МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Мобільний додаток до онлайн гри з підвищеними вимогами до інформаційної безпеки»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Москаленко В.В.**

Студента групи КБ–71 **Мальцев А.В.**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 г.

ЗАВДАННЯ
до випускної роботи

Студента четвертого курсу, групи КБ-71 спеціальності "Кібербезпека" денної форми навчання Мальцева Андрія Віталійовича.

Тема: "Мобільний додаток до онлайн гри з підвищеними вимогами до інформаційної безпеки"

Затверджена наказом по СумДУ

№_____ від _____20__ г.

Зміст пояснювальної записки: 1) вступ; 2) аналіз існуючих рішень; 3) постановка задачі; 4) проектування додатку; 5) вибір інструментів; 6) програмна реалізація; 7) висновки; 8) список літератури.

Дата видачі завдання "_____"_____2021 г.

Керівник випускної роботи _____ Москаленко В.В.

Завдання приняв до виконання _____ Мальцев В.В.

РЕФЕРАТ

Записка: 81 стор., 47 рис., 4 табл., 1 додаток, 19 джерел.

Об'єкт дослідження – інформаційна технологія для пошуку та перегляду ігрових колод з телефону з підвищеними вимогами до інформаційної безпеки.

Мета роботи – розробка інформаційної технології підтримки прийняття рішень у вигляді мобільного додатку для пошуку та перегляду колод для онлайн-гри.

Методи дослідження – аналіз, декомпозиція, синтез та формалізації.

Результати — був розроблений повноцінний додаток для платформи «Android», який дозволяє шукати ігрові колоди з онлайн-гри «Харстоун» і додавати їх у свою колекцію. Також у роботі було продемонстровано, як можна захистити дані з бази даних завдяки використанню Database Rules та як розробити безпечний механізм автентифікації користувачів.

ПРОГРАМУВАННЯ, ACTIVITY, ANDROID, RULES,
NULLPOINTEREXCEPTION, NULL SAFETY, ANDROID STUDIO,
DATA CLASS, TOSTRING, EQUALS, HASHCODE, XML

ЗМІСТ

ВСТУП

Робота присвячена темі розробці андроїд-додатків. В даний час внаслідок збільшення доступності мобільного Інтернету спостерігається тенденція до використання мобільних додатків, у тому числі допоміжних програм для комп'ютерних чи мобільних ігор. Це дає можливість отримувати актуальну інформацію та новини в місцях, де немає доступу до комп'ютера – в кафе, автобусі, таксі, чи будь-якому іншому місці. Окрім того, сайти не вирішують проблему, адже вони не завжди адаптивні під мобільні пристрої і вони не можуть своєчасно інформувати користувача push-повідомленнями про новини. Тому більш зручним варіантом є впровадження мобільних додатків.

Також слід зазначити, що значущою проблемою у світі інформаційних технологій є безпека даних користувачів. У цій роботі я продемонструю, як захистити свою базу даних навіть якщо зловмисник має посилання та доступ до неї.

Для досягнення мети ми будемо аналізувати ринок мобільних додатків для пошуку колод до гри «Харстоун» та проектувати власний додаток. Метою даної роботи є написання такого додатку в програмі Android Studio.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд існуючих рішень

Для досягнення мети, викладеної у постановці задачі, потрібно розробити таку інформаційну технологію, яка буде задовольняти найбільшу кількість користувачів і яка буде найбільш затребувана серед ком`юніті. По-перше, проаналізуємо ринок існуючих технологій у інтернет-просторі, які дають змогу переглядати і копіювати колоди з гри «Hearthstone». Ця гра доступна на ПК (Windows та MacOS), iOS та Android, тому ми можемо розробити додаток на одну з цих платформ. Оглядаючи існуючі рішення, можна виділити дві популярні технології – веб-сайти та програми для мобільних пристроїв. Далі ми розглянемо найпопулярніші із існуючих сайтів та додатків на ці платформи, проаналізуємо їх за такими критеріями, як актуальність, популярність, зручність, підтримка додатку або сайту та можливість конкуренції з ними і зробимо висновок, в якому зробимо вибір цільової платформи [1] .

1.1.1 HsReplay.net (веб-сайт)

HSReplay – веб-сайт, розроблений компанією HearthSim у 2016 році, яка також розробила дек-трекер для збіру статистики особистих матчів. Цей веб-сайт є доволі популярним серед користувачів і має усі функції, необхідні для того, щоб переглянути та обрати найкращу колоду. Сайт має велику базу зіграних ігр і статистику вінрейту для кожної деки. Колоди додаються власниками сайту, оновляються щотижня і не можуть додаватися користувачами. Але через погану оптимізацію інтерфейсу сайт погано відображається на мобільних пристроях.

