
- конкретний аналіз даних;
- коректна робота модулів системи;

- виведення нових параметрів, що покращують роботу додатку;

автоматизація процесу.

Дані критерії дозволяють більш точно і якісно фільтрувати маршрути та запропонувати користувачам більш оптимальні для них маршрути, також великим плюсом буде створення автоматизованого фільтру на основи штучного інтелекту.

4 ПЕРЕВІРКА АДЕКВАТНОСТІ ЗАПРОПОНОВАНОЇ МОДЕЛІ

4.1 Вибір платформи для розробки програмного продукту

Android- операційна система та платформа для мобільних телефонів

та планшетних комп'ютерів, створена компанією Google на базі ядра Linux. Підтримується альянсом Open Handset alliance (OHa).

Хоча android базується на ядрі Linux, він стоїть дещо осторонь Linux-спільноти та Linux-інфраструктури. Базовим елементом цієї операційної системи є реалізація Dalvik віртуальної машини Java, і все програмне забезпечення і застосування спираються на цю реалізацію Java.

З 2008 року android пережив численні оновлення, які поступово покращували операційну систему, додаючи нові функції, та виправляють помилки у попередніх випусках. І тепер кодове ім'я кожного великого релізу android, починаючи з версії 1.5, являє собою назву якого-небудь десерту.

- 1.5 Cupcake(«Кекс»),
- 1.6 Donut(«Пончик»),
- 2.0 / 2.1 Eclair(«Еклер» або «глазур»),
- 2.2 Froyo(СКОРОЧЕННЯ від «Заморожений йогурт»),
- 2.3 Gingerbread(«Імбірній пряник»),
- 3.0 Honeycomb(«Медові стільнікі»),
- 4.0 Ice Cream Sandwich(«Брикет морозива»),
- 4.1 / 4.2 / 4.3 Jelly Bean(«Желейні боби»),
- 4.4 KitKat(На честь однойменного бренду шоколадних батончиків; Ранее планувалася назва Key Lime Pie),
- 5.0 / 5.1 Lollipop(«Льодяник»),
- 6.0 / 6.1 Marshmallow(«Зефір»),
- 7.0 / 7.1 Nougat- Nougat («Нуга»);

- 8.0 / 8.1 Oreo- Oreo (печиво «Орео»);
- 9.0 Pie- Pie («Пиріг»);

На рисунку 4.1 представлена частота використання версій android.

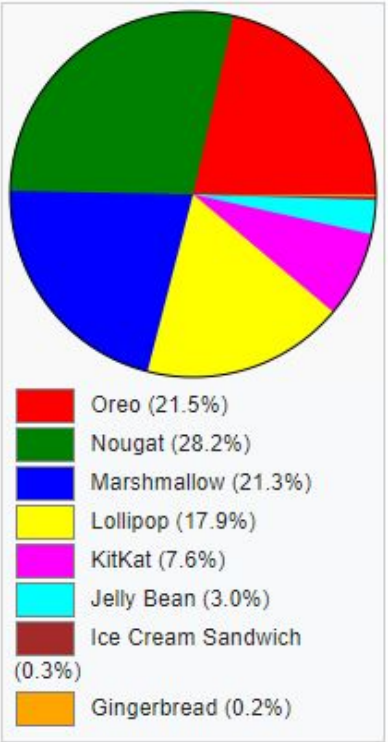

Рисунок 4.1 – Частота використання версій андроід

IOS - це власницька мобільна операційна система від apple. Розроблена спочатку для iPhone, згодом також вдосконалена для використання на iPad.

iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою.

Користувацький інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Multi-Touch. Елементи інтерфейсу управління складаються з повзунків, перемикачів і кнопок. Він призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою скасування, або обертання пристрою у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами.

Станом на 2019 рік інтернет-магазин app Store містить понад 2 мільйони застосунків для iOS, які були завантажені понад 15 мільярдів разів. Станом на травень 2010 року, iOS становив 15,4 % ринку операційних систем для смартфонів, третій після Symbian і Blackberry.

Хронологія версій IOS:

- 1.0- Червень 2007 року, перша версія
- 2.0- 11 липня 2008 року, підтримка iPhone SDK и app Store,3G,GPS
- 3.0- 17 червня 2009
- 4.0- 21 червня 2010 (анонсована 7 червня 2010). Понад 100 Нових функцій, включаючи багатозадачність понад 1500 Нових API для розробників застосунків. Сумісна із Phone 3G,iPhone 3G S, IPhone 4,iPod Touch Іншого, третього та четвертого поколінь.
- 5.0 (build 9a334) - 12 жовтня 2011. Нова версія операційної системи була анонсована 6 червня 2011 во время конференции WWDC. Оголошено про більш чем 200 Нових функцій, включаючі Зміни в режимі оповіщень, можливість оновлення ПО без использование комп'ютера, програму iMessage, «натуральну» інтеграцію зТ witter та понад 1500 Нових API для розробників Додатків. Сумісна з iPhone 3GS, iPhone 4, iPhone 4S, iPod Touch третього и четвертого поколінь, а також iPad всіх поколінь.
- 6.0- 19 вересня 2012.
- 7.0- 18 вересня 2013. Фінальний реліз. Має новий сучасний інтерфейс та много корисних функцій.
- 8.0- 17 вересня 2014 року. Покращена стабільність. Включає додаток Health, Який дозволяє слідкувати за здоров'ям. Інтегрований сервіс з пошуку музики Shazam в Siri.
- 9.0- 16 вересня 2015 року. Покращена стабільність. Збільшена продуктивність та безпека. Siri стала розумнішою. Тепер вона предлагает варіанти ще до того, як ви задали питання.

Діаграму заповнення ринку мобільними платформами можно побачити на рисунку 4.2.

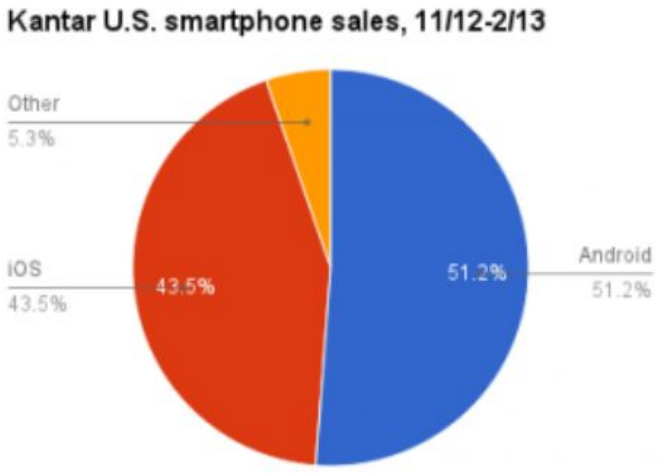

Рисунок 4.2 – Діаграма заповнення ринку мобільними платформами

4.1.1 Створення архітектури програми

Архітектура Clean architecture з блок-схемами показана на рисунку 4.3. та рисунку 4.4

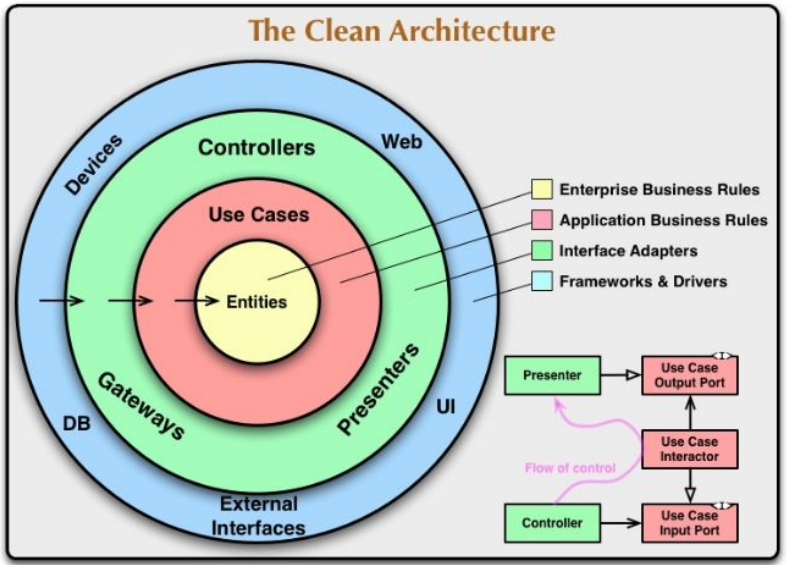

Рисунок 4.3 – Схема Clean architecture

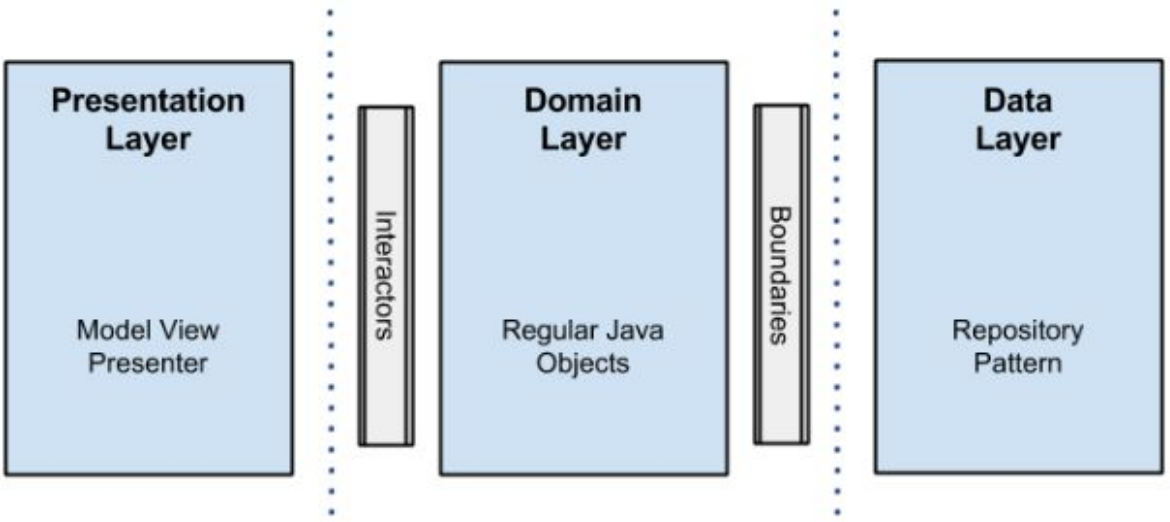

Рисунок 4.4 – Блок-схема Clean architecture

Clean architecture об'єднала в собі ідеї кількох інших архітектурних підходів, які зводятся в тому, что архітектура повинна:

• Бути тестуємою;

• Не залежати від UI;

• Не залежати від БД, зовнішніх фреймворків и бібліотек.

