Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ЗАСТОСУНКУ З ВИКОРИСТЯННЯМ FLUTTER ДЛЯ ПЕРЕГЛЯДУ ОСНОВНОЇ ІНФОРМАЦІЇ ПРО ГЕРОЇВ ГРИ DOTA 2
(тема)

Виконав:
здобувач 4 року навчання,
групи ІТІНФ-21-3
Носкін А.В.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Руденко Д.О.
(посада, прізвище, ініціали)

Допускається до захисту

Завідувач кафедри інформатики _____ Кобилін О. А.
(підпис) (прізвище, ініціали)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту

Кафедра Інформатики

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Носкіну Арсенію В'ячеславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка застосунку з використанням Flutter для перегляду основної інформації про героїв гри Dota 2

затверджена наказом університету від 19 травня 2025 року № 381Ст
2. Термін подання здобувачем роботи до екзаменаційної комісії 01 червня 2025 р.
3. Вихідні дані до роботи науково-методична та науково-технічна література з розробки мобільних застосунків, матеріали конференцій, технічна документація з Flutter, Dart та архітектури BLoC, офіційна документація OpenDota API, специфікації форматів обміну даними, ресурси інтернет-мережі, матеріали спільноти розробників, офіційна документація SQLite та бібліотек для роботи з HTTP-запитами у Flutter, а також вихідні дані з відкритих джерел, що містять інформацію про героїв гри Dota 2.

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області комп'ютерних ігор на прикладі Dota 2 та визначення інформаційних потреб користувачів.
2. Вибір інструментів та технологій для розробки кросплатформного мобільного застосунку (Flutter, Dart, архітектура BLoC).
3. Розробка структури даних для зберігання інформації про героїв гри Dota 2 та отримання її з зовнішнього джерела (OpenDota API).

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Аналіз предметної області, постановка задачі, вибір інструментів та технологій для розробки, програмна реалізація.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	07.04.2025	
2	Аналіз завдання, підбір літератури	08.04.25-10.04.25	
3	Аналіз літератури з досліджуваної проблеми	11.04.25-14.04.25	
4	Аналіз технічних засобів	15.04.25-20.04.25	
5	Розробка методу	21.04.25-27.04.25	
6	Програмна реалізація	28.04.25-25.05.25	
7	Оформлення пояснювальної записки	12.05.25-26.05.25	
8	Перевірка на нормоконтроль	21.05.25-01.06.25	
9	Перевірка на плагіат	21.05.25-01.06.25	
10	Рецензування	21.05.25-01.06.25	
11	Підготовка презентації та доповіді	21.05.25-18.06.25	
12	Занесення роботи в електронний архів	02.06.25-18.06.25	
13	Попередній захист кваліфікаційної роботи	02.06.25-18.06.25	

Дата видачі завдання 7 квітня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ _____ доц. Руденко Д.О._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 70 с., 1 табл., 27 рис., 30 джерел.

МОБІЛЬНА РОЗРОБКА, FLUTTER, DART, API, DOTA 2, JSON, КРОСПЛАТФОРМНІ ЗАСТОСУНКИ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

Об'єктом роботи є мобільний застосунок для перегляду ігрової інформації про персонажів Dota 2.

Метою роботи є розробка кросплатформного мобільного застосунку, який дозволяє користувачам отримувати актуальну інформацію про героїв гри Dota 2 з використанням фреймворку Flutter та мови програмування Dart.

Розглянуто технології та інструменти, які використовуються у мобільній розробці, в тому числі Flutter та мову Dart. Досліджено архітектуру Flutter-застосунків, обробку JSON-даних, навігація в застосунку. Проведено інтеграцію з API Dota 2, з офіційного сайту гри, для отримання актуальної інформації про кожного з героїв, а саме: зображення, імена, характеристики, здібності та дерево талантів.

У результаті роботи розроблено мобільний застосунок з можливістю перегляду списку героїв, пошуку за ім'ям та перегляду детальної інформації про обраного персонажа.

MOBILE DEVELOPMENT, FLUTTER, DART, API, DOTA 2, JSON, CROSS-PLATFORM APPLICATIONS, OBJECT-ORIENTED PROGRAMMING.

The object of this work is a mobile application for viewing game-related information about Dota 2 characters.

The purpose of the work is to develop a cross-platform mobile application that allows users to access up-to-date information about Dota 2 heroes using the Flutter framework and the Dart programming language.

The work examines technologies and tools used in mobile development, including Flutter and Dart. The architecture of Flutter applications is studied, as well as JSON data processing and in-app navigation. Integration with the official Dota 2 API has been implemented to retrieve current information about each hero, including images, names, attributes, abilities, and talent trees.

As a result, a mobile application has been developed that enables users to browse a list of heroes, search by name, and view detailed information about a selected character.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface (інтерфейс програмування застосунків)

Dota 2 – Defense of the Ancients 2 (оборона стародавніх)

ООП – об’єктно-орієнтоване програмування

JSON – JavaScript Object Notation (запис об’єктів JavaScript)

iOS – iPhone Operating System (операційна система від Apple)

IT – Information Technology (інформаційні технології)

UI – User Interface (дизайн інтерфейсу користувача)

MOBA – Multiplayer Online Battle Arena (багатокористувацька онлайнова бойова арена)

ВСТУП

Мобільна розробка є одним із найактуальніших напрямків сучасної IT-індустрії, що охоплює створення застосунків для платформ Android та iOS. У зв’язку з цим зростає популярність інструментів кросплатформної розробки, які дають змогу створювати один програмний продукт для декількох операційних систем водночас. Мова програмування Dart, що використовується у Flutter, забезпечує високу швидкодію та зручність при побудові динамічних інтерфейсів.

Індустрія відеоігор також невпинно розвивається, і багатьом гравцям потрібні додаткові застосунки для вивчення героїв, здібностей та механік гри. Однією з найпопулярніших багатокористувацьких онлайн-ігор є Dota 2 – командна стратегія в реальному часі, в якій кожен герой має унікальні характеристики та здібності. Швидкий доступ до актуальної інформації про героїв є важливим для кожного гравця.

Актуальність роботи полягає в розробці мобільного застосунку, що дозволяє гравцям швидко й ефективно переглядати актуальні характеристики про героїв гри Dota 2. Застосунок має поєднувати простоту використання, привабливий інтерфейс, достовірну інформацію про персонажів та надійне підключення до відкритого API для отримання актуальних даних.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальна характеристика гри Dota 2

1.1.1 Жанр та популярність гри

Dota 2 – це одна з найвідоміших і найуспішніших багатокористувацьких онлайн-ігор у жанрі MOBA, розроблена компанією Valve Corporation. Вона є повноцінним продовженням модифікації Dota, створеної на основі Warcraft III. З моменту свого офіційного релізу у 2013 році гра здобула величезну популярність та стала ключовою дисципліною у сфері кіберспорту.

Жанр MOBA передбачає стратегічні командні битви, де гравці змагаються за контроль над картою і намагаються знищити базу опонентів. Dota 2 має стабільну базу гравців, яка сягає мільйонів по всьому світу. Вона регулярно оновлюється, розвивається з урахуванням зворотного зв’язку спільноти та бере участь у щорічних турнірах на кшталт The International, де призовий фонд сягає десятків мільйонів доларів.

1.1.2 Структура гри та значення героїв

Основу геймплею становить командна взаємодія: дві команди по п’ять гравців змагаються на симетричній мапі, кожна команда має власну базу, яку потрібно захистити та водночас знищити ворожу.

Герої в Dota 2 мають ключове значення для успіху команди. Вибір героя та його правильне використання може бути вирішальним для перемоги. Кожен герой має унікальні характеристики, серед яких основний атрибут (сила, спритність, інтелект), роль в команді (наприклад, Carry, Support, Tank, Initiator) та тип атаки (ближній або дальній бій).

1.1.3 Класифікація героїв за ролями та характеристиками

У Dota 2 герої класифікуються за кількома характеристиками, зокрема за основним атрибутом, типом атаки та ігровою роллю. Основний атрибут визначає базові характеристики героя, впливає на ефективність його атак і рівень здоров'я. Існує чотири типи атрибутів: Сила, яка підвищує запас здоров'я та регенерацію; спритність**,** що збільшує швидкість атаки та броню; інтелект**,** який впливає на запас мани, її відновлення та ефективність магічних здібностей. У 2023 році було введено новий тип – універсальний. Герої з цим атрибутом отримують однакове збільшення шкоди від кожного з трьох основних атрибутів, що робить їх гнучкими у використанні різноманітних предметів і стратегій.

Тип атаки поділяє героїв на тих, що атакують зблизька (ближній бій) і на дистанції (дальній бій), що має значення для позиціонування у грі. Окрім цього, кожен герой виконує одну або кілька ролей, які визначають його функціональне призначення у грі: Carry (основне джерело шкоди у пізній грі), Support (підтримка команди), Initiator (розпочинає командні бої), Nuker (наносить сильну шкоду за короткий час), а також інші, як-от Disabler, Durable, Escape, Pusher тощо. Від поєднання атрибутів, типу атаки та ролі залежить, як гравець буде використовувати героя в командній взаємодії.

У грі Dota 2 атрибути відіграють ключову роль у визначенні бойових характеристик кожного героя. Вони впливають на виживаність, ефективність атак, використання здібностей та загальну роль героя на полі бою. Кожен герой володіє трьома основними атрибутами: силою, спритністю та інтелектом, один з яких є для нього основним. Саме основний атрибут визначає базову шкоду, що наноситься під час звичайної атаки, і в залежності від класу героя може суттєво впливати на його бойову ефективність.

Сила відповідає за кількість очок здоров'я героя та швидкість його відновлення. Герої з високим рівнем сили здатні довше витримувати шкоду, тому зазвичай використовуються як «танки» – ті, хто перебуває на передовій

та приймає основну частину атак супротивника. Спритність визначає рівень броні героя та швидкість його атак. Герої, що спеціалізуються на нанесенні шкоди зі звичайних атак, часто мають високий показник спритності, що дозволяє їм атакувати швидко та ефективно ухилятися від шкоди. Інтелект забезпечує героя маною, яка необхідна для використання активних здібностей, та визначає швидкість її відновлення. Інтелектуальні герої зазвичай є магами або сапортами й використовують вміння як основний інструмент впливу на хід бою.

Важливим ресурсом у грі є мана – енергетичний запас героя, який витрачається при використанні здібностей або активних предметів. Кожне вміння має свою вартість у мані, і якщо її недостатньо, герой не зможе виконати дію. Збільшення показника інтелекту безпосередньо впливає на розмір манапулу та швидкість його відновлення, що критично важливо для ефективної гри героями, які покладаються на магічні атаки.

Ще одним важливим поняттям у грі є кулдаун – період часу, який повинен пройти після використання здібності до її повторного застосування. Кожне вміння має власний кулдаун, який регулює частоту його використання. Наприклад, сильніші здібності, особливо ультимативні, зазвичай мають тривалий кулдаун, що вимагає стратегічного планування їх використання. Деякі предмети та ефекти можуть зменшувати час відновлення, що робить героя більш гнучким і небезпечним у бою.

Таким чином, атрибути, мана та кулдаун є базовими механіками Dota 2, які визначають функціональність і стиль гри кожного героя. Їх правильне розуміння і ефективне використання є запорукою успішного виконання поставлених ігрових завдань, а також дозволяє адаптувати героя під конкретні ситуації в матчі. У рамках створення мобільного застосунку для перегляду характеристик героїв Dota 2, було важливо відобразити саме ці аспекти, оскільки вони є основною інформацією, що визначає можливості персонажів.

1.2 Інформаційна структура даних

1.2.1 Необхідні дані: ім’я, характеристики, здібності

У процесі створення застосунку для перегляду інформації про героїв Dota 2 важливо визначити, які саме дані необхідно обробляти та відображати. Кожен герой має унікальне ім’я, зображення, опис, набір базових характеристик (сила, спритність, інтелект або універсальність), рівень здоров’я, мани, швидкість атаки, броню, швидкість пересування тощо. Крім базових характеристик, критичну роль відіграє інформація про здібності – активні та пасивні вміння, які безпосередньо впливають на ігровий процес. Для кожного героя також необхідно відобразити дерево талантів – набір вибіркових покращень, доступних на певних рівнях. Повноцінне уявлення про персонажа неможливе без цих елементів, тому мобільний застосунок має забезпечити зручний доступ до них у структурованому вигляді.

1.2.2 Формати подання даних (JSON, API)

У сучасній розробці мобільних застосунків обмін даними між клієнтською частиною та зовнішніми сервісами, зокрема у сфері ігрової індустрії, найчастіше реалізується за допомогою API із використанням формату JSON. Формат JSON є текстовим форматом обміну даними, який відзначається лаконічністю, легкістю сприйняття людиною та широкою підтримкою з боку більшості мов програмування.

У контексті розробки мобільного застосунку для перегляду інформації про героїв гри Dota 2, JSON-формат виступає основним способом отримання структурованих даних із відкритого або внутрішнього API. Отримані дані містять у собі повну інформацію про героїв: локалізовану назву, унікальний ідентифікатор, опис, набір характеристик (сила, спритність, інтелект,

здоров’я, мана тощо), список здібностей (з відповідними описами, витратами мани, часом перезарядки, типом ефекту), а також посилання на графічні ресурси (іконки, зображення моделей тощо).

Кожен об’єкт JSON має вкладену структуру. Наприклад, атрибут abilities зазвичай є масивом об’єктів, у якому кожен об’єкт представляє окрему здатність героя та містить назву, опис, показники витрат ресурсів, cooldown, та інші технічні деталі. Це дозволяє мобільному застосунку ефективно парсити та відображати дані у зручному для користувача вигляді.