Користувачів за день: 265 000+

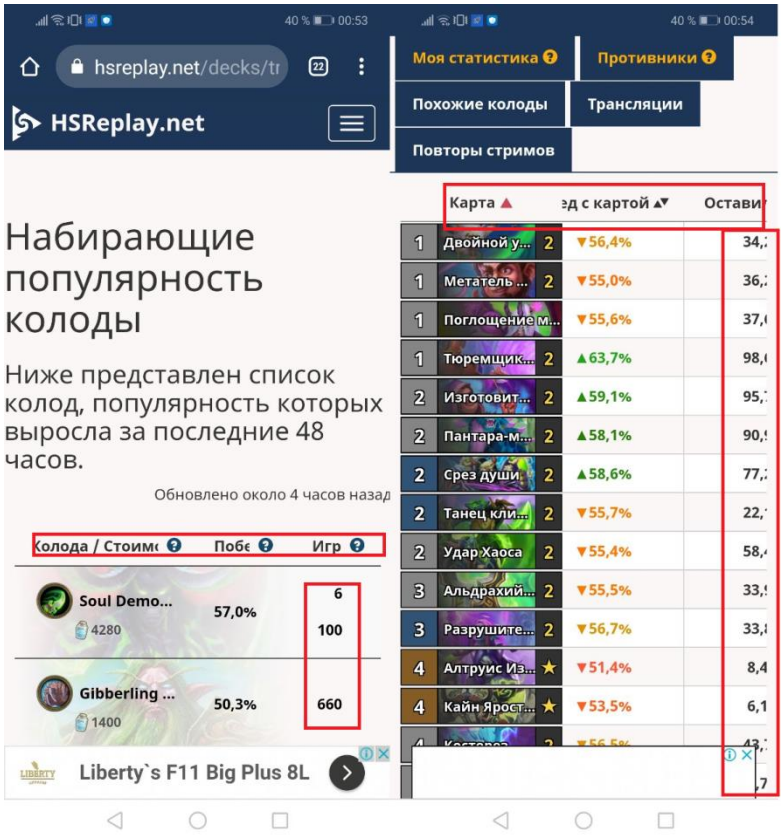


Рисунок 1 – Скріншоти веб-сайту HsReplay.net

Загальний огляд:

\+ велика аудиторія;

\+ статистика;

\+ регулярні оновлення колод від власників сайту;

– не адаптований під мобільні пристрої;

– важко конкурувати.

1.1.2 Hearthpwn.com (веб-сайт)

Створений у 2013 році, Hearthpwn є надзвичайно популярним серед шанувальників гри Hearthstone і має велике ком'юніті, яке спілкується на форумі сайту. Користувачі можуть створювати і додавати свої деки до сайту, а також оцінювати і коментувати деки інших людей. Завдяки відкритості сайту і можливості донести свої власні концепти колод до ком'юніті, цей сайт придбав прихильників зі всього світу. Нажаль, розробники сайту не поспішають

оновлювати дизайн, який майже не помінявся з 2015 року. І незважаючи на адаптованість під мобільні пристрої, важко не помітити недоліки та застарілий дизайн, який може бути причиною втрати популярності.