Це досягається поділом на шари та проходження Dependency Rule (правило залежностей).

Dependency Rule говорить нам, що внутрішні шари не повинні залежати від зовнішніх. Тобто наша бізнес-логіка и логіка програми не повинна залежати від презентерів, UI, баз даних та т.і

Це правило дозволяє будувати системи, які будуть простіше підтримуватись, тому що зміни в зовнішніх шарах не торкнути внутрішні шари.

Uncle Bob виділяє 4 шари:

- Entities. Бізнес-логіка загальна для багатьох додатків.
- Use Cases (Interactors). Логіка програми.
- Interface adapters. адаптери між Use Cases та зовнішнім світом.

- Frameworks. Самий зовнішній шар, тут лежить все інше: UI, база даних, http-клієнт, і т.п.

Сюди потрапляють Presenter'и з MVP, а також Gateways (більш популярна назва репозиторії).

Детальніше, що з себе представляють ці шари, ми розглянемо пізніше. а поки що зупинимося на передачі даних між ними. Схему передачі даних в Clean architecture можна побачити на рисунку 4.5.

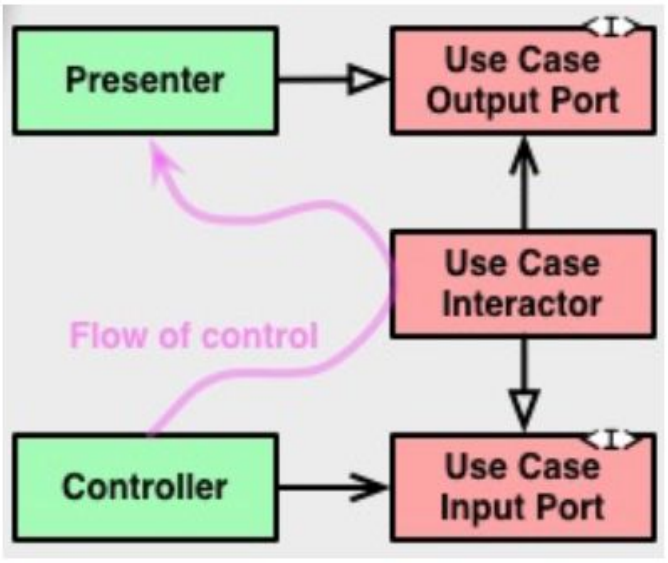

Рисунок 4.5 – Схема передачі даних в Ca

Переходи між шарами здійснюються через Boundaries, тобто через два інтерфейси: один для запиту і один для відповіді. Вони потрібні, щоб внутрішній шар не залежав від зовнішнього (слідуючи Dependency Rule), але при цьому міг передати йому дані.

Обидва інтерфейси відносяться до внутрішнього шару (зверніть увагу на їх колір на зображенні).

Дивіться, Controller викликає метод у InputPort, його реалізує UseCase, а потім UseCase віддає відповідь інтерфейсу OutputPort, який реалізує Presenter. Тобто данні перетнули кордон між шарами, але при цьому всі залежності вказують всередину на шар UseCase'ов.

Щоб залежність була направлена в бік зворотний потоку даних, застосовується принцип інверсії залежностей (Буква D з абревіатури SOLID). Тобто, замість того щоб UseCase прямо залежав від Presenter'a (що порушувало б Dependency Rule), він залежить від інтерфейсу в своєму шарі, а Presenter повинен цей інтерфейс реалізувати.

Точно так схема працює і в інших місцях, наприклад, при зверненні UseCase до Gateway / Repository. Щоб не залежати від сховища, виділяється інтерфейс і кладеться в шар UseCases.

Що ж стосується даних, які перетинають кордони, то це повинні бути прості структури. Вони можуть передаватися як DTO або бути загорнуті в HashMap, або просто бути аргументами при виклику методу. але вони обов'язково повинні бути в формі більш зручною для внутрішнього шару (лежати у внутрішньому шарі).

Крім того не менш важливо відзначити, що ми будемо намагатися писати абсолютно тестований код.

4.1.2 Тестування коду

Модульні тестування - це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестовано. У процедурному програмуванні модулем вважають окрему функцію або процедуру. В об'єктно-орієнтованому програмуванні-інтерфейс,клас. Модульні тести, або unit-тести, розробляються в процесі розробки програмістами, та іноді, тестувальниками білої скриньки(White-box testers).

Зазвичай unit-тести застосовують для того, щоб запевнити, що код відповідає вимогам архітектури та має очікувану поведінку.

Переваги:

Метою модульного тестування є ізоляція кожної частини програми та впевненість у тому, що кожна окрема частина є коректною. Модульний тест

забезпечує жорсткий «контракт», за яким має працювати тестований код. Як результат, це надає деякі переваги. Модульні тестування допомагають знайти помилки раніше у циклі розробки ПЗ, що робить розробку дешевшою та швидшою.

Легкий рефакторинг

Модульні тестування дозволяють програмісту, коли він буде змінювати код (Проводити рефакторинг) бути впевненим, що модуль працює вірно (це - регресивне тестування). Оскільки модульне тестування потребує написання тестів для всіх функцій та методів у програмі, помилки швидко локалізуються та поправляються.

Спрощення інтеграційного тестування

Модульні тестування можуть бути застосовані в інтеграційному тестуванні: Тестування окремих модулів та сукупності цих модулів Робить інтеграційне тестування легшим. але модульне тестування знизу вгору не є інтеграційним тестуванням. Інтеграція з зовнішніми модулями має включатися до інтеграційних тестів, а не до модульних.

Документування

Модульні тести являють собою специфічний вид документації до системи. Розробники можуть подивитися на модульний тест, щоб дізнатися про функції, що виконує модуль, та як його застосовувати.

Unit-тест перевіряє критичні характеристики модулю. Відповідність чи невідповідність цим характеристикам демонструє коректність модуля. Модульний тест «документує» ці критичні характеристики, але не треба покладатися лише на код в документуванні ПЗ під час розробки.

Слід відзначити, що звичайна письмова документація дуже повільно реагує на зміни в коді, тоді як модульні тести завжди відображають поточний стан модуля.

Дизайн

Під час розробки програмного забезпечення методом TDD (Test-driven development), модульний тест стає частиною дизайну. Кожен тест визначає потрібні класи та методи, їх очікувану поведінку.

Важливою перевагою модульного тестування є те, що тести одразу демонструють, відповідає, чи не відповідає реалізація вимогам дизайну. Невірна реалізація не зможе пройти модульні тести.

Відокремлення інтерфейсу від реалізації.

Призначення модульного тесту — тестування єдиного програмного модулю. але часто розробники та тестувальники створюють тести, що не відповідають цій умові. У випадку, якщо клас **a** містить посилання на клас **B**, тестування класу **a** перетікає в тестування класу **B**. Розглянемо приклад.

Є клас, що залежить від бази даних. Модульні тести, які пишуться для нього, можуть взаємодіяти з БД, щоб визначити коректність поведінки класу. Такий підхід є некоректним. У такому випадку модульний тест виходить за межі своєї відповідальності, за межі тестованого класу та перетинає межу іншого класу/процесу/комп'ютерної мережі. Модульні тести, що написані таким чином, перетворюються на інтеграційні тести. Коли ці тести перестають працювати, важко локалізувати помилку.

Правильним підходом можна вважати створення абстрактних інтерфейсів для запитів до БД, а реалізуються ці інтерфейси за допомогою mock-об'єктів (фіктивних об'єктів).

Взаємодія unit-тестів лише з абстрактними інтерфейсами забезпечує ретельніше тестування. Як результат — легше та дешевше супроводження.

Обмеження

Тестування програмного забезпечення не може знайти всіх помилок у програмі. У більшості програм неможливо розрахувати кожен варіант виконання. Це також вірно для модульного тестування. Крім того, модульне тестування, власне, повинен тестувати тільки модулі. Так що цей вид

тестування не може знайти інтеграційні помилки та інші: помилки архітектури, проблеми з витримкою навантажень на ПЗ. Unit-тестування має проводитись разом з іншими видами тестування програмного забезпечення.

Як і будь-який вид тестування, модульне тестування може визначити лише наявність помилок, а не їх відсутність.

Тестування ПЗ — це комбінаторна задача. Наприклад, кожне логічне судження повинно мати не менш двох тестів: один перевіряє результат істинно, а другий хибно. Внаслідок цього, часто на кожний рядок коду доводиться написати від 3 до 5 рядків тесту. Це вимагає багато часу та грошей, які часто не варті результату.

Застосування

Екстремальне програмування

Модульне тестування — це наріжний камінь екстремального програмування, яке покладається на автоматичне створення тестів. автоматична генерація тестів може здійснюватись сторонніми бібліотеками, як то jUnit, або власними продуктами компанії-розробника.

Екстремальне програмування використовує створення модульних тестів для test-driven development. Розробники пишуть модульні тести, які визначають усі вимоги до програми. Ці тести будуть неуспішними, якщо потрібні вимоги ще не реалізовано, або якщо реалізація має дефекти.

Екстремальне програмування дотримується стратегії «тестуйте все, що потенційно може викликати помилку», замість традиційної «тестуйте усі можливі шляхи виконання». Тому більша частина коду покрита unit-тестами, але не весь код.

Важливо відзначити, що код тестів вважається одним з найважливіших артефактів проекту, бо супроводжується так само якісно і в такому ж обсязі, як і функціональний код. Розробники відправляють код тестів до репозиторію разом із тим кодом, що вони тестують. Ретельне тестування в екстремальному програмуванні дозволяє легше супроводжувати код, проводити

його рефакторинг, інтеграцію, вести добру документацію. Ці модульні тести також працюють як форма регресивного тестування.

Технологія

Модульне тестування зазвичай автоматизують, але його можна проводити й вручну. IEEE не надає перевагу якомусь з цих методів.

Мета модульного тестування — ізолювати модуль та визначити його коректність. автоматизоване тестування дозволяє зробити це ефективно та має багато переваг. Якщо тестування було погано заплановано, то необережне ручне тестування може стати ще й інтеграційним тестуванням, що погано.