Flutter, як фреймворк для створення мобільних інтерфейсів, забезпечує ефективну роботу з HTTP-запитами та обробкою JSON-формату. Для виконання запитів можуть використовуватись як базові засоби (http пакет), так і розширені бібліотеки, зокрема dio (яка підтримує перехоплення запитів, повторні спроби, токенізацію), retrofit (що дозволяє організовувати виклики API у вигляді типізованих методів), та json_serializable (для автоматичної генерації моделей з підтримкою серіалізації та десеріалізації даних).

Використання цих інструментів дозволяє підтримувати високий рівень організації коду, зменшити ризик помилок при обробці складних структур, а також значно пришвидшити розробку за рахунок генерації моделей, адаптерів та інтеграції з BLoC-архітектурою, яка широко застосовується у Flutter-проєктах.

Таким чином, формат JSON у поєднанні з REST API та сучасними інструментами екосистеми Flutter створює ефективне середовище для передачі та обробки ігрових даних, що є критично важливим для побудови надійного та масштабованого мобільного застосунку.

1.2.3 Частота оновлення ігрових даних

У Dota 2 періодично виходять оновлення, які змінюють характеристики героїв, додають нові здібності або навіть вводять абсолютно нових персонажів. Ці оновлення можуть бути як невеликими покращеннями з балансуванням, так і великими оновленнями з новим контентом. Тому важливо, щоб застосунок мав механізм регулярного оновлення інформації. Це може реалізовуватись через динамічне завантаження даних із зовнішніх API при кожному запуску або за допомогою періодичної синхронізації. Також можливо реалізувати локальне кешування даних – тобто збереження останньої версії в пам’яті пристрою з оновленням лише за потреби. Частота оновлення залежить від змін у грі та функціональних вимог – для більшості користувачів важливо мати актуальні дані, особливо під час активної гри або підготовки до матчів.

1.3 Джерела інформації про героїв

1.3.1 OpenDota API та його можливості

Існує кілька джерел, з яких можна отримувати дані про героїв гри Dota 2. Це можуть бути офіційні ресурси, сторонні API та сервіси, що спеціалізуються на зборі та аналізі статистики гри. Важливою складовою розробки застосунку є правильний вибір джерел даних, оскільки це визначає актуальність і точність інформації, яка буде надана користувачеві.

Одним з основних джерел є OpenDota API – безкоштовне API, яке надає доступ до широкого спектра статистичних даних про гру Dota 2. Це включає інформацію про героїв, їхні характеристики, здібності, популярність, а також деталі щодо ігор і матчів. API дозволяє отримувати дані не тільки про поточні властивості героїв, а й історичну інформацію, що є

важливою для аналізу зміни характеристик персонажів у різних оновленнях гри.

OpenDota API має просту і зрозумілу структуру запитів і відповідей, що робить його ідеальним для інтеграції в мобільні застосунки. До того ж, цей API підтримує велику кількість параметрів фільтрації для отримання конкретних даних про героїв.

1.3.2 Інші потенційні джерела

Окрім OpenDota API, для отримання актуальної інформації про героїв гри можна використовувати інші джерела. Одним з них є офіційний сайт гри Dota 2, де публікуються новини, зміни в балансі героїв та їх здібностей, а також інші дані, що стосуються гри. Офіційний сайт є важливим джерелом для отримання підтвердженої та актуальної інформації про героїв, оскільки дані безпосередньо надходять від розробників.

Крім того, існують сторонні сервіси та вебсайти, такі як Dotabuff, DotaPlus, які надають додаткову аналітику, статистику та інші дані про героїв, стратегії і матчі. Вони можуть бути корисними для розширення можливостей застосунку, зокрема для збору додаткової статистики, яка не завжди доступна через OpenDota API.

1.3.3 Технічні аспекти взаємодії з API

При роботі з API важливо враховувати кілька технічних аспектів. Для доступу до даних часто необхідна автентифікація користувача, що може включати отримання API-ключа. Крім того, багато API мають ліміти на кількість запитів, які можна здійснити за певний період часу. Це важливо

враховувати, щоб не перевищити дозволену кількість запитів і уникнути блокування доступу до сервісу.

Структура відповідей API зазвичай містить JSON-формат, що є зручним для обробки в мобільних застосунках. Важливо правильно обробляти дані, зокрема виконувати перевірку на наявність помилок у відповіді API, забезпечити коректну обробку статусів відповіді та обробку помилок, що можуть виникнути під час взаємодії з API.

1.4 Аналіз існуючих застосунків

1.4.1 Огляд аналогів

Вивчення існуючих рішень на ринку, які вже пропонують подібний функціонал для перегляду інформації про героїв гри Dota 2, є важливим етапом при розробці нового застосунку. Це дозволяє оцінити сильні та слабкі сторони наявних продуктів, визначити найбільш популярні функції, а також виявити прогалини, які може заповнити новий застосунок. Огляд аналогів дозволяє не тільки зібрати корисну інформацію для розробки, але й дає змогу зрозуміти потреби кінцевих користувачів.

Існує кілька популярних застосунків та онлайн-сервісів, які надають інформацію про героїв Dota 2, їх здібності та статистику. Серед них найбільш відомими є:

- DotaPlus, офіційний застосунок від Valve для Dota 2, який надає доступ до глибокої статистики, допомагає покращити навички гри через різні інструменти, як-от аналітика матчів, історія ігор, поради щодо вибору героїв. Однак, користувачі часто зазначають високу вартість підписки та обмежений функціонал без платного доступу;
- Dotabuff, один з найбільш популярних онлайн-сервісів для аналізу статистики та характеристик героїв. Dotabuff надає глибокий аналіз ігор, популярних героїв, їхніх здібностей, а також статистику за різними

періодами часу. Незважаючи на велику кількість даних, інтерфейс може здатися занадто перевантаженим для новачків, а доступ до деяких функцій обмежений;

– Overwolf, платформа, яка дозволяє створювати надбудови для ігор, у тому числі для Dota 2. Через застосунки на базі Overwolf, гравці можуть отримувати дані про героїв, історію матчів, геймерську статистику безпосередньо під час гри. Однак цей продукт більше орієнтований на геймерів, які хочуть отримати функціональність безперервного моніторингу.

1.4.2 Порівняльний аналіз

У порівнянні з іншими застосунками, існуючі рішення мають як переваги, так і недоліки:

– з точки зору функціоналу: DotaPlus пропонує глибокі аналітичні інструменти для покращення гри, але багато функцій доступні лише за передплату; Dotabuff надає доступ до великих обсягів статистики безкоштовно, але інтерфейс може бути складним для нових користувачів; Overwolf дозволяє інтегрувати різноманітні функції, однак це не завжди зручно для користувачів, які шукають простоту та зручність;

– з точки зору зручності всі зазначені застосунки мають деякі труднощі в області зручності використання. Наприклад, Dotabuff може виглядати перевантаженим для новачків, а DotaPlus обмежує доступ до більшості функцій безкоштовної версії;

– продуктивність таких застосунків часто залежить від того, як вони обробляють великі обсяги даних. Dotabuff працює ефективно, але інколи помічається певне уповільнення при відображенні результатів за великий період часу. Застосунки на платформі Overwolf можуть іноді бути ресурсномісткими, якщо активно використовуються додаткові надбудови.

1.4.3 Визначення недоліків

Аналіз існуючих застосунків вказує на кілька недоліків, які може виправити новий мобільний застосунок.

Новий застосунок може зосередитись на простому та зрозумілому інтерфейсі, який дасть доступ до найбільш важливих даних без зайвих складнощів, що стане плюсом для новачків.

Існуючі застосунки часто не оновлюють інформацію в реальному часі або з певною затримкою. Для нового застосунку можна реалізувати механізм автоматичного оновлення даних при кожному запуску або інтеграцію з API для отримання найсвіжішої інформації.

Створення застосунку з можливістю безкоштовного використання більшої частини функціоналу, з опцією преміум-функцій для просунутих користувачів, може залучити ширшу аудиторію.

Існуючі рішення часто мають обмеження, коли йдеться про адаптацію до мобільних платформ. Створення кросплатформного застосунку з оптимізованим інтерфейсом для смартфонів може значно підвищити зручність користування.

Новий застосунок, орієнтуючись на ці аспекти, може запропонувати користувачам більш зручний, актуальний та ефективний інструмент для роботи з даними про героїв гри DOTA 2.

1.5 Вибір технології для реалізації

1.5.1 Обґрунтування вибору Flutter як фреймворку

У процесі розробки мобільного застосунку для перегляду інформації про героїв гри Dota 2 необхідно прийняти обґрунтоване рішення щодо вибору технологій, які забезпечать високу продуктивність, зручність у використанні та можливість ефективної роботи з даними. Враховуючи

специфіку проєкту, важливо підібрати сучасні інструменти, які дозволять швидко та якісно реалізувати всі необхідні функціональні можливості.

Flutter є потужним фреймворком для розробки мобільних застосунків, який дозволяє створювати кросплатформні рішення для iOS та Android з єдиної кодової бази. Одна з ключових переваг Flutter полягає в його здатності забезпечувати високу продуктивність завдяки компіляції в нативний код. Це дозволяє забезпечити швидкий і плавний інтерфейс користувача, що є важливим для застосунків, де користувачі повинні швидко отримувати оновлення та переглядати великий обсяг даних.

Flutter також дозволяє легко створювати адаптивні інтерфейси, що мають гарний вигляд на різних пристроях, незалежно від розміру екрану або операційної системи. Оскільки застосунок для перегляду даних про героїв Dota 2 повинен бути доступний як для користувачів Android, так і для iOS, використання Flutter дозволить значно скоротити час розробки, оскільки не потрібно створювати окремі версії для кожної платформи.

Також важливо відзначити, що Flutter має потужну екосистему з численними плагінами та бібліотеками, що значно спрощує інтеграцію з іншими сервісами, зокрема з API для отримання даних про героїв Dota 2.

1.5.2 Dart як основна мова програмування

Dart є мовою програмування, що використовується для розробки в Flutter. Вона є об’єктно-орієнтованою, з багатим набором можливостей для управління асинхронними операціями, що особливо корисно при роботі з зовнішніми API. У випадку цього проєкту, Dart дозволяє ефективно обробляти запити до API, отримувати дані в реальному часі та забезпечувати швидку обробку великих обсягів інформації про героїв.

Dart має вбудовану підтримку асинхронного програмування через конструкції async/await, що значно полегшує роботу з API та обробку великих наборів даних.

Dart має простий та зручний синтаксис, що дозволяє швидко освоїти мову та продуктивно працювати над проєктом.

Завдяки тому, що Dart компілюється в нативний код, застосунки, створені на ньому, працюють швидко й ефективно. Dart дозволяє також використовувати численні пакунки, що значно прискорюють процес розробки. Для взаємодії з API використовуються популярні бібліотеки, що спрощують реалізацію HTTP-запитів та обробку JSON-даних.

1.5.3 REST API, JSON та локальне кешування

Для отримання та обробки даних про героїв гри Dota 2 застосунок буде використовувати REST API. Цей підхід дозволяє здійснювати запити до зовнішніх серверів, отримувати актуальну інформацію та відображати її користувачеві. REST API є стандартом для взаємодії між вебсервісами, забезпечуючи гнучкість і простоту у використанні.

Дані, що передаються через API, зазвичай використовуються в форматі JSON. Це формат, який є легким для розуміння та обробки як для розробників, так і для користувачів. Він також є компактним, що дозволяє зменшити навантаження на мережу та швидко обробляти дані. JSON дозволяє передавати складні структури даних, що включають вкладені об’єкти, масиви та інші дані, необхідні для опису героїв.

Одним з ключових аспектів є локальне кешування даних, що дозволяє зберігати отриману інформацію на пристрої користувача, зменшуючи потребу в регулярних запитах до API. Це важливо для забезпечення швидкої роботи застосунку, навіть коли немає стабільного інтернет-з’єднання або коли необхідно зменшити навантаження на сервери. Локальне кешування

дозволяє зберігати дані на пристрої й оновлювати їх лише при необхідності, зберігаючи актуальність інформації без втрат у продуктивності.

1.5.4 Особливості кросплатформних рішень

Однією з головних переваг Flutter є можливість розробки кросплатформних застосунків, які можуть працювати на обох основних мобільних операційних системах – Android та iOS – за допомогою єдиної кодової бази. Це дозволяє значно знизити витрати на розробку та тестування, а також пришвидшити процес впровадження нового функціоналу.

Кросплатформність Flutter також забезпечує високу продуктивність застосунків, що неможливо гарантувати в разі використання інших кросплатформених інструментів. Завдяки Flutter, створення графічних інтерфейсів не обмежується стандартними елементами дизайну, тому можна легко створювати інтерфейси з високим рівнем кастомізації. Це є важливим при створенні застосунку, де користувачам необхідно зручно переглядати великі обсяги даних про героїв та їх здібності.

Кросплатформне рішення також дозволяє досягти високої узгодженості функцій на всіх платформах, що означає, що користувачі з різних пристроїв отримуватимуть однаковий досвід користування застосунком. Це створює позитивний досвід для користувачів і дозволяє залучати більшу аудиторію.

1.6 Постановка задачі

Таким чином, розробка мобільного застосунку для перегляду інформації про героїв гри Dota 2 є актуальним завданням для покращення користувацького досвіду та ефективного доступу до важливої ігрової інформації. Тому ставиться завдання розробки застосунку, який дозволить

отримувати та відображати актуальну інформацію про героїв, їх характеристики, здібності, дерево талантів та інші важливі дані через API гри.

Об’єктом роботи є мобільний застосунок для перегляду даних про героїв Dota 2.

Метою роботи є розробка мобільного застосунку для зручного перегляду та аналізу інформації про героїв Dota 2, що включає список усіх персонажів, їх базові характеристики, здібності та інші важливі деталі, доступні через API гри.