Користувачів за день: 125 000+

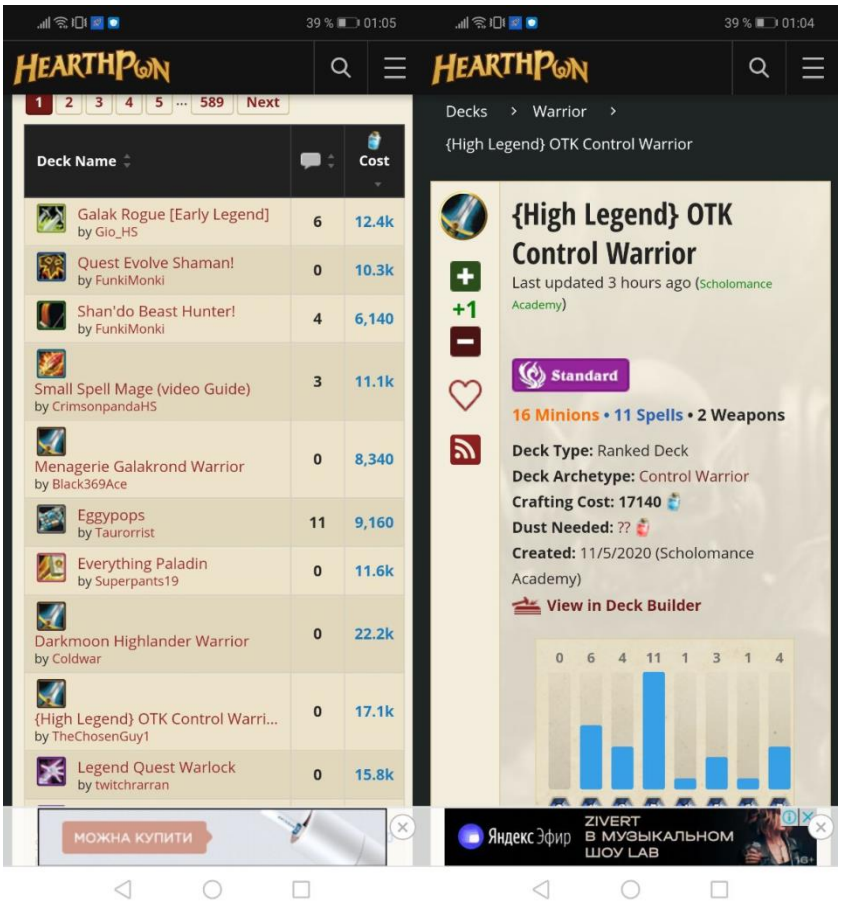


Рисунок 2 – Скріншоти веб-сайту Hearthpwn.com

Загальний огляд:

\+ велика аудиторія;

\+ широкий фільтр;

\+ користувацькі колоди;

– застарілий дизайн.

1.1.3 Deck Finder For HS (Android)

Додаток Deck Finder For HS реалізує можливості, яких вистачає середньому юзеру для знаходження та копіювання нових колод. Може відображать перелік колод, дивитися опис, копіювати колоди. Підтримує фільтр по широкому спектру параметрів та всім ігровим класам та має додаткову можливість зберігання колод до своєї колекції.

Останнє оновлення: 03.08.2020

Кількість завантажень: 100 000+

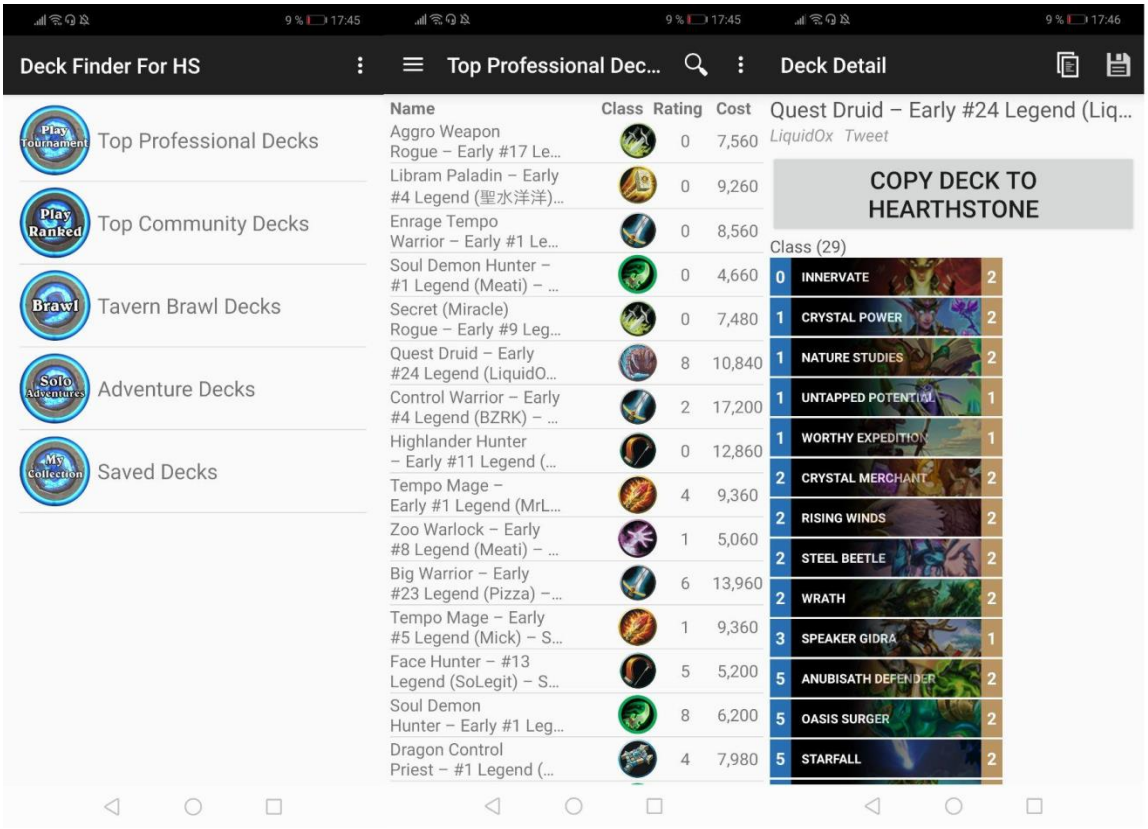


Рисунок 3 – Скріншоти додатку Deck Finder For HS

Загальний огляд:

\+ оновлення колод через інтернет;

\+ багато фільтрів;

\+ збереження колод для офлайн-перегляду;

– застарілий дизайн;

– немає перегляду карт;

– немає нагадувань (Push-повідомлення);

– повільне завантаження списку;

– обмежений функціонал (платний пошук, платне копіювання колод).