Щоб повністю реалізувати ефект ізоляції модуля під час автоматизованого тестування, його код виконують у межах особливого фреймворку, без зв'язку з природнім середовищем. Іншими словами, код запускають поза продуктом, або контекстом виклику, для якого його було написано. Такий спосіб тестування показує неважливі залежності між кодом, що тестується та іншими модулями системи.

Використовуючи фреймворк для модульного тестування, розробник задає критерії, за якими визначається успішність тесту. Під час виконання тестування, фреймворк повідомляє про тести, що не задовольняють критеріям. Залежно від важливості невиконаного тесту, фреймворк може зупинити подальше тестування.

Як наслідок, модульне тестування традиційно мотивує програмістів писати код із меншою зв'язністю *Coupling*, та більшою пов'язаністю *Cohesion*.

Фреймворки

Фреймворки для модульного тестування зазвичай не розповсюджуються в комплекті з компіляторами — це програмне забезпечення пишуть окремо. Вони розроблені для багатьох мов програмування й допомагають спростити процес тестування. Серед відомих фреймворків є проекти open source, наприклад ті, що зазвичай називають xUnit, та комерційні рішення, такі як TBrun, Testwell CTa++ and VectorCaST/C++.

Також можливо виконувати модульне тестування без додаткових бібліотек, створюючи код, що застосовує механізми assertion, виняткових ситуацій англ. *exception*, та інші механізми контролю виконання програми, які можуть сигналізувати про неуспіх.

Тестування з Jest

Jest є найбільш широко використовуваним блоком тестування JavaScript.

Що таке тест знімка, і чому це корисно? Тести для знімків є дуже корисним інструментом, коли ви хочете переконатися, що ваш інтерфейс не змінюється несподівано.

Типовий тест зйомки для мобільної програми надає компонент користувальницького інтерфейсу, робить знімок, а потім порівнює його з еталонним файлом знімка, що зберігається поряд з тестом. Тест буде невдалим, якщо два знімки не збігаються: або зміна є несподіваною, або посилання-знімок потрібно оновити до нової версії компонента інтерфейсу.

артефакт знімка має бути знятий поряд із змінами коду та переглянуто як частина процесу перегляду коду. Jest використовує порозумілу форму для створення знімків, які можна читати під час перегляду коду. На наступних тестових прогонах Jest буде просто порівнювати візуалізований вихід з попереднім знімком. Якщо вони збігаються, тест пройде. Якщо вони не збігаються, тестовий бігун знайшов помилку у вашому коді (в даному випадку, це <Link> компонент), який повинен бути виправлений, або зміна реалізації змінилася, а знімок потрібно оновити.

4.1.3 Вирішення проблеми сумісності версій

Проблема сумістності версій вирішується за допомогою системи Docker. Емблема представлена на рисунку 4.6

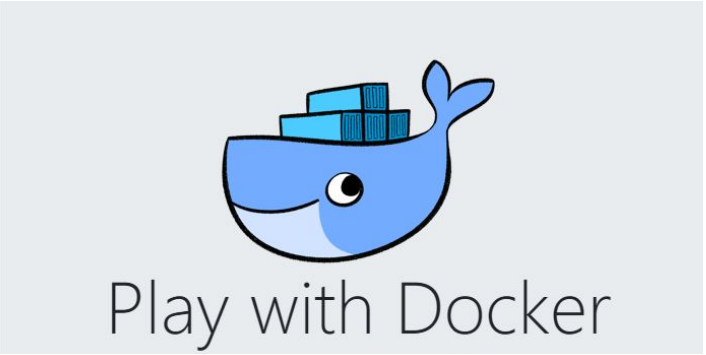

Рисунок 4.6 – Эмблема Docker

Docker — інструментарій для управління ізольованими Linux-контейнерами. Docker доповнює інструментарій LXC більш високорівневим aPI, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.

Сирцевий код Docker написаний мовою Go і поширюється під ліцензією apache 2.0. Інструментарій базується на застосуванні вбудованих в ядро Linux штатних механізмів ізоляції на основі просторів імен (namespaces) і груп управління (cgroups). Для створення контейнерів використовуються скрипти lxc. Для формування контейнера досить завантажити базовий образ оточення (команда docker pull base), після чого можна запускати в ізольованих оточеннях довільні програми (наприклад, для запуску bash можна виконати docker run -i -t base/bin/bash).

Основні можливості Docker:

- Можливість розміщення в ізольованому оточенні різнорідної начинки, що включає різні комбінації виконуваних файлів, бібліотек, файлів конфігурації, скриптів, файлів jar, gem, tar тощо
- Підтримка роботи на будь-якому комп'ютері на базі архітектури x86_64 з системою на базі ядра Linux, починаючи від ноутбуків,

закінчуючи серверами та віртуальними машинами. Можливість роботи поверх немодифікованих сучасних ядер Linux (без накладення патчів) і в штатних оточеннях всих великих дистрибутивів Linux, включаючи:

- Fedora;
- RHEL;
- Ubuntu;
- Debian;
- SUSE;
- Gentoo.
- Використання легкових контейнерів для ізоляції процесів від інших процесів і основної системи.
- Оскільки контейнери використовують свою власну самодостатню файлову систему, не важливо де, коли і в якому оточенні вони запускаються.
- Ізоляція на рівні файлової системи: кожен процес виконується у повністю окремій кореневій ФС;
- Ізоляція ресурсів: споживання системних ресурсів, таких як витрата пам'яті і навантаження на CPU, можуть обмежуватися окремо для кожного контейнера з використанням cgroups;
- Ізоляція на рівні мережі: кожен ізольований процес має доступ тільки до пов'язаного з контейнером мережевого простору імен, включаючи віртуальний мережевий інтерфейс і прив'язані до нього IP-адреси;
- Коренева файлова система для контейнерів створюється з використанням механізму copy-on-write[en] (окремо зберігаються тільки змінені і нові дані), що дозволяє прискорити розгортання, знижує витрату пам'яті і економить дисковий простір;
- Всі стандартні потоки (stdout/stderr) кожного виконуваного в контейнері процесу накопичуються і зберігаються у вигляді логу;
- Змінена файлова система одного контейнера може використовуватися як основа для формування нових базових образів і

створення інших контейнерів, без необхідності оформлення шаблонів або ручного налаштування складу образів;

- Можливість використання інтерактивної командної оболонки: до стандартного вводу будь-якого контейнера може бути прив'язаний псевдо-tty для запуску shell.
- Підтримка використання різних систем зберігання, які можуть підключатися як плагіни. Серед підтримуваних драйверів зберігання заявлені aufs, device mapper(використовуються снапшоты LVM), vfs (на основі копіювання директорій) і Btrfs. Очікується поява драйверів для ZFS, Gluster і Ceph;
- Можливість створення контейнерів, що містять складні програмні стеки, через зв'язування між собою вже існуючих контейнерів, що містять складові частини формованого стека. Зв'язування здійснюється не через злиття вмісту, а через забезпечення взаємодії між контейнерами (створюється мережевий тунель).

4.1.4 API

API Будемо розробляти на базі GraphQL, емблема представлена на малюнку 4.7

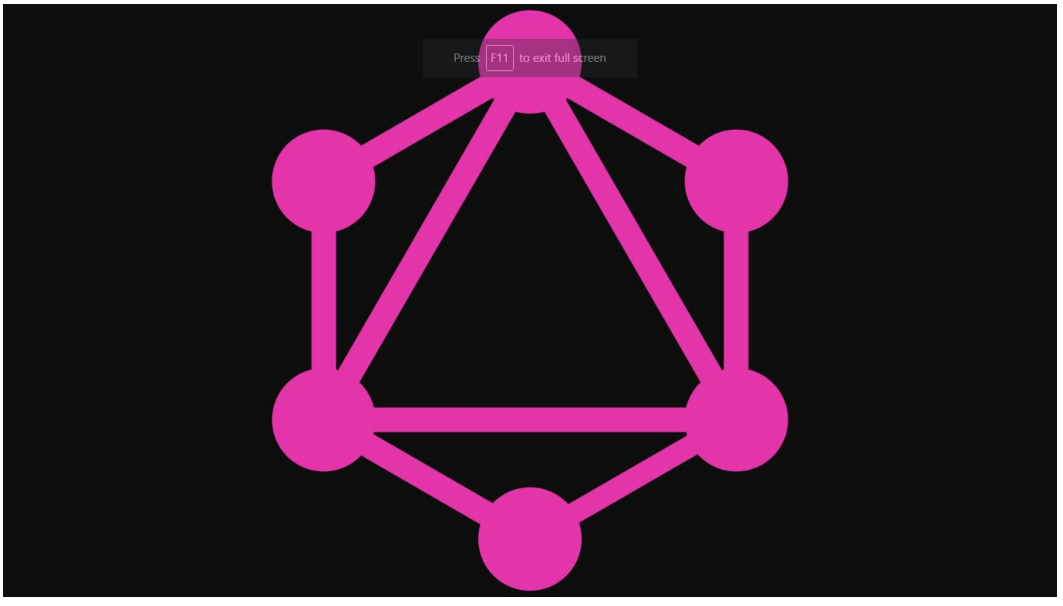

Рисунок 4.7 – Емблема GraphQL

GraphQL - це мова запитів для API і час виконання для виконання цих запитів з вашими існуючими даними. GraphQL надає повний і зрозумілий опис даних у вашому API, дає клієнтам можливість вимагати саме те, що їм потрібно, і нічого більше, що полегшує розвиток API з часом, а також надає потужні інструменти для розробників.

Надішліть запит GraphQL до вашого API і отримайте саме те, що вам потрібно, нічого більше і нічого менше. Запити GraphQL завжди повертають передбачувані результати. Програми, що використовують GraphQL, є швидкими та стабільними, оскільки контролюють отримані дані, а не сервер.

Запити GraphQL доступ не тільки до властивостей одного ресурсу, але й плавно слідують посиланням між ними. Хоча типові REST API вимагають завантаження з декількох URL, API API GraphQL отримує всі дані, які потрібні вашим програмам, в одному запиті. Програми, що використовують GraphQL, можуть бути швидкими навіть на повільних мережевих з'єднаннях.

API GraphQL організований з точки зору типів і полів, а не кінцевих точок. Доступ до повних можливостей даних з однієї кінцевої точки. GraphQL використовує типи, щоб гарантувати, що apps лише запитують, що можливо, і надає чіткі та корисні помилки. Програми можуть використовувати типи, щоб уникнути написання ручного коду розбору.