Для досягнення мети необхідно вирішити такі завдання:

- провести аналіз існуючих рішень для перегляду інформації про героїв гри Dota 2;
- розробити алгоритм для отримання актуальних даних з OpenDota API;
- реалізувати функціонал для перегляду списку героїв, пошуку за іменем та відображення детальної інформації про героя;
- розробити механізм для періодичного оновлення даних з API.

2 ПРОЄКТУВАННЯ ФУНКЦІОНАЛЬНОЇ ЧАСТИНИ ЗАСТОСУНКУ

2.1 Аналіз вимог до застосунку

2.1.1 Опис цільової аудиторії та їх потреб

Цільова аудиторія розробленого мобільного застосунку складається переважно з гравців популярної багатокористувацької онлайн-гри Dota 2. Ці користувачі можуть бути як початківцями, які лише починають своє знайомство з ігровим світом, так і досвідченими гравцями, які регулярно беруть участь у матчах, тренуваннях або турнірах. Кожна з цих категорій користувачів має свої специфічні потреби щодо інформації про ігрових героїв, яку можна задовольнити за допомогою зручного, швидкого та функціонального мобільного застосунку.

Для новачків основна потреба полягає у швидкому доступі до базової інформації про персонажів, яку зазвичай важко знайти або запам’ятати, особливо на початку гри. Вони потребують зрозумілої та візуально структурованої інформації, яка допоможе ознайомитися з героями, їхніми ролями, характеристиками, здібностями, а також з основною термінологією гри. Наявність іконок, описів та основних показників таких як здоров’я, мана, основний атрибут, кулдауни та витрати мани – дозволяє швидко сформувати уявлення про сильні та слабкі сторони кожного героя. Це сприяє кращому розумінню ігрової механіки та більш обґрунтованому вибору персонажа у матчах.

Досвідчені гравці, у свою чергу, вже володіють загальними знаннями про гру, однак потребують інструменту, що дозволить їм оперативно знаходити актуальну інформацію під час підготовки до гри, перегляду трансляцій, обговорення стратегій або аналітики матчів. Часто такі гравці беруть участь у тренуваннях, де потрібно швидко знаходити деталі про конкретного героя або здібність, не відволікаючись на повільні та

перевантажені вебресурси. Мобільний застосунок, який працює без потреби постійного доступу до Інтернету (завдяки кешуванню даних), задовольняє цю потребу в зручний та ефективний спосіб.

Крім того, застосунок може бути корисним і для тренерів, аналітиків, коментаторів кіберспортивних подій, які регулярно працюють з великою кількістю героїв і повинні мати доступ до структурованої, стислої, але повної інформації про них. Також його функціонал може стати у пригоді контент-мейкерам, які створюють гайди або навчальні відео для своєї аудиторії, адже застосунок дозволяє швидко отримати перевірену інформацію з надійного джерела (OpenDota API).

Важливо зазначити, що застосунок орієнтований на широку аудиторію, тому одним із пріоритетів його розробки є створення інтуїтивно зрозумілого, адаптивного та приємного інтерфейсу. Простота навігації, логічна структура відображення даних та візуальна підтримка у вигляді іконок і кольорового оформлення – усе це підвищує зручність використання, а отже – задовольняє потреби як новачків, так і професіоналів.

Таким чином, застосунок вирішує одразу кілька важливих задач: забезпечує інформативність, зручність, швидкодію та доступність, що є ключовими вимогами для сучасного мобільного продукту, орієнтованого на аудиторію гравців Dota 2.

2.1.2 Основні функціональні вимоги

Застосунок для перегляду основної інформації про героїв гри Dota 2 повинен мати кілька ключових функціональних можливостей, що дозволяють користувачеві зручно та швидко отримувати необхідну інформацію. Ось основні з них:

– користувач повинен мати можливість побачити весь список героїв, представлених в грі. Кожен герой повинен бути зображений зі своєю іконкою

та іменем, що дозволяє легко орієнтуватися серед персонажів. Список повинен бути прокручуваним, щоб зручно переглядати всіх героїв, навіть якщо їх кількість велика;

– одна з найважливіших функцій застосунку – це пошуковий рядок, який дає можливість швидко знайти героя за його іменем або певними характеристиками. Користувач може вводити частину імені героя, і система буде автоматично фільтрувати список відповідно до введених даних. Це значно спрощує процес пошуку, особливо коли користувач вже має конкретного героя на увазі, але не пам’ятає його точне ім’я;

– кожен герой в списку має супроводжуватися детальними характеристиками. Це включає основні статистики, такі як сила, спритність, інтелект, рівень здоров’я та мани, швидкість атаки, броня та швидкість пересування. Користувач може вибрати героя для перегляду цієї інформації, що дозволить йому краще зрозуміти особливості кожного персонажа і оцінити, як вони можуть впливати на ігровий процес;

– інформація про здібності кожного героя є важливою частиною застосунку. Це дозволить користувачам ознайомитися з активними та пасивними здібностями героїв, їх описом, витратами мани, кулдауном та іншими параметрами. Здібності мають бути чітко описані та доступні для перегляду, щоб користувач міг краще орієнтуватися в силах героя та їх впливі на гру;

– кожен герой повинен бути відображений з відповідною іконкою та ім’ям, щоб користувач міг легко ідентифікувати персонажа в списку. Це важливо для швидкої навігації, оскільки дозволяє користувачеві швидко вибирати потрібного героя та ознайомлюватися з його характеристиками та здібностями;

– застосунок повинен підтримувати регулярне оновлення інформації про героїв, зокрема після кожних великих оновлень гри. Це може бути здійснено через синхронізацію з зовнішнім API, яке надає дані про зміни в

грі. Це дозволить користувачам отримувати актуальну інформацію про зміни характеристик героїв, балансування та нові можливості;

– застосунок має працювати як на платформі Android, так і на iOS, щоб охопити більшу кількість користувачів. Для цього вибрано використання Flutter, що дозволяє створювати кросплатформний застосунок з єдиною кодовою базою.

Ці функціональні вимоги допоможуть створити зручний і інтуїтивно зрозумілий застосунок, який буде корисний для будь-якого користувача, незалежно від рівня його досвіду в грі.

2.1.3 Нефункціональні характеристики

Нефункціональні вимоги визначають не стільки саму функціональність застосунку, скільки якість та ефективність його роботи. Вони впливають на загальний досвід користувача, забезпечуючи зручність, продуктивність та стабільність роботи програми. Основні нефункціональні вимоги до застосунку включають:

– зручність інтерфейсу. Для того, щоб застосунок був приємним і простим у використанні, інтерфейс повинен бути інтуїтивно зрозумілим і зручним. Всі елементи інтерфейсу, такі як список героїв, пошуковий рядок, повинні бути розташовані в зручних місцях та забезпечувати легкий доступ до необхідної інформації. Користувачі повинні мати змогу швидко знайти потрібного героя або інформацію про нього без зайвих зусиль;

– швидкодія. Застосунок повинен працювати без значних затримок. Особливо важливою є швидкість відображення списку героїв, пошукових запитів і завантаження додаткової інформації про героїв. Користувач не повинен відчувати затримок чи «заморожувань» інтерфейсу. Це особливо актуально при великій кількості даних або на старих пристроях;

– сумісність з різними пристроями. Застосунок має бути сумісним з різними моделями смартфонів на платформі Android та iOS, включаючи як нові, так і старі версії операційних систем. Це означає, що застосунок повинен коректно працювати на різних екранах, забезпечуючи адаптивний дизайн для різних розмірів дисплеїв та оптимальне відображення інформації на будь-яких пристроях;

– продуктивність. Застосунок має працювати стабільно, не зависати і не викликати помилок або збою при використанні. Всі запити та операції повинні виконуватися швидко, а система має бути здатною обробляти великі обсяги даних, якщо це буде необхідно, без значного впливу на продуктивність;

– оновлення даних. Інформація про героїв повинна регулярно оновлюватися, щоб користувач завжди отримував актуальні відомості про героїв та їх здібності. Це може бути досягнуто шляхом завантаження даних з зовнішніх джерел або через використання API для отримання актуальної інформації з бази даних гри.

Забезпечення виконання цих нефункціональних вимог дозволить користувачам отримати високоякісний досвід роботи з застосунком, зробивши його зручним, швидким та надійним у використанні.

2.2 Проєктування архітектурної структури застосунку

2.2.1 Використання архітектурних шаблонів та моделей

Архітектура застосунку визначає не тільки структуру коду, а й принципи, за якими працюють компоненти застосунку. Вибір архітектурного шаблону є важливим етапом, оскільки він впливає на ефективність розробки, зручність тестування, а також на майбутні можливості масштабування. Для застосунку на Flutter одним з найбільш підходящих архітектурних підходів є модель Model-View-ViewModel (MVVM). Цей шаблон архітектури

забезпечує чітке розділення між логікою управління даними та інтерфейсом користувача, що дозволяє спростити підтримку і розширення функціональності.

Model-View-ViewModel (MVVM) передбачає три основні компоненти: Model, View і ViewModel.

Модель є серцем застосунку і відповідає за зберігання і маніпулювання даними. У випадку нашого застосунку модель буде включати інформацію про героїв: їхні характеристики, здібності, імена, зображення та інші атрибути. Це є структурами даних, які забезпечують відображення інформації користувачеві. Дані, отримані через API, будуть представлено у вигляді об’єктів моделей, що містять всю необхідну інформацію. Модель взаємодіє з зовнішнім API для отримання актуальних даних і передає їх на наступний рівень, у ViewModel, для подальшої обробки та відображення.

Представлення або View є компонентом, що відповідає за відображення інтерфейсу користувача та взаємодію з ним. В даному випадку View відображатиме списки героїв, детальну інформацію про кожного персонажа, а також забезпечить функціональність пошуку та фільтрації. View отримує дані від ViewModel і оновлює UI відповідно до змін. Це дозволяє розділити відображення інформації і бізнес-логіку, що полегшує налагодження інтерфейсу та взаємодії з користувачем.

ViewModel є мостом між Model і View. Вона відповідає за обробку даних, отриманих від Model, і підготовку їх до відображення в View. ViewModel містить бізнес-логіку застосунку, наприклад, обробку пошукових запитів, фільтрацію героїв за певними характеристиками або зміни, що мають бути відображені в інтерфейсі. ViewModel отримує запити від View, обробляє їх і оновлює дані в інтерфейсі користувача. Зв’язок між ViewModel та View зазвичай реалізується через реактивне програмування, що дозволяє автоматично оновлювати UI, коли дані змінюються.

Цей підхід забезпечує чітке розділення задач між компонентами і дозволяє легко підтримувати та оновлювати кожен з них. MVVM також

спрощує тестування, оскільки бізнес-логіка знаходиться в ViewModel і може бути протестована окремо від UI. Оскільки ViewModel не містить конкретної логіки інтерфейсу, її можна тестувати незалежно від вигляду застосунку.

Ще однією перевагою архітектури MVVM є її масштабованість. Коли проєкт росте, можна легко додавати нові функції, зберігаючи код чистим і підтримуваним. Також завдяки використанню реактивних підходів, зміни в даних автоматично відображаються в інтерфейсі без необхідності вручну оновлювати UI, що спрощує розробку та забезпечує кращий досвід користувача.

Таким чином, використання архітектури MVVM для застосунку на Flutter дозволить створити зручну, масштабовану та підтримувану систему для перегляду даних про героїв Dota 2.

2.2.2 Опис компонентів системи та їх взаємодій

Функціональна архітектура мобільного застосунку для перегляду героїв гри Dota 2 базується на взаємодії кількох ключових компонентів, кожен з яких виконує окрему роль у процесі збору, обробки та відображення інформації. Таке розділення дозволяє досягти більшої модульності, забезпечити зручність у розробці, масштабуванні та підтримці застосунку.

Користувацький інтерфейс (UI) відповідає за взаємодію користувача із застосунком. Він відображає список героїв, їхні імена, іконки, характеристики, здібності та інші деталі. Також інтерфейс включає пошукову строку, яка дозволяє фільтрувати героїв за ім’ям. UI реагує на дії користувача, наприклад, натискання на героя для перегляду розширеної інформації. Компонент UI реалізується за допомогою Flutter-виджетів і є тісно пов’язаним з ViewModel, від якої отримує підготовлені до відображення дані.

Менеджер стану (ViewModel) координує дані, що надходять із зовнішніх джерел (наприклад, з API), і передає їх у зручному вигляді до UI. Він також обробляє логіку фільтрації, пошуку, сортування або відбору героїв, і реагує на події, ініційовані користувачем. Менеджер стану є своєрідним «мостом» між даними та інтерфейсом.

Сервіс доступу до API (API Service) відповідає за взаємодію із зовнішніми джерелами інформації, зокрема, з OpenDota API. Він надсилає HTTP-запити, обробляє відповіді у форматі JSON і перетворює отримані дані у формат, зручний для подальшого використання у моделі. Також враховуються можливі помилки під час передачі даних (наприклад, відсутність інтернету, помилки з’єднання або перевищення лімітів запитів).

Моделі даних (Data Models) представляють собою структури, які описують, як зберігається інформація про героїв. Наприклад, одна модель може включати ім’я, здоров’я, силу, інтелект, спритність, список здібностей, посилання на іконку, опис персонажа тощо. Дані з API конвертуються у ці моделі для зручного використання в застосунку.

Взаємодія між компонентами відбувається наступним чином:

– коли користувач відкриває застосунок, UI надсилає запит до ViewModel, яка ініціює завантаження даних через API Service;

– API Service надсилає запит до зовнішнього OpenDota API та отримує список героїв у форматі JSON;

– отримані дані конвертуються у відповідні моделі, які передаються назад у ViewModel;

– ViewModel виконує додаткову обробку (наприклад, фільтрацію або сортування) та надсилає результат у UI;

– UI оновлюється і відображає список героїв або детальну інформацію про обраного персонажа.