Точно знати, які дані ви можете запросити з вашого API, не виходячи з редактора, виділіть потенційні проблеми, перш ніж надсилати запит, і скористайтеся покращеним інтелектуальним кодом. GraphQL дозволяє легко створювати потужні інструменти, такі як GraphiQLby, використовуючи систему типів вашого API.

Додавання нових полів та типи до вашого API GraphQL, не впливаючи на існуючі запити. Поля старіння можуть бути застарілими та приховані від інструментів. Використовуючи одну розвивається версію, API API GraphQL надають програмам безперервний доступ до нових функцій і заохочує більш чистий, підтримуваний код сервера.

GraphQL створює єдиний API по всьому вашому додатку, не

обмежуючись конкретним механізмом зберігання даних. Напишіть GraphQL API, які використовують ваші існуючі дані та код з двигунами GraphQL, доступними на багатьох мовах. Ви надаєте функції для кожного поля в системі типу, і GraphQL викликає їх з оптимальною паралельністю.

На рисунку 4.8 можно побачити відомі бренди, які використовують GraphQL.

Рисунок 4.8 – Хто використовує GraphQL

4.1.5 Технологія розробки клієнтської частини програми

React Native - це фреймворк для мобільних додатків з відкритим вихідним кодом, створений на Facebook. Він використовується для розробки додатків для android, iOSandUWPby, що дозволяє розробникам використовуватиReactalong з власними можливостями платформи.

Створюйте власні мобільні програми за допомогою JavaScript та React

React Native дозволяє створювати мобільні програми, використовуючи лише JavaScript. Він використовує той самий дизайн, що й React, дозволяючи створювати багатий мобільний інтерфейс за допомогою декларативних компонентів.

Програма React Native - це реальне мобільне додаток

Програми, які ви створюєте за допомогою React Native, не є мобільними веб-програмами, оскільки Re-act Native використовує ті ж основні блоки інтерфейсу користувача, що й звичайні програми для iOS і android. Замість

того, щоб використовувати Swift, Kotlin або Java, ви збираєте ці блоки разом, використовуючи JavaScript і React.

Не витрачайте час на перекомпіляцію:

React Native дозволяє вам швидше створювати свої програми. Замість того, щоб перекомпілювати, ви можете миттєво перезавантажити програму. без перезавантаження. Можна навіть запустити новий код, зберігаючи стан програми. Постарайтеся - це чарівний досвід.

Використовуйте рідний код, коли це потрібно:

React Native плавно з'єднується з компонентами, написаними у Swift, Java або Objective-C. Якщо вам потрібно оптимізувати декілька аспектів програми, просто опустіть її до рідного коду. Також легко створити частину своєї програми в React Native, а частину програми - безпосередньо з власного коду - так працює додаток Facebook

Хто використовує React Native?

Тисячі програм використовують React Native, від створених компаній Fortune 500 до нових стартапів. Якщо вам цікаво побачити, що можна зробити з React Native, перевірте ці програми.

На рисунку 4.9 можно побачити відомі бренди, які використовують React Native.

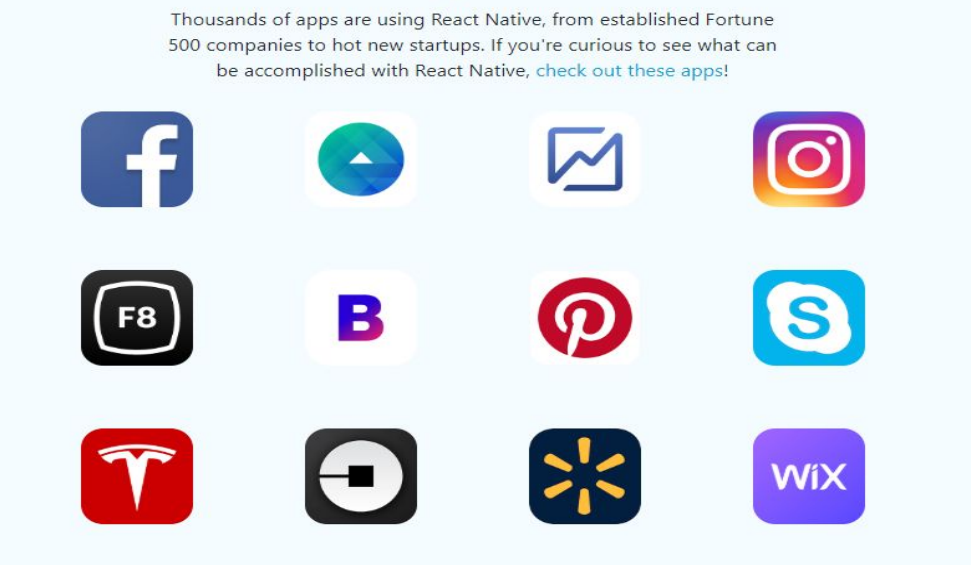

Рисунок 4.9 – Хто використовує React Native

4.1.6 Вибір середовища розробки програмного продукту

Microsoft Visual Studio — серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-службияк в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework,.NET Compact Framework та Microsoft Silverlight.

Visual Studio Code — засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

Це середовище розробки стало першим крос-платформовим продуктом у лінійці Visual Studio.

За основу для Visual Studio Code використовуються напрацювання вільного проекту atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над atom Shell, що використовують браузерний рушій Chromium і Node.js. Примітно, що про використання напрацювань вільного проекту atom на сайті Visual Studio Code і в прес-релізі і в офіційному блозі не згадується.

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку длянку платформ aSP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Логотип Visual Studio Code представлений на рисунку 4.10

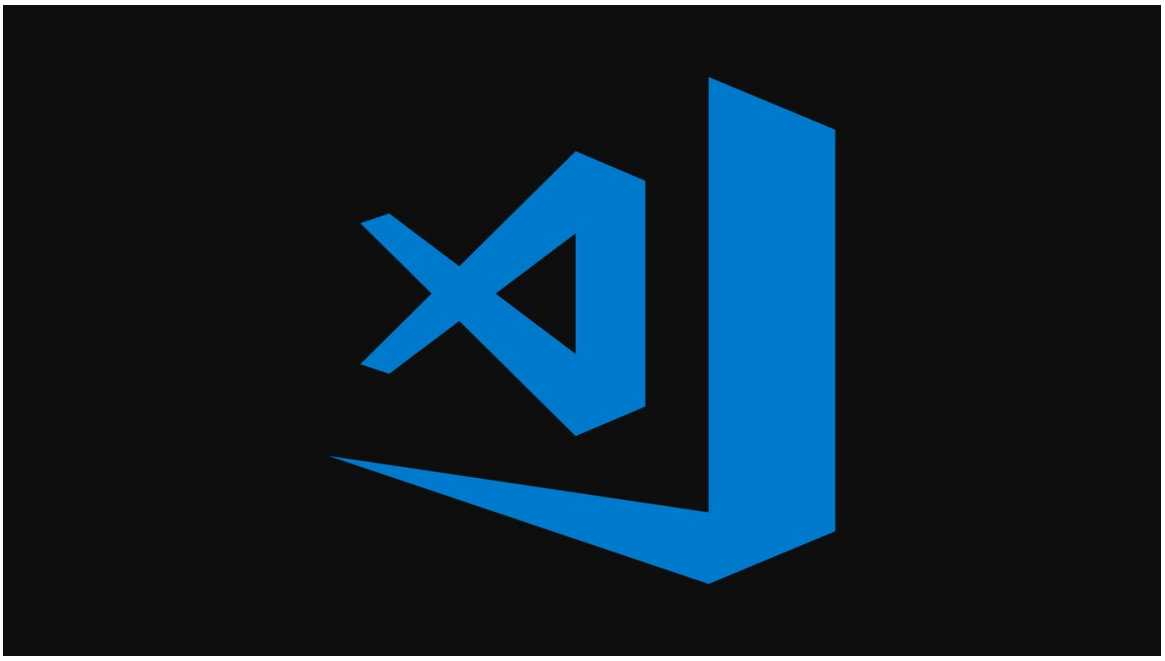

Рисунок 4.10 – Логотип Visual Studio Code

4.1.7 Побудова структури бази даних

Code First - це техніка, яка допомагає нам створювати базу даних, мігрувати і підтримувати базу даних і її таблиці з коду. З коду означає, що ви безпосередньо підтримуєте базу даних та її відповідні таблиці з коду .NET. Це корисно, коли у вас немає готової бази даних, і ви хочете почати роботу з новим свіжим проектом і хочете створити базу даних і підтримувати базу даних безпосередньо з вашого коду.

1) Менше розкрадань, менше наворотів

Використання існуючої бази даних для створення файлу моделі .edmx і пов'язаних з нею кодових моделей призводить до гігантської купи автоматично згенерованого коду. Ви благали ніколи не торкатися цих згенерованих файлів, щоб ви не зламали щось, або ваші зміни перезаписані на наступне покоління. Контекст і ініціалізатор також застрягли в цьому безладі. Коли вам потрібно додати функціональність до створених моделей, як, наприклад, обчислене властивість лише для читання, потрібно розширити клас моделі. Це закінчується тим, що вимога для майже кожної моделі, і ви в кінцевому

підсумку з розширенням для всього.

По-перше, вашою рукою стануть ваші ручні моделі. Точні файли, які ви створюєте, це те, що генерує дизайн бази даних. Немає додаткових файлів і немає необхідності створювати розширення класу, коли ви хочете додати властивості або що-небудь ще, про що база даних не потрібно знати. Ви можете просто додати їх до того ж класу, якщо ви дотримуєтеся відповідного синтаксису. Чорт, ви навіть можете створити модель. edmx файл для візуалізації коду, якщо ви хочете.

2) Більший контроль

По-перше, коли ви йдете з БД, ви знаходитесь на милості того, що генерується для ваших моделей для використання у вашому додатку. Іноді угоди про найменування небажані. Іноді відносини і асоціації не зовсім те, що ви хочете. В інших випадках неперехідні відносини з ледачим завантаженням завдають шкоди вашим відгукам API.

Незважаючи на те, що майже завжди існує рішення для проблем, пов'язаних з генерацією моделей, ви можете зіштовхнутися з цим, коли код спочатку надасть вам повний і тонкий контроль з самого початку. Ви можете керувати кожним аспектом як ваших моделей коду, так і вашого дизайну бази даних, не виходячи з вашого бізнес-об'єкта. Можна точно вказати відносини, обмеження та асоціації.