Такий підхід до розробки структури застосунку дозволяє ефективно розділити логіку, забезпечити простоту в обслуговуванні та гнучкість під час розширення функціоналу. Усі компоненти працюють незалежно, що

полегшує налагодження, повторне використання коду та адаптацію до змін у зовнішніх джерелах даних.

2.3 Обробка зовнішніх даних

Основною задачею обробки зовнішніх даних у цьому застосунку є ефективне отримання актуальної інформації про героїв гри DOTA 2 через OpenDota API, а також їх кешування для підвищення продуктивності та зменшення навантаження на мережу. Інформація про героїв включає такі параметри, як ім’я, характеристики, здібності, зображення тощо, і постачається у форматі JSON. Доступ до цих даних здійснюється через HTTP-запити з використанням відповідного клієнта, зокрема бібліотек http або dio, що дозволяють реалізовувати надійний механізм взаємодії з REST API.

Застосунок має забезпечити користувачів найбільш актуальною інформацією про героїв гри, яку можна швидко отримати навіть при обмеженому доступі до Інтернету. З цією метою реалізовано механізм локального кешування за допомогою SQLite-бази даних через бібліотеку sqflite. Це дає змогу зберігати отримані з API дані на пристрої користувача, тим самим прискорюючи повторне завантаження інформації та зменшуючи кількість мережевих запитів.

Локальне кешування також відіграє важливу роль у покращенні користувацького досвіду – дані про героїв доступні навіть у разі відсутності підключення до Інтернету. Водночас, при наявності з’єднання з мережею, застосунок може автоматично оновлювати локальне сховище, забезпечуючи релевантність і актуальність даних без необхідності додаткових дій з боку користувача.

З технічного боку, кешування реалізовано шляхом серіалізації отриманих JSON-даних у відповідні Dart-моделі, після чого вони

зберігаються у таблиці бази даних. При повторному запуску застосунку або запиті до списку героїв система перевіряє наявність кешованих даних і лише за їхньої відсутності або застарілості звертається до API.

Таким чином, обробка зовнішніх даних у цьому проєкті охоплює як етап отримання інформації з віддаленого джерела, так і її обробку, зберігання та подальше використання всередині застосунку. Це забезпечує високу швидкодію, стабільність роботи та незалежність від постійного доступу до Інтернету, що є ключовими вимогами до сучасних мобільних рішень.

2.3.1 Отримання та обробка даних з OpenDota API

Застосунок використовує OpenDota API для отримання даних про героїв гри DOTA 2. Це включає в себе інформацію про:

- основні характеристики героїв – імена, атрибути, здоров’я, ману та інші основні дані;
- здібності героїв, що дозволяє здійснити глибший аналіз для користувача;
- іконки та зображення для візуалізації героїв, що надаються API.

Дані отримуються у форматі JSON, який потім перетворюється у моделі Dart. Це дозволяє зручно працювати з ними в мобільному застосунку, застосовувати необхідні фільтри, сортування та інші маніпуляції з даними. Процес обробки також включає механізми для обробки помилок та перевірки на коректність даних.

Завдяки використанню OpenDota API, користувач отримує доступ до актуальної та точної інформації, що дозволяє використовувати її для реалізації різноманітних функцій, таких як пошук героїв, фільтрація за атрибутами та здібностями. Опис даних, які приходять від OpenDota API, коли запитуються дані про героїв гри Dota 2, описані в таблиці 2.1.

Таблиця 2.1 – Дані від OpenDota API про кожного з героїв

Параметр	**Опис**	**Тип даних**	**Приклад значення**
id	Унікальний ідентифікатор героя	Ціле число (int)	1
name	Назва героя	Рядок	‘‘Anti-Mage’’
localized_name	Локалізоване ім’я героя	Рядок	‘‘Анти-Маг’’
primary_attr	Основна атрибутика героя (сила, спритність, інтелект)	Рядок	‘‘agi’’(спритність)
attack_type	Тип атаки героя	Рядок	‘‘Melee’’ (ближній бій)
base_health	Базове здоров’я героя	Ціле число (int)	200
base_health_regen	Базова регенерація здоров’я героя	Число (double)	0.25
base_mana	Базова ману героя	Число (float)	75
base_mana_regen	Базова регенерація мани героя	Число (double)	0,25
base_armor	Базова кількість броні героя	Число (double)	1,5
base_attack_min	Мінімальний рівень шкоди	Ціле число (int)	30
base_attack_max	Максимальний рівень шкоди	Ціле число (int)	40
attack_rate	Час між атаками героя (в секундах)	Число (double)	1,7
move_speed	Швидкість руху героя	Ціле число (int)	300
turn_rate	Швидкість повороту героя	Число (double)	0,7
strGain	Кількість сили, яку буде отримувати герой за новий рівень	Число (double)	1,6
intBase	Базова кількість інтелекту у героя	Число (double)	12
intGain	Кількість інтелекту, яку буде отримувати герой за новий рівень	Число (double)	1,8
icon	URL іконки героя	Рядок	‘‘https://api.opendota.com/assets/heroes/antimage_lg.png’’
abilities	Список здібностей героя з їхніми іменами та описами	Список об’єктів (List<Ability>)	[{‘‘name’’ : ‘‘mana_break’’, ‘‘localized_name’’ : ‘‘Mana Break’’}, ...]
agiBase	Базова кількість спритності у героя	Число (double)	24
agiGain	Кількість спритності, яку буде отримувати герой за новий рівень	Число (double)	2,8
strBase	Базова кількість сили у героя	Число (double)	21

2.3.2 Кешування даних для підвищення продуктивності

Оскільки OpenDota API може мати ліміти на кількість запитів або залежати від стабільності інтернет-з’єднання, важливим аспектом є кешування даних у мобільному застосунку для підвищення продуктивності. В даному випадку використовується кешування через локальну базу даних SQLite за допомогою пакету sqflite. За рахунок нього можна отримати швидкий доступ до даних, зниження навантаження на сервер, оновлення даних і покращення користувацького досвіду.

Кешування дозволяє зберігати дані про героїв в локальній базі даних після першого отримання їх з API. Таким чином, користувачі можуть отримувати доступ до даних про героїв навіть при відсутності Інтернет-з’єднання або поганому зв’язку.

Кешування допомагає значно зменшити кількість запитів до OpenDota API, оскільки дані будуть зберігатися локально та використовуватися повторно. Це дозволяє знизити навантаження на сервери і підвищити загальну ефективність застосунку.

Кешовані дані можуть бути періодично оновлені для підтримки актуальності інформації. Наприклад, користувач може отримати оновлення після кожного запуску застосунку або за допомогою спеціальної функції, яка перевіряє актуальність кешу. Цей підхід дозволяє підтримувати баланс між продуктивністю та точністю даних.

Завдяки кешуванню застосунок може працювати швидше, зменшуючи час на завантаження сторінок та забезпечуючи безперебійну роботу навіть за умов поганого Інтернет-з’єднання. Це покращує загальний досвід користувачів, оскільки вони отримують доступ до даних миттєво.

Завдяки кешуванню застосунок може ефективно працювати з великими обсягами даних, зменшуючи навантаження на сервери та забезпечуючи користувачам більш швидкий та стабільний доступ до потрібної інформації.

2.4 Визначення технологічного стеку для реалізації проєкту

Для розробки мобільного застосунку для перегляду інформації про героїв гри Dota 2 було обрано технології, що забезпечують високу продуктивність, кросплатформність і зручність у розробці. Основним інструментом став фреймворк Flutter, який дозволяє створювати застосунки для Android та iOS з єдиної кодової бази. Flutter підтримує декларативний підхід до побудови інтерфейсу, має багату систему віджетів і забезпечує високу швидкодію завдяки використанню власного графічного рушія.

Мова програмування Dart була обрана як основна, оскільки вона повністю інтегрована з Flutter, підтримує асинхронне програмування та забезпечує зрозумілий синтаксис, що полегшує розробку складних застосунків з великою кількістю взаємодій з мережею.

Для керування станами в застосунку використовується архітектура BLoC (Business Logic Component), яка дозволяє чітко розділити логіку застосунку від користувацького інтерфейсу. Це спрощує тестування, підтримку та масштабування застосунку. Взаємодія між частинами застосунку організована через потоки подій і станів, що відповідає реактивному підходу до розробки.

Для обміну даними із зовнішнім джерелом використовується OpenDota API – відкритий REST API, що надає актуальну інформацію про героїв, їх характеристики та здібності. Для взаємодії з API використовується бібліотека http, яка забезпечує зручну реалізацію HTTP-запитів та обробку відповідей у форматі JSON.

Для покращення продуктивності та зменшення навантаження на мережу реалізовано кешування даних за допомогою бібліотеки sqflite, яка дозволяє працювати з локальною базою даних SQLite. Це забезпечує збереження вже отриманих даних і швидкий доступ до них без потреби повторних запитів. Дані з API зберігаються у вигляді моделей Dart у

відповідних таблицях SQLite, що дозволяє ефективно структурувати локальну інформацію.

Керування залежностями, конфігурація пакетів і організація структури проєкту здійснюється через файл pubspec.yaml, у якому визначаються всі сторонні бібліотеки, ресурси та шрифти, необхідні для застосунку.

Як середовище розробки було обрано Android Studio, яке забезпечує глибоку інтеграцію з Flutter SDK, має розширену підтримку налагодження, емуляторів, автоматичне доповнення коду та зручне управління пакетами. Це середовище дозволяє ефективно тестувати застосунок як на віртуальних пристроях, так і на реальних.

Загалом, обраний стек технологій дозволяє створити стабільний, масштабований і легко підтримуваний мобільний застосунок з інтерактивним інтерфейсом, який ефективно працює з великими обсягами зовнішніх даних.

Додатково реалізовано базову обробку помилок під час мережевої взаємодії, що дозволяє попередити зависання або аварійне завершення роботи застосунку. Усі мережеві запити виконуються асинхронно, що гарантує стабільну роботу інтерфейсу навіть при довготривалому очікуванні відповіді від сервера.

Проєкт має чітко організовану структуру директорій з розділенням на шари: data, domain, presentation. Такий підхід покращує масштабованість застосунку та дозволяє ефективно управляти залежностями.

У разі потреби застосунок можна легко масштабувати для підтримки локалізації та розширення функціональності, зокрема додавання можливостей авторизації, обраних персонажів або персоналізації.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Налаштування середовища розробки

Для реалізації мобільного застосунку було обрано технології Flutter та Dart, які дозволяють створювати кросплатформенні застосунки з єдиною базою коду, забезпечуючи швидку розробку, високу продуктивність та нативний інтерфейс. Як інтегроване середовище розробки використовується Android Studio, що має повну підтримку Flutter SDK, а також необхідні інструменти для запуску, налагодження та тестування застосунку на віртуальних і фізичних пристроях.

Початкове налаштування середовища включає кілька ключових кроків:

Крок 1. Передусім, з офіційного сайту завантажується остання стабільна версія Flutter SDK. Після розпакування архіву у вибране місце на диску, необхідно додати шлях до Flutter SDK у змінну середовищ PATH, що дозволить використовувати Flutter з будь-якої директорії через термінал.

Крок 2. Наступним кроком є встановлення Android Studio – офіційного середовища розробки для Android. Після встановлення необхідно через меню Plugins додати плагіни Flutter і Dart, що забезпечують інтеграцію з Flutter SDK і автозаповнення коду, перевірку синтаксису, запуск симуляторів тощо.

Крок 3. Після завершення налаштування середовища можна створити новий проєкт Flutter.

Крок 4. Наступним кроком є додавання необхідних пакетів та бібліотек, які будуть використані, у файл pubspec.yaml.

Лістинг 3.1 Реалізація файлу pubspec.yaml:

```
name: new_project
description: A new Flutter project.

publish_to: ‘none’
```

```
version: 1.0.0+1

environment:
  sdk: ">=2.16.2 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  floor:
  sqflite: ^2.4.2
  floor_generator:
  http:
  flutter_bloc:
  dio:
  cupertino_icons:
  simple_gradient_text:
  video_player: ^2.9.5

dev_dependencies:
  floor_generator:
  build_runner:
  flutter_test:
    sdk: flutter

  flutter_lints:

flutter:

  uses-material-design: true

  assets:
    - assets/
```

Крок 5. Після внесення змін необхідно виконати команду «flutter pub get».

Крок 6. Та запустити програму, щоб протестувати інтерфейс, логіку взаємодії з API, кешування та відображення даних у мобільному середовищі.

3.2 Програмна реалізація

3.2.1 Опис реалізації моделей даних для структурування інформації про персонажів гри Dota 2

В рамках реалізації мобільного застосунку на Flutter для перегляду інформації про героїв гри Dota 2 було створено набір моделей, що забезпечують обробку й трансформацію даних, отриманих у форматі JSON. Розроблені класи відображають ієрархічну структуру вхідного JSON-документу, що надходить з API, та реалізують механізми серіалізації та десеріалізації об’єктів.

Для реалізації структури даних необхідно виконати наступні кроки:

Крок 1. Зробити імпорт та конвертацію JSON (рис. 3.1). В першу чергу імпортується стандартний модуль dart:convert для роботи з JSON. Також підключаються моделі здібностей героя, які розміщені в окремому файлі heroAbilities.dart.

Далі реалізуються дві функції для конвертації:

- welcomeFromJson – приймає JSON-рядок, декодує його у Map і створює об’єкт класу Welcome;
- welcomeToJson – виконує зворотнє перетворення об’єкта Welcome у JSON-рядок.

Це створює зручний інтерфейс для отримання та передачі даних.

```dart
import 'dart:convert';

import 'package:new_project/data/models/heroAbilities.dart';

Welcome welcomeFromJson(String str) => Welcome.fromJson(json.decode(str));

String welcomeToJson(Welcome data) => json.encode(data.toJson());
```

Рисунок 3.1 – Імпорт та конвертація JSON

Крок 2. Опис кореневої модель Welcome (рис. 3.2), яка відображає структуру відповіді API на найвищому рівні. Вона містить поле result, яке інкапсулює основний зміст відповіді.