Можна одночасно встановлювати обмеження символів властивостей та розміри стовпців бази даних. Ви можете вказати, які пов’язані колекції потрібно завантажувати з нетерпінням або взагалі не серіалізувати. Коротше кажучи, ви несете відповідальність за більше речей, але ви повністю контролюєте дизайн своєї програми.

3) Контроль версій бази даних

Це великий. Бази даних версій є важкими, але з першими міграціями коду та кодами, це набагато ефективніше. Оскільки ваша схема бази даних повністю ґрунтується на моделях коду, за версією, яка контролює ваш вихідний код, ви допомагаєте версії вашої бази даних. Ви несете відповідальність за керування

ініціалізацією контексту, яка може допомогти вам зробити такі речі, як насіння фіксованих бізнес-даних. Ви також відповідальні за створення перших міграцій коду.

При першому включенні міграцій генеруються клас конфігурації та початкова міграція. Початкова міграція - це ваша поточна схема або базова лінія v1.0. З цього моменту ви будете додавати міграції, які позначені тимчасовими даними і позначені дескриптором, щоб допомогти в упорядкуванні версій. Коли ви викликаєте міграцію додатків з менеджера пакунків, буде створено новий файл міграції, що містить все, що автоматично змінилося у вашій моделі коду як у функції UP , так і DOWN ().

Функція UP застосовує зміни до бази даних, функція DOWN видаляє ті самі зміни у події, яку потрібно відкотити. Більш того, ви можете редагувати ці міграційні файли, щоб додати додаткові зміни, такі як нові види, індекси, збережені процедури та інше. Вони стануть справжньою системою версій для схеми бази даних.

Ми створюємо основні сутності:

Створення сутності юзера представлено на рисунку 4.11

```
namespace Core.Entities
{
    public class User
    {
        public int Id { get; set; }
        public string Email { get; set; }
        public string Name { get; set; }
        public string PhoneNumber { get; set; }
        public double Mileage { get; set; }
    }
}
```

Рисунок 4.11 – Сутність юзера

Створення страйдів юзера представлено на рисунку 4.12

```
using System;

namespace Core.Entities
{
    public class Stride
    {
        public int Id { get; set; }
        public DateTime Start { get; set; }
        public DateTime End { get; set; }
        public Route Route { get; set; }
        public User User { get; set; }
    }
}
```

Рисунок 4.12 - Сутність страйдів

Створення сутності рутів представлено на рисунку 4.13

```
public int Id { get; set; }
public LineString Points { get; set; }
public double Mileage { get; set; }
```

Рисунок 4.13– Сутність рутів

Для зберігання координат використовуємо бібліотеку Net TopologySuite. Geometries

4.2 Загрузка даних

Для завантаження даних напишемо для себе окремий модуль, який назвемо load.js і покладемо в папку utils. Завантаження даних буде відбуватися асинхронно після ініціалізації самого додатка, тому зручно користуватися

Промісами:

Код загрузки даних представлений на малюнку 4.14

```
export default url => {
 return new Promise((success, fail) => {
  const request = new XMLHttpRequest();
  request.open('GET', url, true);

  request.addEventListener('load', () => {
   request.status >= 200 && request.status < 400
    ? success(request.responseText)
    : fail(new Error(`Request Failed: ${request.statusText}`));
  });
```

Рисунок 4.14 – Код загрузки даних

Тепер ми можемо асинхронно завантажити будь-який файл з сервера подібним чином, це зображено на малюнку 4.15

```
load('data.json')
 .then(data => {
 });
```

Рисунок 4.15 – Представлення даних для загрузки з сервера

4.2.1 Головний компонент App та стан

На чолі всього програми стоїть великий начальник <App />, який групує всі інші компоненти і вказує їм своїм станом, коли потрібно ререндериться. Після ініціалізації програми <App /> проводить завантаження даних і оновлює вміст сторінки.

У додатку є всього три властивості (Рисунок 4.16), які необхідно використовувати в стані: дані, отримані з сервера, номер активного користувача

та пошуковий запит, введений в рядок пошуку.

```
this.state = {
  data: null,
  term: '',
  active: 0
};
```

Рисунок 4.16 – Основні властивості додатку

Початкові налаштування для стану будуть виглядати наступним чином: даних немає, пошукового запиту теж, активний користувач під номером 0

4.2.2 Окремий метод для загрузки

Для завантаження даних найпростіше створити окремий метод. Назвемо його умовно loadData. Використовуючи компонент, ми можемо вказати його властивості, передаючи "атрибути". В даному випадку буде дуже зручно вказати подібний параметр для компонента <App /> для позначення того, який файл необхідно завантажити. Мі можемо побачити код завантаження даних на рисунку 4.17.

```
 load(this.props.data).then(users => {
    this.setState({
      data: JSON.parse(users)
    });
  });
```

Рисунок 4.17 – Завантаження даних

4.2.3 Спілкування з іншими компонентами

Щоб дати можливість дочірнім компонентів постійно оновлювався,

створимо ще один метод, який назвемо updateData. Метод буде приймати об'єкт і просто встановлювати його в якості поточного стану, це зображено на малюнку 4.18.

```
updateData(config) {
  this.setState(config);
}
```

Рисунок 4.18 – Компонент для приймання даних

Пізніше при використанні методу render ми зможемо передати цю функцію в якості параметра іншого компонента, і він зможе оновити стан батьківського компонента, код цього представлений на рисунку 4.19.

```
render() {
  return (<div update={this.updateData.bind(this)}></div>)
}
```

Рисунок 4.19 – Оновлювач батьківського компоненту

4.2.4 Відображення

Дані на сторінці відображаються в двох областях: перша область відповідає безпосередньо за відображення всіх існуючих користувачів (виділимо для неї компонент <UserList />), друга область відображає поточного користувача, якого можна вибрати при натисканні з <UserList /> (назвемо <ActiveUser />).

Очевидно, що для простого відображення даних немає необхідності створювати компоненти зі станом. Тому для такої простої операції скористаємося функціональним компонентом (functional stateless component).

Отже, у нас є масив даних, які слід відобразити на сторінці у вигляді

таблиці. Нам знадобиться трохи шаблонізації, з якої відмінно впорається React. Складемо для себе шаблон, по якому будуть відображатися кожен користувач.

4.2.5 Шаблонізація

Ми вже з'ясували, що для компонента <UserList /> немає необхідності в стані, тому, як і в прикладі вище, ми можемо обійтися функціональним варіантом. На ряду з "динамічними" даними, завантаженими з сервера, компонент містить статичну частину - заголовки для таблиць. Тому відразу можемо намітити для себе базову розмітку:

Дані є, компонент для шаблонізації даних є, залишилося тільки придумати спосіб, як все з наших даних (звичайних об'єктів) зробити компонент. Все просто. Пройдемося по масиву даних за допомогою map і створимо новий масив компонентів (рисунок 4.20).

```
const users = data.map((user, index) => {
  return (<UserData user={user} index={index} update={update} />);
});
```

Рисунок 4.20 – Створення масиву компонентів

І це все, що від нас вимагається. Майже. Залишається тільки подумати про те, що буде відбуватися, якщо ми не отримали дані. У поточному стані ми отримаємо помилку, data буде містити null і, відповідно, не вийде використовувати map.

Щоб не потрапити в таку неприємну ситуацію, перше, що нам необхідно зробити - перевірити, чи були отримані дані (Рисунок 4.21).

```
if (!data) { return (<p>Loading...</p>); }
```

Рисунок 4.21 – Перевірка приймання даних

4.2.6 Пошук

Шукати користувачів будемо по імені. Для реалізації пошуку створимо компонент <SearchBar />. Для цього в компоненті створимо поле введення, яке будемо відстежувати на подію change.

При введенні або видаленні тексту подія буде негайно спрацьовувати і відсівати або додавати результати в пошук. Компонент знову буде написаний в функціональному стилі, як на Рисунку 4.22.

```
export default ({ term, data, update }) => {
 const dataSearch = e => {};
 return (
  <div className="searchbar form-group">
    <input
      value={term}
      type="text"
      className="form-control"
      placeholder="Search people by name..."
      onChange={dataSearch}
    />
  </div>
 );
```

Рисунок 4.22 – Створення компоненту для пошуку даних

4.2.7 Сортування

Сортування даних є найбільш складним завданням, тому тут ми не можемо обійтися функціональним компонентом. Сортування повинна працювати з поточними даними, тобто тими, які знаходяться в стані компонента <App />. Метод масивів sort, з яким ми будемо працювати має не

саму приємну особливість - він сортує вихідний масив.

При роботі зі станом діє правило: "змінювати будь-які дані можна тільки за допомогою функції this.setState". Тому при реалізації механізму сортування кожен раз нам доведеться створювати новий масив. Також для визначення того, в якому порядку варто сортувати дані, ми створили властивість sorted з об'єктом, що вказує на черговість.

Тепер ми повністю готові написати метод sort (Рисунок 4.23):

```
sort (type) {
  // за допомогою реструктуризації створюємо дві змінні
  const {update, data} = this.props;
  // отримуємо порядок сортування
  const isSorted = this.sorted [type];
  // встановлюємо напрямок
  let direction = isSorted? 1: 1;

  // створюємо новий масив з даних, щоб не перезаписувати
  // стан і сортуємо його
  const sorted = [] .slice.call (data) .sort ((a, b) => {
    // щоб сортування завжди була однаковою врахуємо всі умови
    // функція може повернути 0, 1 або -1, в залежності від що повертається
    // значення метод масивів sort зробить свій вибір
    if (a [type] === b [type]) {return 0; }
    return a [type]> b [type]? direction: direction * -1;
  });

  // міняємо порядок сортування
  this.sorted [type] =! isSorted;

  // оновлюємо стан
```

```
  update ({
    data: sorted,
    active: 0
  });
}
```

Рисунок 4.23 – Код для сортування даних

Методом sort будуть користуватися перші дві кнопки тулбара, для третьої кнопки необхідно написати ще один метод reset, який відновить початковий стан програми (Рисунок 4.24):

```
reset() {
  this.props.update({
    data: this.props.initialData,
    term: '',
    active: 0
  });
}
```

Рисунок 4.24 – Код для відновлення стану програми

4.3 Стартове меню

Для користувача на початку відкривається меню в якому він може вибрати створити нову пробіжку або подивитися інформацію про минулі пробіжки, це можна побачити на рисунку 4.25

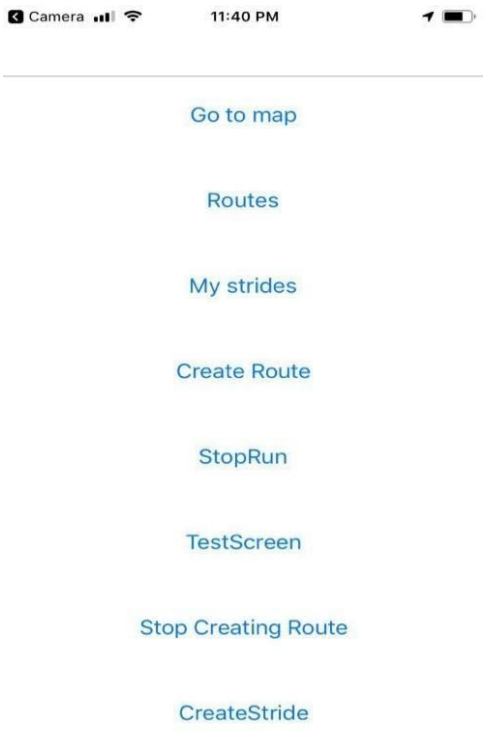

Рисунок 4.25 – Меню

4.3.1 Створення пробіжки

При натисканні на клавішу Create Route відкривається карта на якій при натисканні клавіші go починається пробіжка, рисунок 4.26

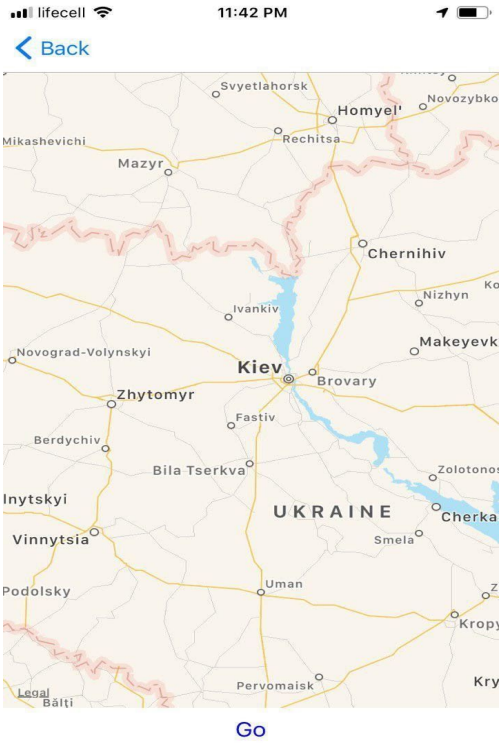

Рисунок 4.26 - Карта перед пробіжкою

4.3.2 Пробіжка

При натисканні на клавішу Go починається запис пробіжки, кнопка Go змінюється на кнопку Stop , це можна побачити на рисуноку 4.27

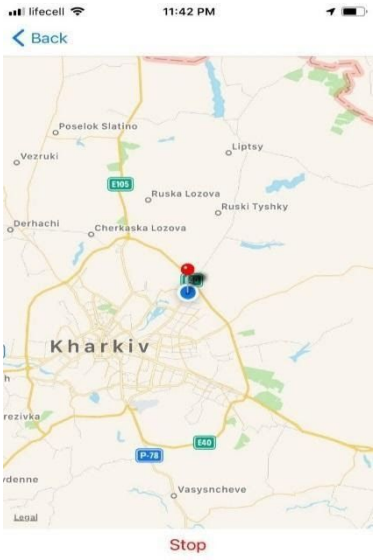

Рисунок 4.27 - Процес бігу

Збільшуючи масштаб карти ми чітко бачимо малюнок нашого маршруту (Рисунок 4.28)

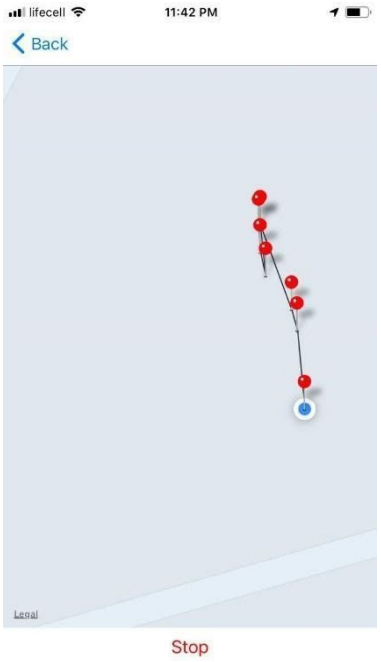

Рисунок 4.28 – Маршрут в приближенному форматі

4.3.3 Підтвердження завершення пробіжки

Після натискання кнопки Stop з'являється скрін на якому можна підтвердити зупинку пробіжки і тим самим зберегти її в базі даних, при натисканні клавіші Decline ви продовжите вашу поточну пробіжку (Рисунок 4.29)

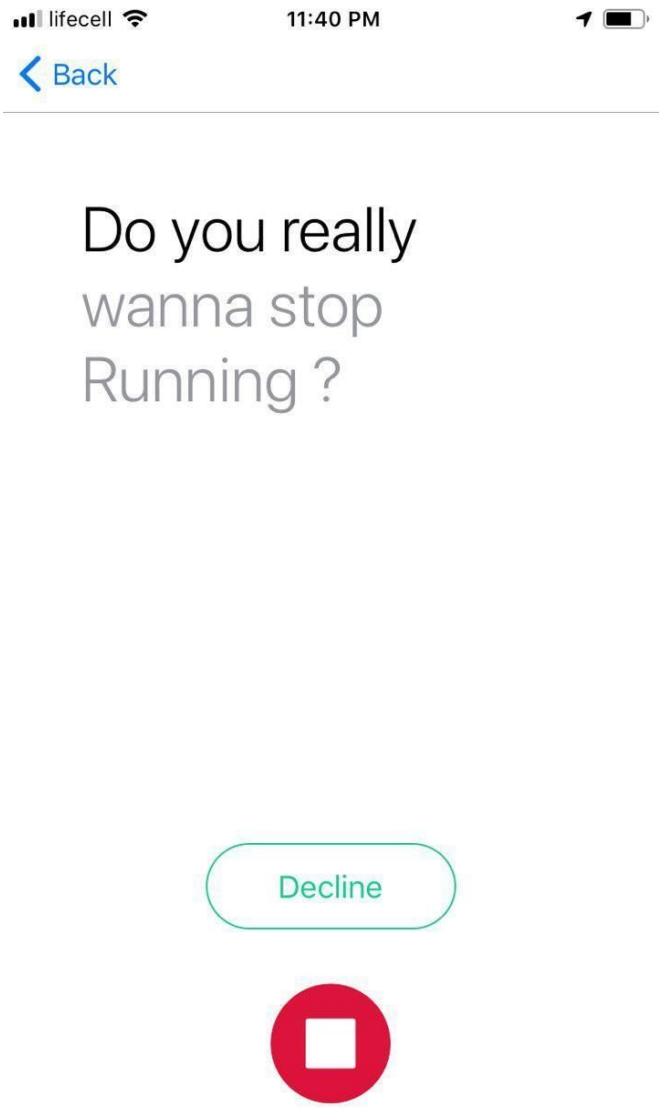

Рисунок 4.29 – Підтвердження завершення пробіжки

4.3.4 Завершення пробіжки

При зупинці випливає вітальне повідомлення з результатами пробіжки (Рисунок 4.30)

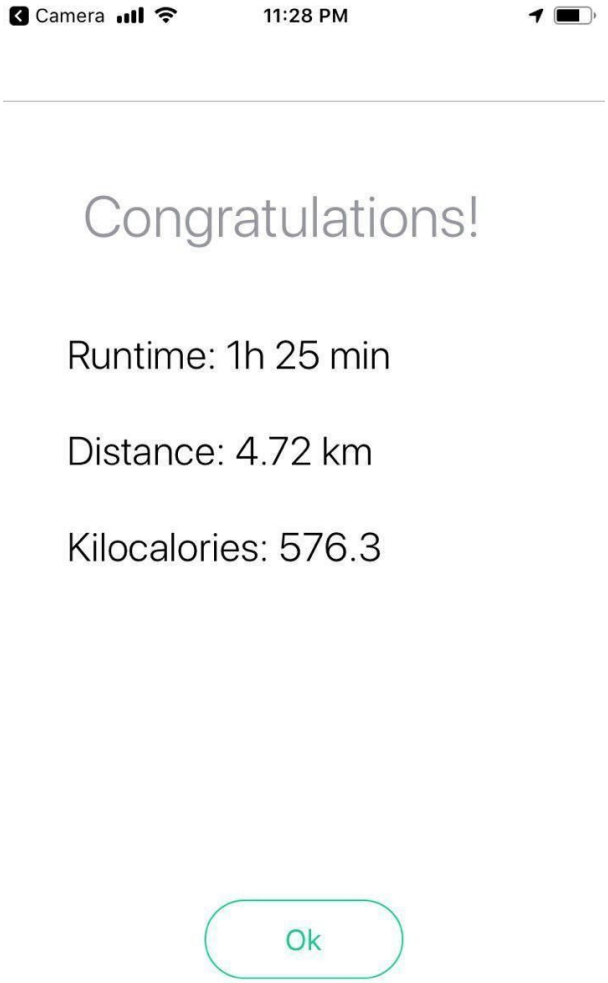

Рисунок 4.30 – Завершення пробіжки

4.4.5 Історія пробіжок

При натисканні на My Strides можна подивитися останні пробіжки. Це показано на рисунку 4.31

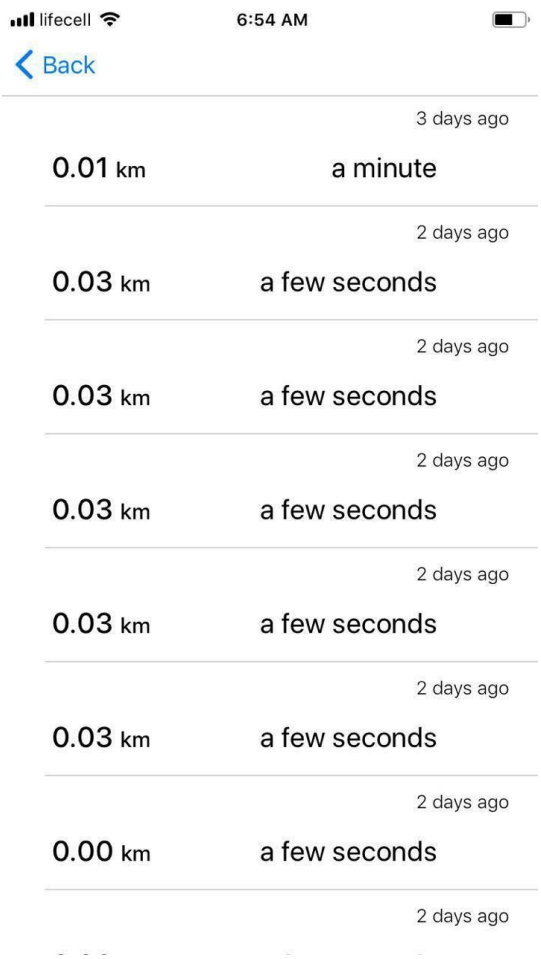

Рисунок 4.31 - Історія пробіжок

5 ЕКОНОМІЧНА ЧАСТИНА

Мета економічної частини дипломного проекту - розрахувати витрати на розробку програмного забезпечення та визначити економічну ефективність від його впровадження.