Метод fromJson відповідає за створення екземпляру класу на основі даних JSON, викликаючи парсер відповідного вкладеного класу Result. Метод toJson виконує зворотнє кодування в JSON-формат.

```
class Welcome {
  Welcome({
    required this.result,
  });

  Result result;

  factory Welcome.fromJson(Map<String, dynamic> json) => Welcome(
        result: Result.fromJson(json["result"]),
      );  // Welcome

  Map<String, dynamic> toJson() => {
        "result": result.toJson(),
      };
}
```

Рисунок 3.2 – Реалізація класу Welcome

Крок 3. Визначення класу Result (рис. 3.3), який містить ключові поля відповіді API:

- data, основний блок із переліком персонажів;
- status, числовий код статусу відповіді.

Цей клас відповідає за коректний парсинг та серіалізацію вкладеного об’єкта Data.

```dart
class Result {
  Result({
    required this.data,
    required this.status,
  });

  Data data;
  int status;

  factory Result.fromJson(Map<String, dynamic> json) => Result(
        data: Data.fromJson(json["data"]),
        status: json["status"],
      );  // Result

  Map<String, dynamic> toJson() => {
        "data": data.toJson(),
        "status": status,
      };
}
```

Рисунок 3.3 – Реалізація класу Result

Крок 4. Опис класу Data (рис. 3.4), який містить список персонажів – героїв. Для цього необхідно виконати ітерацію по JSON-масиву heroes, створюючи список об’єктів класу DotaHero.

Серійалізація виконується шляхом зворотного відображення списку героїв у форматі JSON.

```dart
class Data {
  Data({
    required this.heroes,
  });

  List<DotaHero> heroes;

  factory Data.fromJson(Map<String, dynamic> json) => Data(
        heroes: List<DotaHero>.from(json["heroes"].map((x) => DotaHero.fromJson(x))),
      );  // Data

  Map<String, dynamic> toJson() => {
        "heroes": List<dynamic>.from(heroes.map((x) => x.toJson())),
      };
}
```

Рисунок 3.4 – Реалізація класу Data

Крок 5. Визначення класу DotaHero, який моделює основні характеристики персонажа. Для кожного героя містяться:

- ідентифікатори (id, name, orderId);
- локалізовані рядки (nameLoc, bioLoc, hypeLoc, npeDescLoc);
- базові атрибути (сила, спритність, інтелект і їх прирости);
- основні параметри бою (атака, броня, швидкість руху, зір, здоров'я та мана);
- список здібностей (абіліті), які також представляються у вигляді об'єктів класу Ability.

Метод fromJson здійснює глибокий парсинг JSON:

- спочатку перетворює список здібностей у об'єкти Ability;
- далі збирає всі спеціальні значення (specialValues) із здібностей;
- обробляє бонуси (bonuses), якщо такі є.

Метод toJson реалізує обернену операцію, формуючи JSON-структуру зі значень полів.

Крок 6. Реалізація набору Dart-класів для моделювання відповіді API, що містить інформацію про персонажів гри Dota 2. Основна мета – здійснити коректний парсинг JSON-даних, отриманих від сервера, у відповідні об'єкти Dart, а також забезпечити обернене серіалізування цих об'єктів у JSON. Це необхідно для ефективної роботи мобільного застосунку на Flutter, який відображає список героїв із базовими характеристиками.

У представленому фрагменті реалізовано чотири взаємопов'язані класи, які слугують для десеріалізації JSON-відповіді з API гри Dota 2. Центральною точкою цієї структури є клас HeroesResponse, який виступає кореневою моделлю усієї відповіді. Він містить єдине опціональне поле result типу Result?, що інкапсулює доступ до вкладених даних. Конструктор HeroesResponse приймає необов'язковий параметр result, а метод fromJson створює об'єкт цієї моделі на основі JSON-мапи, за необхідності ініціалізуючи вкладений об'єкт типу Result через виклик Result.fromJson.

Метод toJson, у свою чергу, повертає JSON-мапу, включаючи ключ result, лише якщо значення цього поля не є null.

Клас Result інкапсулює поле data, що репрезентує тіло фактичної відповіді, яка містить інформацію про героїв. Це поле є опціональним (Data?). Конструктор класу дозволяє передавати значення data при ініціалізації, а метод fromJson ініціалізує його, якщо вхідна мапа містить ключ data. Метод toJson формує JSON-мапу лише з наявними даними, зокрема включає поле data, якщо воно не є null.

Наступний рівень структури займає клас Data, який відповідає за збереження колекції героїв. Його поле heroes є списком об’єктів типу Heroes і також є опціональним. Конструктор приймає необов’язковий список heroes. Метод fromJson перевіряє наявність ключа heroes у вхідному JSON та, у разі його наявності, створює список об’єктів класу Heroes шляхом ітерації через масив та виклику Heroes.fromJson для кожного елемента. Метод toJson повертає мапу з ключем heroes, значенням якого є список героїв, якщо цей список не є порожнім.

Найнижчий рівень цієї структури представлений класом Heroes, що моделює базову інформацію про окремого героя. Усі його поля є опціональними (nullable), що дозволяє враховувати випадки неповних або частково відсутніх даних у відповіді API. Поле id зберігає унікальний числовий ідентифікатор героя. Поле name містить внутрішню назву героя, тоді як nameLoc та nameEnglishLoc відповідають за локалізовані назви (відповідно у стандартній локалізації та англійською). Поле primaryAttr вказує на основну характеристику героя у вигляді числового коду, а complexity відображає складність освоєння персонажа. Конструктор класу дозволяє передавати будь-яке з цих значень, а метод fromJson забезпечує призначення значень відповідним полям на основі ключів у JSON. Метод toJson формує мапу з даними героя, придатну для подальшого перетворення у формат JSON.

Загалом, ця ієрархічна модель відповідає принципам об’єктно-орієнтованого програмування та дозволяє ефективно структурувати й обробляти дані про героїв гри Dota 2, отримані з зовнішнього API.

Крок 7. У файлі HeroAbilities.dart реалізовується набір Dart-класів для моделювання відповіді API, що містить детальну інформацію про здібності героїв гри Dota 2. Основна мета – здійснити коректний парсинг складних JSON-структур, отриманих із сервера, у відповідні об’єкти Dart, а також забезпечити зворотну серіалізацію цих об’єктів у JSON-формат. Це необхідно для коректної обробки, відображення й повторного використання інформації у Flutter-застосунку, який показує характеристики, здібності, ефекти предметів, витрати мани, кулдауни, та інші параметри, пов’язані зі здібностями героїв. Особливу увагу приділено обробці типів здібностей, а також специфічним випадкам – наприклад, формуванню назв талантів на основі параметрів.

3.2.2 Створення екрану для перегляду списку всіх героїв

Для створення екрану перегляду всіх героїв гри Dota 2 у мобільному застосунку на Flutter першим кроком є реалізація бізнес-логіки за допомогою архітектурного шаблону BLoC (Business Logic Component). Цей підхід дозволяє чітко розділити логіку обробки даних від відображення інтерфейсу користувача, що є ключовим принципом у побудові масштабованих та підтримуваних застосунків.

Починаємо з оголошення подій (рис. 3.5) – класів, що визначають, які дії може ініціювати користувач або сама система. У нашому випадку ці події описані у файлі heroes_event.dart і наслідують абстрактний клас HeroEvent. Існує кілька конкретних реалізацій подій:

- LoadAllHeroes – ініціює завантаження повного списку героїв з репозиторію;

– GetSearchListHeroes – фільтрує список героїв відповідно до введеного користувачем тексту;

– ClickHero – резервна подія, яка може використовуватись для обробки кліку по герою (в цьому прикладі вона ще не реалізована).

```dart
abstract class HeroEvent{}

class ClickHero extends HeroEvent{}

class LoadAllHeroes extends HeroEvent{}

class GetSearchListHeroes extends HeroEvent{
  final String name;

  GetSearchListHeroes({this.name = ''});
}
```

Рисунок 3.5 – Реалізація файлу heroes_event.dart

Усі ці події передаються до BLoC-класу HeroBloc (рис. 3.6), який керує обробкою подій та оновленням станів (states). Основна логіка обробки реалізована у методі on<EventType>, що дозволяє виконувати певні дії, коли конкретна подія відбувається.

При обробці LoadAllHeroes, HeroBloc викликає метод fetchAllHeroes() з DotaRepository, який отримує дані (наприклад, з API або локального джерела). Якщо завантаження відбувається успішно, новий стан з оновленим списком героїв (heroList) передається у стрім станів. Якщо виникає помилка – створюється стан із повідомленням про неуспішне завершення операції. Для уникнення повторного завантаження список героїв зберігається у змінній listAllHero, і при повторному виклику події завантаження (LoadAllHeroes) перевіряється, чи дані вже є в пам’яті.

Подія GetSearchListHeroes використовується для реалізації пошуку. Вона фільтрує список героїв за ключовим словом у полі nameLoc, створюючи оновлений стан з відфільтрованим списком.

```dart
import ...

class HeroBloc extends Bloc<HeroEvent, HeroState> {
  final DotaRepository dotaRepository;

  List<Heroes>? listAllHero = null;

  HeroBloc({required this.dotaRepository}) : super(HeroState(heroList: [])) {
    on<LoadAllHeroes>((event, emit) async {

      if(listAllHero != null){
        return;
      }

      var result = await dotaRepository?.fetchAllHeroes();
      if (result!.error == null) {
        listAllHero = result.success;
        emit(HeroState(heroList: result.success, formStatus: SubmissionSuccess()));
      } else {
        emit(HeroState(heroList: result.success, formStatus: SubmissionFailed(Exception(result.error))));
      }
    });

    on<GetSearchListHeroes>((event, emit) async {
      var filteredList =
          listAllHero?.where((element) => element.nameLoc!.toLowerCase().contains(event.name.toLowerCase())).toList() ??
              List.empty();
      emit(HeroState(heroList: filteredList, formStatus: SubmissionSuccess()));
    });
  }
}
```

Рисунок 3.6 – Реалізація файлу heroes_bloc.dart

Стан HeroState, що описаний у файлі heroes_state.dart (рис 3.7), містить:

– heroList, список героїв, які мають бути відображені в інтерфейсі;

– formStatus, об’єкт класу FormSubmissionStatus, який відображає поточний стан обробки даних (наприклад, FormSubmitting, SubmissionSuccess, SubmissionFailed).

```dart
import 'package:new_project/presentation/form_submission_status.dart';

import '../../../data/models/heroesResponseDTO.dart';

class HeroState {
  final List<Heroes> heroList;
  final FormSubmissionStatus formStatus;

  HeroState({required this.heroList, this.formStatus = const FormSubmitting()});
}
```

Рисунок 3.7 – Реалізація файлу heroes_state.dart

Таким чином реалізована повноцінна бізнес-логіка, яка дозволяє завантажувати, зберігати, обробляти та фільтрувати список героїв гри Dota 2. Цей BLoC-компонент стане основою для побудови UI-частини, що буде відображати героїв, реагувати на пошук і забезпечить зручний та швидкий інтерфейс для користувача.

Розглянемо поетапний аналіз коду файлу heroes_page.dart, що є частиною проєкту з розробки мобільного застосунку для перегляду інформації про героїв гри Dota 2. Цей файл відповідає за побудову інтерфейсу сторінки зі списком героїв, реалізованої з використанням патерну BLoC.

На початку файлу імпортуються необхідні пакети (рис. 3.8). Системні імпорти flutter/material.dart та flutter/services.dart використовуються для побудови інтерфейсу та обмеження орієнтації екрана. Імпорти, пов’язані з flutter_bloc, heroes_bloc.dart, heroes_event.dart та heroes_state.dart, залучають механізм станів і подій BLoC, який керує логікою відображення списку героїв. Також підключаються моделі, репозиторій даних і маршрути переходів.

Після цього оголошуємо віджет сторінки (рис. 3.8). Клас ListHeroesView є StatefulWidget, оскільки сторінка потребує змінного стану (наприклад, зміна списку героїв або результатів пошуку). Для керування цим станом реалізується відповідний State – _ListHeroesViewState.

```dart
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:new_project/data/models/heroesResponseDTO.dart';
import 'package:new_project/presentation/allHeroesPage/bloc/heroes_bloc.dart';
import 'package:new_project/presentation/allHeroesPage/bloc/heroes_event.dart';
import 'package:new_project/presentation/allHeroesPage/bloc/heroes_state.dart';
import 'package:new_project/presentation/form_submission_status.dart';

import 'package:new_project/data/repository/dota_repository.dart';
import '../routes/appRoutes.dart';

class ListHeroesView extends StatefulWidget {
  const ListHeroesView({Key? key}) : super(key: key);

  @override
  State<ListHeroesView> createState() => _ListHeroesViewState();
}
```

Рисунок 3.8 – Імпорт необхідних пакетів та оголошення віджету сторінки

Згодом створюємо станову логіку, яким є клас _ListHeroesViewState (рис. 3.9). У мобільному застосунку на Flutter, що базується на реактивній архітектурі, для забезпечення динамічного оновлення інтерфейсу користувача доцільно застосовувати станоорієнтовані компоненти. У межах StatefulWidget, яким у цьому випадку є ListHeroesView, створюється відповідний клас стану _ListHeroesViewState. Саме він відповідає за збереження поточного стану інтерфейсу, а також за ініціалізацію та оновлення логіки взаємодії з бізнес-компонентом.

Функціональне призначення основних змінних та компонентів класу _ListHeroesViewState полягає в організації локального стану інтерфейсу сторінки відображення героїв. Зокрема, змінна listHeroes використовується як локальне сховище для збереження списку героїв, отриманого з бізнес-логіки через BLoC-компонент. Вона також оновлюється під час виконання пошуку та є основним джерелом даних для побудови елементів списку на екрані.

Допоміжна булева змінна flag, хоча у наданому фрагменті явно не реалізує своєї логіки, потенційно передбачена для контролю одноразового виконання певних дій, наприклад, уникнення повторного завантаження героїв при ререндері.

Контролер nameController є інструментом управління введенням користувача в текстовому полі пошуку. Його завдання – забезпечити зв’язок між введеним текстом та механізмом фільтрації списку, що реалізується шляхом виклику відповідної події у BLoC (GetSearchListHeroes). Таким чином, контролер виступає у ролі посередника між користувацьким введенням та оновленням стану інтерфейсу.

У методі build, що є обов’язковим для будь-якого класу стану у Flutter, встановлюється портретна орієнтація екрана за допомогою методу SystemChrome.setPreferredOrientations. Це покращує користувацький досвід, забезпечуючи консистентність відображення інтерфейсу на різних пристроях.

Далі повертається віджет Scaffold, що слугує базовою структурною оболонкою сторінки. Основним вмістом body є віджет SafeArea, який запобігає перекриттю контенту системними елементами (напр. вирізами, панеллю стану). Всередині SafeArea ініціалізується BlocProvider, який створює та надає у контекст компонент HeroBloc, що реалізує бізнес-логіку взаємодії з репозиторієм. Вкладеним методом _heroesForm(context) будується основна форма взаємодії з користувачем.

```
20  class _ListHeroesViewState extends State<ListHeroesView> {
21    List<Heroes> listHeroes = [];
22
23    bool flag = false;
24
25    final nameController = TextEditingController();
26
27    @override
28    Widget build(BuildContext context) {
29      SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
30      return Scaffold(
31        body: SafeArea(
32          child: BlocProvider(
33            create: (context) => HeroBloc(dotaRepository: context.read<DotaRepository>()),
34            child: _heroesForm(context),
35          ),  // BlocProvider
36        )  // SafeArea
37      );  // Scaffold
38    }
39
```

Рисунок 3.9 – Початок реалізації класу _ListHeroesViewState та метод build

Також формуємо інтерфейс перегляду списку героїв (рис. 3.10 та рис. 3.11). Метод _heroesForm виконує ключову роль у побудові реактивного інтерфейсу сторінки перегляду героїв, реалізованого відповідно до принципів архітектурного шаблону BLoC (Business Logic Component). Зокрема, метод організовує динамічну взаємодію між бізнес-логікою та візуальним відображенням даних, використовуючи такі компоненти, як BlocListener та BlocBuilder.

Функціональне призначення складових:

– BlocListener<HeroBloc, HeroState> відповідає за реагування на зміни стану, які стосуються процесу подання форми. При отриманні стану SubmissionSuccess, тобто успішного завантаження даних, виконується оновлення локальної змінної listHeroes методом setState, що ініціює оновлення інтерфейсу користувача;

– BlocBuilder<HeroBloc, HeroState> безпосередньо відповідає за побудову вмісту інтерфейсу відповідно до поточного стану BLoC. На початковій ітерації компонента, якщо список героїв порожній, генерується подія LoadAllHeroes, яка ініціює асинхронне завантаження даних. Одночасно реалізується сортування списку за алфавітним порядком локалізованих назв героїв (nameLoc) та виклик події GetSearchListHeroes, яка здійснює фільтрацію за критерієм пошуку, введеним користувачем;

– візуально інтерфейс формується за допомогою контейнера з фоновим зображенням та структури Form, в яку вкладено два основні елементи: поле введення (TextField) з прив’язаним контролером nameController, і список героїв, побудований за допомогою методу _heroesDotaList.

У сукупності, метод забезпечує адаптивну реакцію на зміну стану, покращує взаємодію з користувачем завдяки миттєвому пошуку та забезпечує узгоджене відображення структурованих даних.

```
40      Widget _heroesForm(BuildContext context) {
41        return BlocListener<HeroBloc, HeroState>(
42            listener: (context, state) {
43            final formStatus = state.formStatus;
44
45            if (formStatus is FormSubmitting) {
46              const CircularProgressIndicator();
47            }
48
49            if (formStatus is SubmissionSuccess) {
50              setState(() {
51                listHeroes = state.heroList;
52              });
53            }
54          },
55          child: BlocBuilder<HeroBloc, HeroState>(builder: (context, state) {
56            if (listHeroes.isEmpty) {
57              context.read<HeroBloc>().add(LoadAllHeroes());
58              flag = true;
59            }
```

Рисунок 3.10 – Перша частина методу _heroesForm

```
60
61          listHeroes.sort((a, b) => a.nameLoc!.toLowerCase().compareTo(b.nameLoc!.toLowerCase()));
62          context.read<HeroBloc>().add(GetSearchListHeroes(name: nameController.text));
63          return Container(
64              decoration: const BoxDecoration(
65                  image: DecorationImage(fit: BoxFit.fill, image: AssetImage('assets/background_dota.png'))),  // BoxDecoration
66              child: Form(
67                  child: Center(
68                      child: Column(
69                children: [
70                  Container(
71                    height: 40,
72                    width: MediaQuery.of(context).size.width,
73                    padding: const EdgeInsets.symmetric(vertical: 5, horizontal: 10),
74                    child: TextField(
75                      controller: nameController,
76                    ),  // TextField
77                  ),  // Container
78                  _heroesDotaList(listHeroes),
79
80                ],
81              ))));  // Column, Center, Form, Container
82        }),  // BlocBuilder
83      );  // BlocListener
84    }
```

Рисунок 3.11 – Кінець реалізації методу _heroesForm

Після цього будуємо список героїв (рис. 3.11 та рис. 3.12). Метод _heroesDotaList відповідає за візуалізацію списку героїв, отриманого із стану BLoC. Він реалізований за допомогою віджета ListView.builder, який забезпечує ефективне побудування списку з динамічною кількістю елементів. Це є особливо актуальним при роботі з великими наборами даних, оскільки дозволяє створювати лише ті елементи, які наразі відображаються на екрані.

Кожен елемент списку побудований з використанням віджета ListTile, який вміщує зображення героя та його локалізовану назву. Зображення підвантажується з зовнішнього джерела на основі унікального ідентифікатора героя (формується з name, шляхом видалення префіксу npc_dota_hero_). Також реалізована навігація на екран з детальною інформацією про героя за допомогою Navigator.pushNamed, де передається відповідний ідентифікатор героя як аргумент.

Список обгорнутий у контейнер із власною стилізацією (градієнт, рамки, скруглення кутів), що покращує естетичне сприйняття інтерфейсу.

```
86  Widget _heroesDotaList(List<Heroes> listHeroes2) {
87    return Container(
88        height: MediaQuery.of(context).size.height - 100,
89        child: ListView.builder(
90          scrollDirection: Axis.vertical,
91          padding: const EdgeInsets.symmetric(horizontal: 15, vertical: 15),
92          itemCount: listHeroes2.length,
93          itemBuilder: (BuildContext context, int index) {
94            return Container(
95                decoration: BoxDecoration(
96                    borderRadius: BorderRadius.circular(20),
97                    border: Border.all(color: Colors.black54),
98                    gradient: const LinearGradient(begin: Alignment.topLeft, end: Alignment.bottomRight, colors: [
99                      Colors.grey,
100                     Colors.white60,
101                     Colors.white10,
102                   ])),  // LinearGradient, BoxDecoration
103               margin: const EdgeInsets.symmetric(vertical: 3),
104               child: ListTile(
105                   onTap: () {
106                     Navigator.pushNamed(context, AppRoutes.heroInfo, arguments: listHeroes2[index].id);
107                   },
108                   title: Row(
109                     children: [
110                       Image.network(
111                         'https://cdn.cloudflare.steamstatic.com/apps/dota2/images/dota_react/heroes/'
112                             '${listHeroes2[index].name!.replaceAll('npc_dota_hero_', '')}.png',
113                         height: 30,
114                         width: 50,
115                       ),  // Image.network
```

Рисунок 3.12 – Кінець реалізації методу _heroesDotaList

```
                        const SizedBox(
                          width: 10,
                        ),  // SizedBox
                        Text(
                          listHeroes2[index].nameLoc!,
                          style:
                              const TextStyle(fontSize: 18, fontFamily: 'Times New Roman', fontWeight: FontWeight.w600),
                        ),  // Text
                      ],
                    )));  // Row, ListTile, Container
            },
          ));  // ListView.builder, Container
    }
}
```

Рисунок 3.13 – Кінець реалізації методу _heroesDotaList

3.2.3 Створення екрану для перегляду детальної інформації обраного персонажа

Крок 1. Оголошення базових імпортів (рис. 3.14). На початку файлу здійснюється імпорт необхідних бібліотек Flutter (material.dart, cupertino.dart, services.dart), бібліотеки video_player (для можливого відтворення відео у майбутньому), а також компонентів BLoC-патерну, моделей даних (DotaHero, HeroAbilities), репозиторію DotaRepository і UI-компонентів (наприклад, LeadingAppBar, утиліт UrlUtils тощо).

Це забезпечує функціональну основу для взаємодії з API, обробки станів та відображення даних в інтерфейсі користувача.

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:video_player/video_player.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:new_project/data/models/heroAbilities.dart';
import 'package:new_project/data/models/heroInfoDTO.dart';
import 'package:new_project/data/repository/dota_repository.dart';
import 'package:new_project/presentation/appBars/leading_appbar.dart';
import 'package:new_project/presentation/form_submission_status.dart';
import 'package:new_project/presentation/heroInfo/bloc/hero_info_bloc.dart';
import 'package:new_project/presentation/heroInfo/bloc/hero_info_event.dart';
import 'package:new_project/presentation/heroInfo/bloc/hero_info_state.dart';
import 'package:new_project/presentation/utils/UrlUtils.dart';
```

Рисунок 3.14 – Оголошення базових імпортів

Крок 2. Створення головного класу сторінки (рис. 3.15). Клас HeroView є віджетом типу StatefulWidget, що дозволяє динамічно змінювати вміст сторінки в залежності від стану. У методі createState() створюється екземпляр стану _HeroViewState, у якому і відбувається основна логіка.

```
17  class HeroView extends StatefulWidget {
18    const HeroView({Key? key}) : super(key: key);
19
20    @override
21    State<HeroView> createState() => _HeroViewState();
22  }
```

Рисунок 3.15 – Створення головного класу сторінки

Крок 3. Клас стану _HeroViewState (рис. 3.16). У класі зберігається ідентифікатор героя (id), що передається з попередньої сторінки, та об’єкт myHero, що містить повну інформацію про обраного героя. Він буде використовуватись для побудови інтерфейсу після завантаження.

```
24  class _HeroViewState extends State<HeroView> {
25    var id;
26
27    DotaHero? myHero;
```

Рисунок 3.16 – Клас стану _HeroViewState

Крок 4. Метод build – побудова UI (рис. 3.17). У методі build() відбувається:

- обмеження орієнтації екрана до портретної;
- отримання аргументу id, що передається через Navigator зі сторінки списку героїв;
- BlocProvider створює новий екземпляр HeroInfoBloc і передає йому залежність DotaRepository;

– далі викликається приватний метод _heroInfoForm(context), який формує інтерфейс сторінки на основі стану.

```
@override
Widget build(BuildContext context) {
  SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
  final args = ModalRoute.of(context)!.settings.arguments as int;
  id = args;
  return Scaffold(
      body: SafeArea(
    child: BlocProvider(
      create: (context) => HeroInfoBloc(dotaRepository: RepositoryProvider.of<DotaRepository>(context)),
      //HeroInfoBloc(dotaRepository: context.read<DotaRepository>()),
      child: _heroInfoForm(context),
    ),  // BlocProvider
  ));  // SafeArea, Scaffold
}
```

Рисунок 3.17 – Метод build

Крок 5. Метод _heroInfoForm – логіка реакції на стан (рис. 3.18). Цей метод реалізує два основні блоки: BlocListener та BlocBuilder

Блок BlocListener (рис. 3.18):

– відслідковує зміну стану форми;

– якщо стан – SubmissionSuccess, у змінну myHero записується завантажений об’єкт героя, що ініціює оновлення інтерфейсу через setState.

```
Widget _heroInfoForm(BuildContext context) {
  return BlocListener<HeroInfoBloc, HeroInfoState>(listener: (context, state) {
    final formStatus = state.formStatus;

    if (formStatus is FormSubmitting) {
      const CircularProgressIndicator();
    }

    if (formStatus is SubmissionSuccess) {
      setState(() {
        myHero = state.hero;
      });
    }
```

Рисунок 3.18 – Метод _heroInfoForm та блок BlocListener

Блок BlocBuilder (рис. 3.19):

– при кожному побудованому фреймі в BLoC надсилається подія LoadingScreen(id) для завантаження героя з певним ID;

– якщо myHero == null, відображається індикатор завантаження;

– у випадку успішного завантаження будується інтерфейс із характеристиками.

Основне тіло сторінки героя (рис. 3.19):

– відображається кастомний AppBar з ім’ям та основним атрибутом героя;

– нижче йдуть секції, які детально описують характеристики, статистику та здібності героя.