Програмний продукт (ПП) - поряд з апаратними засобами, найважливіша складова інформаційних технологій, що включає комп'ютерні програми і дані, призначені для вирішення певного круга задач і зберігаються на машинних носіях.

Програмний продукт являє собою або дані для використання в інших програмах, або алгоритм, реалізований у вигляді послідовності інструкцій для процесора.

Метою створення програмного продукту є отримання необхідного і достатнього системного комплексу якісних програмних виробів за умови реалізації ефективного процесу розробки і супроводу.

5.1 Опис програмного продукту

У процесі роботи було створено програмний продукт для створення маршрутів для пробіжок. Цей додаток має такі можливості:

1. Додавання своїх маршрутів для пробіжок.
2. Можливість подивитися історію своїх пробіжок.
3. Можливість подивитися дистанцію та час пробіжки.
4. Визначити траєкторію до ближайшей пробіжки
5. Відсортувати список пробіжок

5.2 Розрахунок собівартості програмного продукту

Щоб оцінити вартість розробленого програмного продукту необхідно:

1. Скласти перелік робіт, які слід виконати.
2. Розрахувати трудовитрати на їх виконання.

3. Розрахувати заробітну плату розробників.

У витрати на розробку програмного продукту також входять: вартість малоцінних і швидкозношуваних предметів, вартість покупки комп'ютера, відрахування з заробітної плати і т.д.

До переліку робіт, які необхідно виконати входить:

1. Формулювання постановки задачі.

2. аналіз вимог, проектування функціональних і не функціональних можливостей програмного продукту.

3. Розробка програмного продукту.

5.3 Виконавці роботи

Для управління ходом робіт і ведення всього проекту в цілому необхідна посада TeamLead.

Для налаштування хостингу та створення контейнерів потрібна посада Devops.

Для розробки бібліотеки ядра, створення бібліотеки інфраструктури, розробки GraphQl API потрібен Software Engineer.

Для створення Back-end архітектури потрібен Software architect.

Для створення UI – дизайну потрібен UI Designer.

Для розробки додатку потрібен Front-end Engineer.

Для визначення правильного функціонування програми необхідний QA Engineer.

Для планування та управління потрібен Project Manager.

Склад групи для розробки програмного продукту наведено в таблиці 5.1.

Таблиця 5.1 - Склад виконавців роботи

посади	Посадові оклади, грн	
	місячні	Денні

Продовження таблиці 5.1

Devops	28 000	3072,72
Software Engineer	90 000	1 272,72
Software architector	70 000	3181,81
UI Designer	33 000	1500
Front-end Engineer	58 000	2636,36
QA Engineer	25 000	1136,36
Project Manager	80 000	3636,36

При нарахуванні заробітної плати за місяць приймаємо 22 рабочих дня.

5.4 Перелік робіт для створення програмного продукту

Наведемо переліки робіт для розробників програмного продукту.

Заробітна плата (ЗП) - це виражена в грошовій формі частина доходів, що надходять в особисте споживання працівників відповідно до кількості і якості витраченого ними праці.

Перелік робіт, що виконуються співробітниками і їх тривалість приведена в таблиці 5.2.

Таблиця 5.2 - Перелік робіт для створення програмного продукту

Task	Team member	Est Time
Setting up hosting	Devops	8h
Creating container	Devops	40h
Developing core library	Software engineer	24h
Developing infrastructure library	Software engineer	32h
Developing GraphQL API	Software engineer	24h

Продовження таблиці 5.2

Back-end architecture Design	Software architector	32h
Frontend architecture Design	Software architector	24h
UI Designs	UI Designer	24h
application development	Front-end Engineer	80h
Testing UX Design	QA Engineer	64h
Planning, magment con-versation with customer	Project Manager	64h

Розрахунок собівартості робіт почнемо з розрахунку фонду основної заробітної плати (ОЗП) розробників, з урахуванням трудовитрат, кількості виконавців і середньодобової заробітної плати. Фонд основної заробітної плати визначається за формулою:

$$З_{ос} = З_{devop} * Т_{devop} + З_{se} * Т_{se} + З_{sa} * Т_{sa} + З_{ui\ des} * Т_{ui\ des} + З_{fe} * Т_{fe} + + З_{qa\ eng} * Т_{qa\ eng} + \quad (5.1)$$

де - трудомісткість роботи $Т_{devop}$, $Т_{se}$, $Т_{sa}$, $Т_{ui\ des}$, $Т_{fe}$, $Т_{qa\ eng}$, $Т_{pm}$ Devops,

Software Engineer, Software architector, UI Designer, Front-end Engineer, QA Engineer, Project Manager.

$З_{devop}$, $З_{se}$, $З_{sa}$, $З_{ui\ des}$, $З_{fe}$, $З_{qa\ eng}$, $З_{pm}$ - заробітні плати Software Engineer, Software architecture, UI Designer, Front-end Engineer, QA Engineer, Project Manager відносно.

Звідси:

$З_{ос}$ =3072,72 * 6 + 1272,72 * 10 + 3181,81 * 7 + 1500 * 3 + 2636,36 * 10 +

1136,36 * 8 + 3636,38 * 8 = 122 455,18 грн.

Нехай додаткова заробітна плата (ДЗП) обумовлена у розмірі 20% ОЗП.

Додаткова заробітна плата (ДЗП) - включає в себе компенсацію за час відпустки, оплату за перебування в декретній відпустці, лікарняні виплати, виплати за роботу неповнолітніх підлітків, за виконання громадських чи державних робіт, виплата допомоги при звільненні та інші виплати.

$$З_{доп} = З_{ос} * \frac{Н_{доп}}{100} \quad (5.2)$$

де $З_{ос}$ - Основна заробітна плата; $Н_{доп}$ - Відсоток відрахувань до додаткової заробітної плати основних виробничих робочих (20%).

$$З_{доп} = 95727,18 * 0.2 = 24491,44 \text{ грн.}$$

Як наслідок можемо обчислити фонд заробітної плати:

$$Ф_{зп} = З_{ос} + З_{доп},$$

(5.3)

де $З_{ос}$ - основна заробітна плата; $З_{доп}$ - додаткова заробітна плата.

Разом:

$$Ф_{зп} = З_{доп} + З_{ос} = 122\,455,18 + 24\,491,44 = 146946,36 \text{ грн.}$$

Нарахування на заробітну плату у відсотках від основної і додаткової заробітних плат (ЄСВ - єдиний соціальний внесок) становить 22%

$$З_{соц} = Ф_{зп} * \frac{Н_{соц}}{100}, \quad (5.4)$$

де $Ф_{зп}$ - Фонд заробітної плати; $Н_{соц}$ - відсоток відрахувань до фонду соціального захисту населення (22%).

Звідси:

$$З_{соц} = 146946,36 * \frac{22}{100} = 32328,18 \text{ грн.}$$

5.5 Розрахунок кошторису та ціни на розробку програми

Розрахуемо витрати на матеріали та комплектуюче, необхідне для написання программи. Результати занесені в таблицю 5.3.

Загальна сума витрат на матеріали складає - 911,00 грн.

Послуги інтернету за 52 дні: 500 грн.

Для Виконання робіт, пов'язаних з проектування програмного продукту та налаштування, потрібні ПК (оперативна пам'ять НЕ менше 32 Гб, дискова пам'ять НЕ менше 1 Гб) вартістю 47000,00 грн та Ліцензійна операційна система Windows 10 - 3000,00 грн.

Таблиця 5.3 - Витрати на матеріали

№ п / п	матеріал	призначення	Кількість, шт	Ціна за одиниць, грн	Сума, грн
1	Папір формату а4 (упаковка 500 аркушів)	Оформлення документації та відгуків	2	100,50	201,00
2	Картридж для принтера	Друк документації	1	460,00	460,00
3	Флеш - карта (8 Гб)	Збереження ПО	1	250,00	250,00
				СУМа:	911,00

Вартість основних складових приведена в таблиці 5.4.

Таблиця 5.4 - Вартість основних складових

№ п / п	Найменування	Кількість, шт	Ціна за одиниць, грн	Сума, грн
1	Персональний комп'ютер з ліцензійним Windows 10	7	50000,00	350000
			СУМа:	350000

Визначимо витрачений машинний час:

$$Т_{мв} = ВИ * Т_{devop} + ВИ_{se} * Т_{se} + ВИ_{sa} * Т_{sa} + ВИ_{ui\ des} * Т_{ui\ des} + ВИ_{fe} * * Т_{fe} + + \ ВИ_{qa}$$

(5.5)

де T_{devop}, T_{se}, T_{sa}, $T_{ui\ des}$, T_{fe}, $T_{qa\ eng}$, T_{pm} - трудомісткість роботи керівника, програміста і тестувальника;

$ВИ_{devop}$, $ВИ_{se}$, $ВИ_{sa}$, $ВИ_{ui\ des}$, $ВИ_{fe}$, $ВИ_{qa\ eng}$, $ВИ_{pm}$ - час роботи Devops,Software Engineer, Software architecture, UI Designer, Front-end Engineer, Qa Engineer, Project Manager за комп'ютером в середньому за день відповідно.

Отримаємо:

$Т_{мв}$ =8*6+8*10+8*7+8*3+8*10+8*8+8*8= 416 години.