```
}, child: BlocBuilder<HeroInfoBloc, HeroInfoState>(builder: (context, state) {
  context.read<HeroInfoBloc>().add(LoadingScreen(id));
  if (myHero == null) {
    return Container(
      decoration: const BoxDecoration(
        image: DecorationImage(
          fit: BoxFit.fill,
          image: AssetImage('assets/background_dota.png'),
        ),  // DecorationImage
      ),  // BoxDecoration
      child: const Center(
        child: CircularProgressIndicator(),
      ),  // Center
    );  // Container
  } else {
    return Container(
        decoration: const BoxDecoration(
            gradient: LinearGradient(begin: Alignment.topLeft, end: Alignment.bottomRight, colors: [
          Colors.black87,
          Colors.black54,
        ])),  // LinearGradient, BoxDecoration
        child: Container(
          child: Column(
            mainAxisSize: MainAxisSize.max,
            children: [
              LeadingAppBar(myHero!.primaryAttr, myHero!.nameLoc),
              _attributesHero(),
              _statsHero(),
              _abilitiesHero(),
```

Рисунок 3.19 – Блок BlocBuilder та основне тіло сторінки героя

Крок 6. На цьому етапі формуються класи, які є складовими частинами сторінки перегляду повної інформації про героя. Основними елементами

інтерфейсу є методи _attributesHero(), _statsHero() та _abilitiesHero(), кожен з яких відповідає за побудову певної логічної секції на екрані. Розглянемо кожен з них детальніше.

Метод _attributesHero() (рис. 3.20) відповідає за відображення базових атрибутів героя гри Dota 2. Він структурно поділений на дві підсекції: _imageAndHealth() та _attributes().

```
94    Widget _attributesHero() {
95      return Container(
96          padding: const EdgeInsets.only(top: 10, left: 10, right: 10),
97          child: Column(
98            children: [
99              Row(
100               mainAxisSize: MainAxisSize.max,
101               children: [
102                 _imageAndHealth(),
103                 const SizedBox(
104                   width: 20,
105                 ),  // SizedBox
106                 _attributes()
107               ],
108             ),  // Row
109             const SizedBox(
110               height: 10,
111             ),  // SizedBox
112           ],
113       ));  // Column, Container
114   }
```

Рисунок 3.20 – Реалізація методу _attributesHero()

Метод _imageAndHealth() реалізує виведення зображення героя разом із візуальними блоками здоров’я та мани. Для цього використовуються кольорові блоки із заданими стилями, де відображаються значення максимального здоров’я, його регенерації, а також максимального значення мани і швидкості її відновлення. Віджет Visibility дозволяє контролювати наявність додаткової інформації, такої як швидкість регенерації, що

потенційно може бути прихована чи відображена в залежності від подальших функціональних змін.

Метод _attributes() виводить інформацію про основні характеристики героя – силу, спритність та інтелект. Кожен атрибут представлений як базове значення з приростом за рівень, що дозволяє користувачеві швидко оцінити динаміку розвитку героя. Крім цього, реалізовано виведення типу атаки у вигляді відповідного зображення, а також складності керування героєм , яка класифікується на три рівні: легкий, середній, складний.

Таким чином, метод _attributesHero() забезпечує візуалізацію базової інформації, що дає гравцеві швидке уявлення про базові особливості героя.

Метод _statsHero() (рис. 3.21) призначений для виведення розширеної статистики героя, яка має значення під час ігрового процесу. Усі дані групуються на три основні блоки: атака, захист та мобільність.

```
337   Widget _statsHero() {
338     return Container(
339         padding: const EdgeInsets.symmetric(horizontal: 10),
340         child: Column(
341           children: [
342             const Text(
343               'STATS',
344               style: TextStyle(fontSize: 25, color: Colors.white, fontWeight: FontWeight.bold),
345             ),  // Text
346             IntrinsicHeight(
347                 child: Row(
348               mainAxisAlignment: MainAxisAlignment.spaceBetween,
349               crossAxisAlignment: CrossAxisAlignment.stretch,
350               mainAxisSize: MainAxisSize.max,
351               children: [_attackHero(), _defenseHero(), _mobilityHero()],
352             )),  // Row, IntrinsicHeight
353           ],
354         ));  // Column, Container
355   }
```

Рисунок 3.21 – Реалізація методу _statsHero()

До групи атаки відносяться параметри: швидкість атаки , затримка перед нанесенням шкоди, діапазон шкоди та радіус атаки. Це дозволяє користувачеві оцінити ефективність героя у ближньому або дальньому бою.

Група захисту включає значення броні та опору до магії, які критично важливі для виживання героя у командних сутичках.

Мобільність визначається через такі параметри, як швидкість руху, радіус видимості вдень і вночі, а також радіус повороту. Додатково відображається розмір колізії, що впливає на взаємодію героя з об’єктами карти.

Усі ці блоки реалізовані через Row із трьома Expanded елементами, що забезпечує рівномірний розподіл простору між групами статистики та візуально зручну подачу інформації.

Метод _abilitiesHero() (рис. 3.22) відповідає за виведення здібностей, які має конкретний герой. Список здібностей формується на основі об’єктів класу Ability, що передаються у вигляді колекції. Кожна здібність відображається у вигляді іконки, отриманої з мережі, що відповідає її унікальному візуальному представленню в грі.

```
550  Widget _abilitiesHero() {
551    return Container(
552      child: Column(
553        children: [
554          const Text(
555            'ABILITIES',
556            style: const TextStyle(fontSize: 25, color: Colors.white, fontWeight: FontWeight.bold),
557          ),  // Text
558          _iconsAbilities(),
559        ],
560      ),  // Column
561    );  // Container
562  }
```

Рисунок 3.22 – Реалізація методу _abilitiesHero()

Якщо герой не має визначених здібностей або їх завантаження не відбулося, застосовується умова перевірки на наявність елементів у списку, і виводиться текстове повідомлення про відсутність здібностей. Таким чином забезпечується стабільність інтерфейсу у випадках з неповними даними.

Наступним етапом є реалізація логіки відображення детальної інформації про здібності героя гри Dota 2. Основною метою є забезпечення

користувача зручним і структурованим способом перегляду ключових параметрів кожної здібності, включно з її назвою, відеодемонстрацією, описом, технічними характеристиками, перезарядкою та витратами мани.

Основним методом, що викликає відповідне вікно з інформацією, є _abilityInfo() (рис. 3.23 та рис. 3.24). У цьому методі ініціалізується модальне діалогове вікно типу AlertDialog, яке адаптується до розмірів екрана пристрою та заповнюється контентом про відповідну здібність. Вікно містить кілька ключових компонентів: назву здібності, її відеозапис, опис, статистичні значення, а також блоки з інформацією про перезарядку та витрати мани. Структура вікна побудована з використанням віджетів Column, що дозволяє вертикально компонувати елементи інтерфейсу.

```
589   _abilityInfo(Ability ability, String urlAbilityVideo) {
590     double widthPhone = MediaQuery.of(context).size.width;
591     double heightPhone = MediaQuery.of(context).size.height;
592     showDialog(
593         barrierDismissible: true,
594         context: context,
595         builder: (context) {
596           return AlertDialog(
597               contentPadding: const EdgeInsets.all(0),
598               content: Container(
599                 width: widthPhone,
600                 color: Colors.black,
601                 child: Column(
602                   mainAxisSize: MainAxisSize.max,
603                   children: [
604                     Container(
605                       child: Column(
606                         children: [
607                           _abilityName(ability),
608                           _abilityVideo(ability, urlAbilityVideo),
609                           Container(
610                             padding: const EdgeInsets.only(left: 5, right: 5, top: 5),
611                             child: Text(
612                               ability.descLoc,
613                               style: const TextStyle(fontSize: 16, color: Colors.white),
614                               textAlign: TextAlign.center,
615                             ),  // Text
616                           ),  // Container
617                         ],
```

Рисунок 3.23 – Перша частина реалізації методу _abilityInfo()

```
618                     ),  // Column
619                   ),  // Container
620                   SizedBox(
621                     height: 20,
622                   ),  // SizedBox
623                   Container(
624                     child: _abilityStats(ability),
625                   ),  // Container
626                   SizedBox(
627                     height: 20,
628                   ),  // SizedBox
629                   Container(
630                     child: Column(
631                       children: [
632                         _abilityCooldowns(ability),
633                         _abilityManacost(ability),
634                       ],
635                     ),  // Column
636                   ),  // Container
637                 ],
638               ),  // Column
639             ));  // Container, AlertDialog
640         }).then((val) {
641       _videoPlayerController.pause();
642     });
643   }
```

Рисунок 3.23 – Друга частина реалізації методу _abilityInfo()

Для відображення назви здібності використовується метод _abilityName(), який повертає стилізований контейнер з текстовим відображенням локалізованої назви здібності. Візуально елемент підкреслюється градієнтним фоном і великим шрифтом, що виділяє його серед інших компонентів.

Метод _abilityVideo() відповідає за інтеграцію відеоплеєра у вікно здібності. Відео відтворюється з використанням пакету video_player, а його ініціалізація та запуск виконуються в асинхронному режимі. Контролер _videoPlayerController зберігається у стані класу, що дозволяє зупиняти відео після закриття діалогу через метод dispose().

Опис здібності виводиться безпосередньо в методі _abilityInfo() у вигляді текстового поля під відео. Для альтернативного компактного

представлення всієї інформації про здібність реалізовано метод _abilityMechanic(), який об’єднує опис, характеристики, перезарядку та манакост у єдиний вертикальний блок. Він може використовуватися в інших контекстах відображення, де потрібен стислий формат.

Метод _abilityStats() обробляє список об’єктів SpecialValue, що містять конкретні числові значення здібності. Він виконує перевірку кожного об’єкта на наявність заголовка (headingLoc) та значень, більших за нуль. Після обробки всі значення форматуються та виводяться окремими текстовими полями. Таким чином забезпечується динамічне формування списку характеристик для кожної здібності.

Методи _abilityCooldowns() та _abilityManacost() працюють за схожим принципом. Вони підраховують суму всіх значень відповідно для перезарядки та витрат мани, а також формують текстову стрічку з цими значеннями, розділеними символом «/». У випадку, якщо сума є нульовою, відповідні блоки не виводяться, що дозволяє уникнути зайвого інформаційного шуму. Окремі графічні елементи, такі як піктограми або градієнтні блоки, використовуються для візуального розділення інформації.

Таким чином, реалізовані методи формують гнучку, масштабовану та інформативну систему представлення здібностей героїв, що значно підвищує зручність взаємодії користувача із застосунком. Компонентність підходу дозволяє ефективно повторно використовувати окремі елементи інтерфейсу в різних частинах застосунку.

3.3 Тестування розробленої моделі

Для перевірки працездатності та коректності функціонування мобільного застосунку було проведено серію функціональних тестів, які охоплюють основні сценарії взаємодії користувача з інтерфейсом. Нижче наведено результати тестування ключових екранів і компонентів системи.

На головному екрані застосунку (рис. 3.24) реалізовано відображення повного списку доступних героїв гри Dota 2 з пошуковим рядком. Для кожного героя виводиться його назва та зображення. Інформація підтягується з API та відображається у форматі динамічного списку з прокруткою.

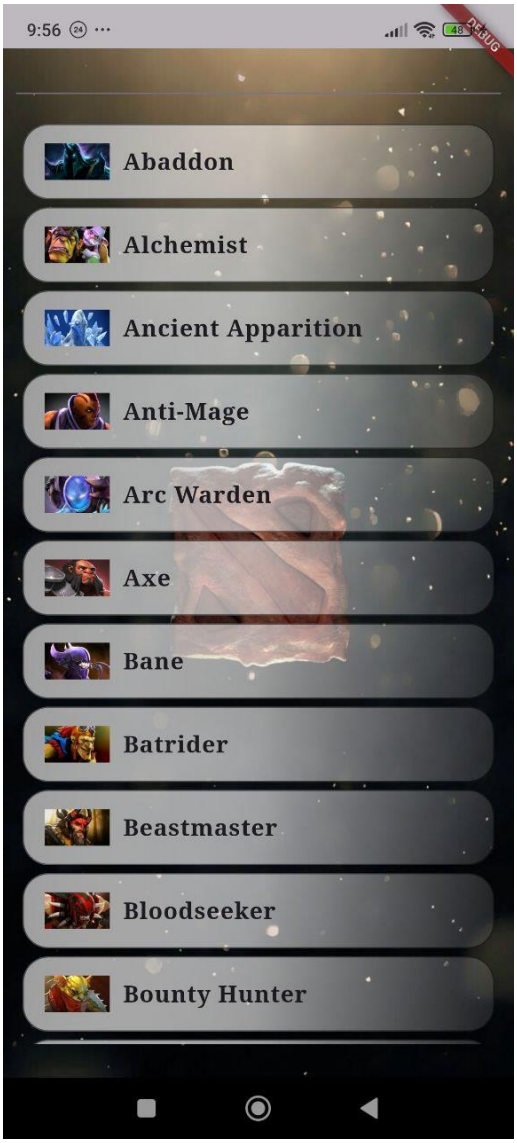

Рисунок 3.24 – Головний екран застосунку

Для зручності користувача реалізовано механізм пошуку (рис. 3.25), який дозволяє швидко знаходити потрібного героя за ім’ям. При введенні пошукового запиту список героїв фільтрується в реальному часі, відображаючи лише ті результати, які відповідають введеному тексту.

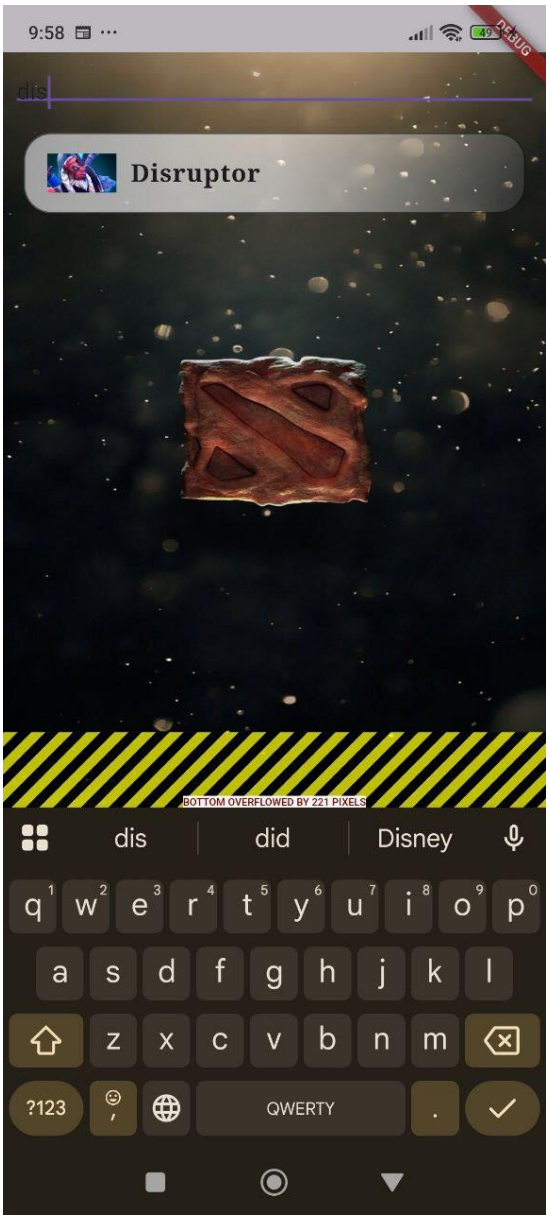

Рисунок 3.25 – Механізм пошуку

При натисканні на будь-якого героя зі списку користувач переходить на сторінку детальної інформації (рис. 3.26). Тут відображаються основні характеристики героя, такі як сила, спритність, інтелект, кількість здоров’я, а також список здібностей з коротким описом кожної.

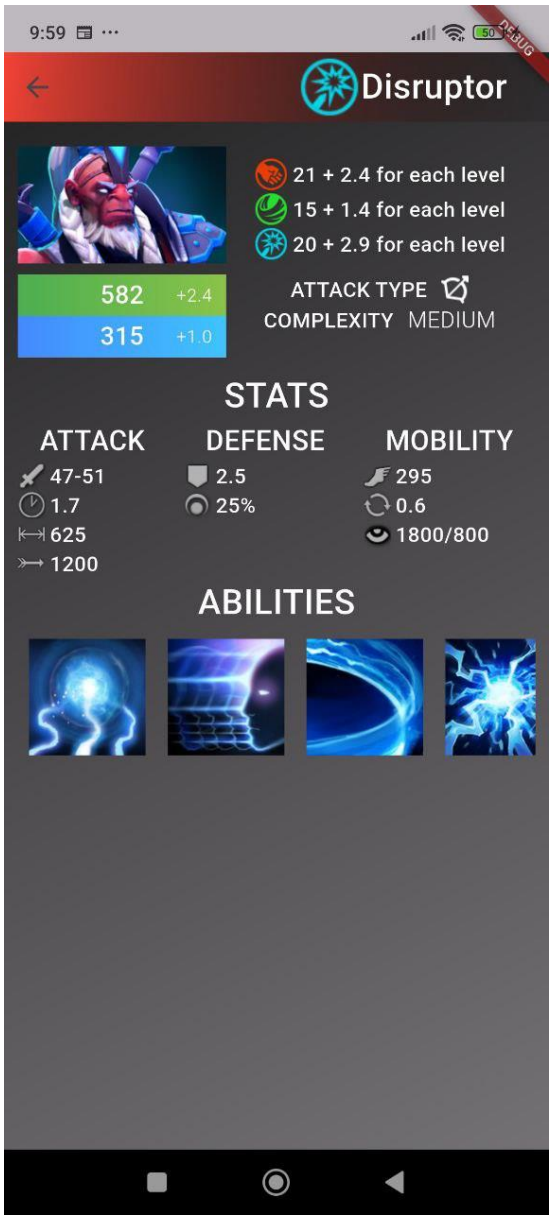

Рисунок 3.26 – Сторінка героя

При натисканні на одну із здібностей відкривається модальне вікно (рис. 3.27), в якому представлено відео з демонстрацією дії цієї здібності, її назву, опис, технічні характеристики, час перезарядки та витрати мани. Це дозволяє користувачеві ознайомитися з механікою здібності без переходу на зовнішні джерела.

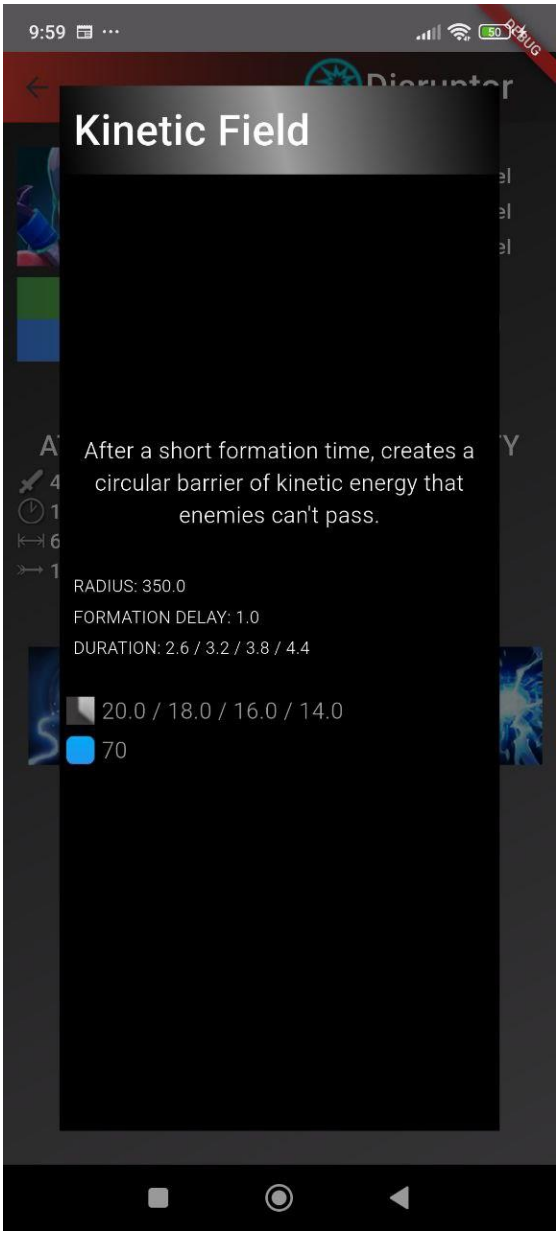

Рисунок 3.27 – Вікно здібності героя

Таким чином, тестування підтвердило правильність роботи застосунку відповідно до заданих функціональних вимог. Майже всі компоненти відображаються коректно, а користувацький інтерфейс забезпечує інтуїтивно зрозумілу навігацію. Отримані результати свідчать про готовність застосунку до подальшого використання.

ВИСНОВКИ

У межах кваліфікаційної роботи спроєктовано та реалізовано мобільний застосунок для перегляду детальної інформації про героїв гри Dota 2. Застосунок розроблено з використанням сучасного кросплатформного фреймворку Flutter, що дозволяє запускати його як на Android-, так і на iOS-пристроях з єдиною кодовою базою.

В процесі реалізації побудовано зручний та інтуїтивно зрозумілий інтерфейс, який забезпечує легкий доступ до інформації про кожного героя, включаючи його характеристики, здібності, здоров’я, а також іконки та описи. Вся інформація динамічно отримується з відкритого API OpenDota, обробляється у застосунку та при потребі кешується локально з використанням SQLite (sqflite).

Для ефективного управління станом було використано архітектуру BLoC, яка забезпечила чітке розділення бізнес-логіки від інтерфейсу користувача, що в результаті сприяло кращій підтримуваності, тестованості та масштабованості коду.

Уся структура застосунку реалізована з урахуванням сучасних принципів розробки програмного забезпечення, включаючи поділ на шари, що полегшує подальший розвиток, модифікацію та додавання нових функцій.

У результаті виконання роботи досягнуто поставлену мету – створено якісний, функціональний та зручний мобільний застосунок, який дозволяє користувачам швидко отримувати інформацію про героїв гри Dota 2. Отримані результати можуть бути використані як основа для подальшого розширення функціоналу, зокрема додавання системи обраних героїв, фільтрів, локалізації або інтеграції з іншими сервісами, пов’язаними з Dota 2.

Таким чином, розроблений застосунок не лише демонструє можливості сучасних технологій у мобільній розробці, а й має потенціал для подальшого вдосконалення та практичного використання серед гравців, аналітиків та фанатів гри Dota 2.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Руденко Д.О., Колосок Е.В. (2021) Огляд можливостей Google Analytics для роботи з даними. 9 Міжнародна науково-практична конференція «Results of modern scientific research and development» (14-16 листопада, 2021) Barca Academy Publishing, Мадрид, Іспанія. 2021. С.162-165.

2. Tvoroshenko, I., & Andrieieva, A. (2021). Development of web applications for remote learning of English.

3. Mashtalir, S., Mashtalir, V., & Stolbovyi, M. (2018, August). Representative based clustering of long multivariate sequences with different lengths. In 2018 IEEE second international conference on Data Stream Mining & Processing (DSMP) (pp. 545-548). IEEE.

4. Kobylin, O. A., Vyskrebentseva, S. O., & Petrova, R. V. (2019). Обробка даних, що містять пропуски в задачах кластеризації. Системи управління, навігації та зв'язку. Збірник наукових праць, 5(57), 45-50.

5. Руденко Д.О., Бондар В.О. (2020). Огляд можливостей використання стратегій об'єктно-орієнтованого маппінгу для зіставлення сутностей при розробці web застосунків. Матеріали III Міжнародної науково-практичної конференції «Priority Directions of Science and Technology Development» Київ, Україна.с.377-380.

6. Shafronenko, A., Bodyanskiy, Y., Pliss, I., & Irina, K. (2021, September). Online Credibilistic Fuzzy Clustering Method Based on Cauchy Density Distribution Function. In 2021 11th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 704-707). IEEE.

7. Oleg, K., Sergii, M., & Mykhailo, S. (2017, October). Video clustering via multidimensional time-series analysis. In Proceedings of the 9th International Conference on Information Management and Engineering (pp. 60-63).

8. Руденко, Д., Лотвінова, В., & Безверха, Є. (2023). Навчання нейронної мережі в задачах обробки даних. Collection of scientific papers «SCIENTIA», (September 22, 2023; Singapore, Singapore), 94-95.

9. Руденко Д.О., Самородов В. К. (2024). Вплив та використання сучасних методів розробки засобів для психологічної допомоги. Матеріали XIX Міжнародної науково-практичної конференції «Modern trends are the driving force of scientific progress» Лісабон, Португалія, с.101-103.

10. Shafronenko, A., Dolotov, A., Bodyanskiy, Y., & Setlak, G. (2018, August). Fuzzy clustering of distorted observations based on optimal expansion using partial distances. In 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP) (pp. 327-330). IEEE.

11. Gorokhovatskyi V., Tvoroshenko I. (2023) Identification of visual objects by the search request. International scientific symposium «INTELLIGENT SOLUTIONS-S». Computational intelligence (results, problems and perspectives). Decision making theory: proceedings of the international symposium, September 28, 2023, Kyiv-Uzhorod, Ukraine, pp. 25-27.

12. Gorokhovatskyi V., Tvoroshenko I., and Yakovleva O. (2024) Transforming image descriptions as a set of descriptors to construct classification features, Indonesian Journal of Electrical Engineering and Computer Science, 33(1), pp. 113-125.

13. Tvoroshenko I., Gorokhovatskyi V., Kobylin O., and Tvoroshenko A. (2023) Application of deep learning methods for recognizing and classifying culinary dishes in images, International Journal of Academic and Applied Research, 7(9), pp. 57-70.

14. O. Kobylin, V. Gorokhovatskyi, I. Tvoroshenko, O. Peredrii (2020). The Application of Non-Parametric Statistics Methods in Image Classifiers Based on Structural Description Components, Telecommunications and Radio Engineering, 79 (10), pp. 855-863.

15. Dmitry Kinoshenko, Sergey Mashtalir, Andreas Stephan, Vladimir Vinarski (1993). Neural Network Segmentation Of Video Via Time Series Analysis, INFORMATION THEORIES & APPLICATIONS, pp. 232.

16. Flutter BLoC Pattern. URL: https://bloclibrary.dev/#/ (дата звернення 01.05.2025)

17. DataBase Floor. URL: https://pub.dev/packages/floor (дата звернення 16.04.2025)

18. A composable, Future-based library for making HTTP requests. URL: https://pub.dev/packages/http (дата звернення 19.04.2025)

19. Flutter. Official Documentation. URL: https://flutter.dev/docs (дата звернення 10.04.2025)

20. Dart Language Tour. URL: https://dart.dev/guides/language/language-tour (дата звернення 10.04.2025)

21. OpenDota API Documentation. URL: https://docs.opendota.com/ (дата звернення 25.04.2025)

22. SQLite plugin for Flutter. URL: https://pub.dev/packages/sqflite (дата звернення 25.04.2025)

23. Provider package for Flutter. URL: https://pub.dev/packages/provider (дата звернення 13.04.2025)

24. Clean Architecture в Flutter. URL: https://medium.com/flutter-community/flutter-clean-architecture-fe125e9859f5 (дата звернення 14.04.2025)

25. Dota 2 Heroes. URL: https://www.dota2.com/heroes (дата звернення 20.04.2025)

26. GitHub репозиторій Flutter Examples. URL: https://github.com/flutter/samples (дата звернення 02.05.2025)

27. Firebase for Flutter. URL: https://firebase.flutter.dev/ (дата звернення 17.04.2025)

28. Material Design для Flutter. URL: https://m3.material.io/develop/flutter (дата звернення 09.05.2025)

29. Інтеграція REST API у Flutter. URL: https://docs.flutter.dev/cookbook/networking/fetch-data (дата звернення 11.04.2025)

30. Flutter DevTools. URL: https://docs.flutter.dev/tools/devtools/overview (дата звернення 01.04.2025)