Вартість години машинного часу ЧМЧ будемо вважати рівною 5 грн.

$$С_{мв} = Т_{мв} * Ч_{мв}, \quad (5.6),$$

де $Т_{мв}$ - витрачений машинний час; $Ч_{мв}$ - вартість години машинного часу (5 грн.).

Отримаємо.

$$С_{мв} = 416 * 5 = 2\ 080 \text{ грн.}$$

Амортизаційні відрахування (аМО) - частина вартості основних засобів, що входять у вартість готової продукції.

Річна норма амортизаційних відрахувань (аМО) розраховується як 25% від вартості одного комп'ютера та його комплектуючих.

$$\text{аМО} = 350000,00 * 0,25 = 87\,500,00 \text{ грн}$$

Норма амортизаційних відрахувань на період роботи проекту становить 52 робочих дні, та розраховується за такою формулою:

$$\text{АМО}_{\text{на раб. период}} = \frac{\text{аМО}}{264} * 52 \qquad (5.7)$$

$$\text{АМО}_{\text{на раб. период}} = \frac{88\,030,75}{264} * 52 = 17\,234,\ 84 \text{ грн}$$

$$З_{оп} = З_{ос} * \frac{Н_{оп}}{100}, \qquad (5.8)$$

де $З_{ос}$ - основна заробітна плата; $Н_{оп}$ - відсоток відрахувань на освоєння виробництва.

Звідси:

$$З_{оп} = 122\,455,18 * \frac{10}{100} = 12\,245,52 \text{ грн.}$$

Собівартість - це вартісна оцінка використаних в процесі виробництва продукції (робіт, послуг) природних ресурсів, сировини, матеріалів, основних фондів, трудових ресурсів та інших витрат на її виробництво і реалізацію.

Собівартість розробки програмного продукту дорівнює сумі всіх вищезазначених витрат:

$$С_р = З_м + Ф_{зп} + З_{соц} + З_{оп} + С_{мв} + \text{аМО}_{\text{на раб. период}} + I, \qquad (5.9)$$

де $З_м$ - вартість витрат на матеріали; $Ф_{зп}$ - Фонд заробітної плати; $З_{соц}$ -

єдиний соціальний фонд; $З_{оп}$ - витрати на освоєння виробництва; $С_{мв}$ - вартість електроенергії; $аМО_{на\ раб.\ период}$ - амортизаційні відрахування. І - Інтернет

Звідсі:

$$С_р = 911 + 146946,36 + 32328,18 + 12\ 245 + 2\ 080 + 17\ 234,\ 84 + 500 = \\ = 212\ 245,38 \text{ грн.}$$

Калькуляційні статті на розробку програмного продукту представлені в таблиці 5.5.

Таблиця 5.5 – Статті калькуляції на розробку програмного продукту

№, п / п	Стаття калькуляції	Затрати, грн	Примітка
1	Матеріали	911,00	Таблиця 5.3
2	Основна заробітна плата	122 455, 18	$З_{ос}$
3	Додаткова заробітна плата	44 769, 83	20% від п.2
4	Фонд З.П.	146946, 36	$Ф_{зп} = З_{ос} + З_{доп}$
5	ЄСВ - єдиний соціальний внесок	32328, 18	22% від п.4
6	Інтернет	500	І
7	Вартість основних складових	350 000	Таблиця 5.4
8	Використання електроенергії	2 080	$Т_{мв}$
9	амортизаційні відрахування на робочий період	17 234, 84	$аМО_{на\ раб.\ період} = \frac{З_{ос} * 0,25}{264} * 29$
10	Собівартість розробки	212 245, 38	Сумма п.1 п.8 п.4

Економічний висновок

За допомогою економічної частини ми розрахували вартості основних економічних компонентів та розрахували повну собівартість програмного продукту, яка дорівнює $212\,245,38$ грн.

ВИСНОВКИ

В даній роботі аналізується процес створення фільтру для мобільного додатку «Фітнес асистент».

Актуальність задачі можливостей створення мобільного застосування фільтру для моб. додатку «Фітнес-асистент» визначається тим, що розробка такої системи робить фактично зручним пошук найвигіднішого маршруту для будь-якого користувача. а це впливає на такі важливі чинники як:

1. Надання безпеки користувачам мобільного додатку.

2. Скорочення часу користувача, який витрачається на шлях.

3. Надання користувачеві більш точної інформації про маршрут.

4. Надання можливості користувачеві вибирати маршрут, спираючись на відгуки та фото інших користувачів додатку.

Все це полегшує життя людей і дозволяє їм спокійно, швидко пересуватися по певним місцям.

Були визначені особливості мобільного додатку:

- розрахований на будь-якого користувача
- дозволяє самостійно за допомогою фільтру обирати найбільш відповідний маршрут;
- дозволить скоротити час пошуку свого маршруту;
- дозволить підвищити ефективність додатку;

–

аналіз в роботі дозволив зробити основні висновки, до яких можна віднести такі:

актуальність роботи, яка в даний час є дуже важливою частиною на ринку праці;

дерево проблем, містить такі проблеми:

Не сформульовані основні вимоги та рекомендації;

Не визначені вхідні та вихідні параметри;

Не проаналізовані вхідні та вихідні параметри

дерево цілей, посилаючись на дерево проблем, може бути представлено слідуючими під цілями:

Сформулювати основні вимоги та рекомендації для розробки;

Визначити вхідні та вихідні параметри;

Проаналізувати вхідні та вихідні параметри;

Система складається з трьох рівнів, для кожного з яких визначені функції та параметри;

Знайдені середнє геометричне число властивостей і зв'язків на один елемент, які приблизно дорівнюють 1,450 та 1,643 відповідно;

Проведена класифікація системи, а також наступні описи:

морфологічний,який описує структуру системи, а також властивості кожного елемента системи;

функціональний, в основі якого лежать складові частини системи, їх функції, вхідні та вихідні дані;

інформаційний, який надає аналіз властивостей і зв'язків кожного елемента системи.

Програмний продукт був створений, використовуючи всі необхідні етапи розробки: постановку задач, проектування системи, кодування, тестування і налагодження. Особлива увага приділялася проектування системи, розробці специфікацій вимог і математичної моделі, як до найголовніших етапів, від яких залежало якість майбутньої системи.

Під час виконання данного завдання були закріплені і застосовані на практиці навички, отримані під час роботи. Були використані навички проектування інформаційних систем, закріплено знання з технологій програмування мобільних додатків. Доповнення розроблено за допомогою технологій, typescript та Clean архітектури. Робота з базою даних SQL здійснювалася за допомогою класів налаштувань, в яких реалізована основна логіка роботи з базою даних. Проект дав можливість відчути особливості етапів

розробки, ознайомитися з системами контролю версій і отримати досвід рівномірного і раціонального розподілу часу.

За допомогою економічної частини ми розрахували вартості основних єкономічних компонентів та розрахували повну собівартість програмного продукту, яка дорівнює $212\,245,38$ грн.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Блох И.И. Разработка мобильного приложения туриста на основе алгоритмов подбора маршрутов по параметрам - 2011. - выпускная работа - 45 с.
2. Голощапов а.Л. Google android: программирование для мобильных устройств.- СПб.: БХВ-Петербург, 2010.- 448 с.
3. Mark L.Murphy Beginning android 2.- 417 с.
4. Sayed Y. Hashimi, Satya Komatineni, Dave MacLean Pro android 2. - 737 с.
5. android Design Patterns: Interaction Design Solutions for Developers, 2017 – 657 с.
6. Алгоритмы: построение и анализ / Т.Х. Кормен [и др.]. – 2-е изд. – М. : Издательский дом «Вильямс», 2007. – С. 1296.
7. Вопросы и ответы // aPI Яндекс.Карты [Електронний ресурс]: портал - режим доступу вільний: http://api.yandex.ru/maps/faq.xml.
8. McDonough W. The Next Industrial Revolution // atlantic Monthly [Електронний ресурс]: портал - режим доступу вільний: http://www.theatlantic.com/magazine/archive/1998/10/the-next-industrial-revolution /304695/
9. RU:Map Features // Open Street Map: Wiki [Електронний ресурс]: портал - режим доступу вільний: http://wiki.openstreetmap.org/wiki/RU
10. Sahni S. P-Complete approximation Problems / S. Sahni, T. Gonzalez // Journal ol the association for Computing Machinery. – № 3. - Vol. 23. - P. 555 – 565.
11. Рассел С., Норвиг П. Искусственный интеллект: современный подход, 2-е изд. Пер. с англ. М.: Издательский дом «Вильямс», 2006. 1408 с.
12. Dijkstra E.W. a note on two problems in connexion with graphs // Numer. Math — Springer Science+Business Media, 1959. Vol. 1. Iss. 1. P. 269-271.

13. Hart P.E., Nilsson N.J., Raphael B.a. Formal Basis for the Heuristic Determination of Minimum Cost Paths // IEEE Transactions on Systems Science and Cybernetics SSC4, 1968. № 2. С. 100-107.

14. Дергачев а. М. Проблемы эффективного использования сетевых сервисов / Научно-технический вестник СПбГУ ИТМО. 2011. № 1 (71). С. 83–87

15. Dover D., Dafforn E. Search Engine Optimization Secrets. Indianapolis: Wiley Publishing, Inc., 2011. 456 p.

16. Ouzzani, M., Bouguettaya a. Semantic Web Services for Web Databases. Springer Science+Business Media, 2011. 155 p.

17. Бунак В. В. Web-сервисы. Практический курс. — М., 1941 – 205 с.

18. Тегако Л. И., Марфина О. В. Практическая Web-сервисы. — Ростов-на-Дону, 2003 – 350 с.

19. Иван Репичев. Основные элементы Web-сервисов [Электронный ресурс]: портал – режим доступа свободный: http://fb.ru/article/114288/osnovnyie-e- .

20. Web-сервисы. [Электронный ресурс]: портал – Режим доступа вільний: https://dic.academic.ru/dic.nsf/ruwiki/1279850.

21. Журнал «Технологии программирования». Приложение, Таск-менеджер [Электронный ресурс]: портал – режим доступа свободный: https://www.html.com/c.

22. Єрмолаєва В. В., Калашников Д. а. автоматизовані системи управління // Молодий вчений. - 2016. - №11. - С. 166-168.

23. Організація баз даних та знань / О. В. Тарасов, Л. а. Павленко, М. Ю. Лосев, В. В. Федько – Х.: Вид. ХНЕУ, 2013. – 83 с.