1.1.4 HSCollect (Android)

У додатку є функціонал для зберігання колод, але лише тих які ви створили, а також перегляду карт усіх доповнень. Воно не має функціоналу для перегляду переліку нових колод, та не має зв’язку з інтернетом.

Останнє оновлення: 04.08.2020

Кількість завантажень: 10 000+

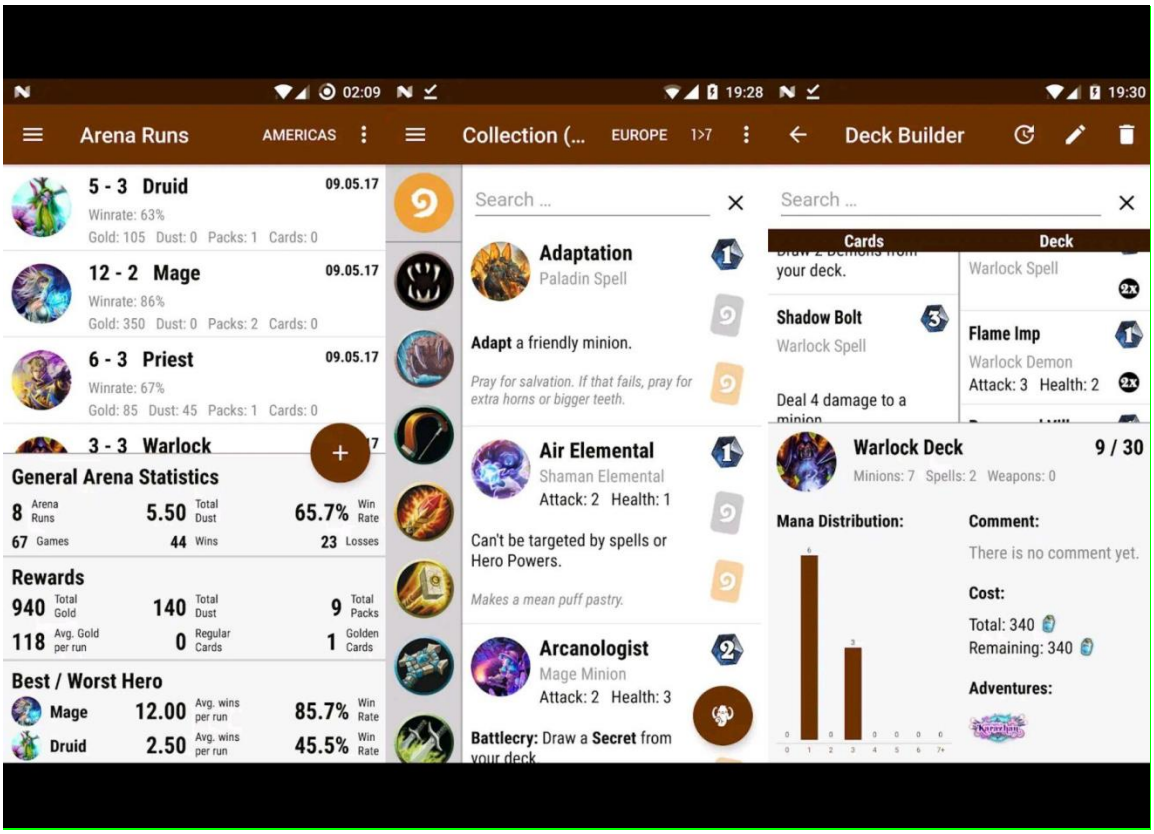


Рисунок 4 – Скріншоти додатку HSCollect

Загальний огляд:

\+ можна зберігати колоди;

\+ зручний інтерфейс;

\+ є бекап;

– застарілий дизайн;

– немає перегляду карт;

– немає пошуку по колодам;

– немає колод від ком’юніті.

1.1.5 DeckBox (Android)

Цей додаток, як і попередній, може створювати власні колоди та зберігати їх, але в ньому немає функціоналу для перегляду інших колод з інтернету.

Останнє оновлення: 04.08.2020

Кількість завантажень: 10 000+

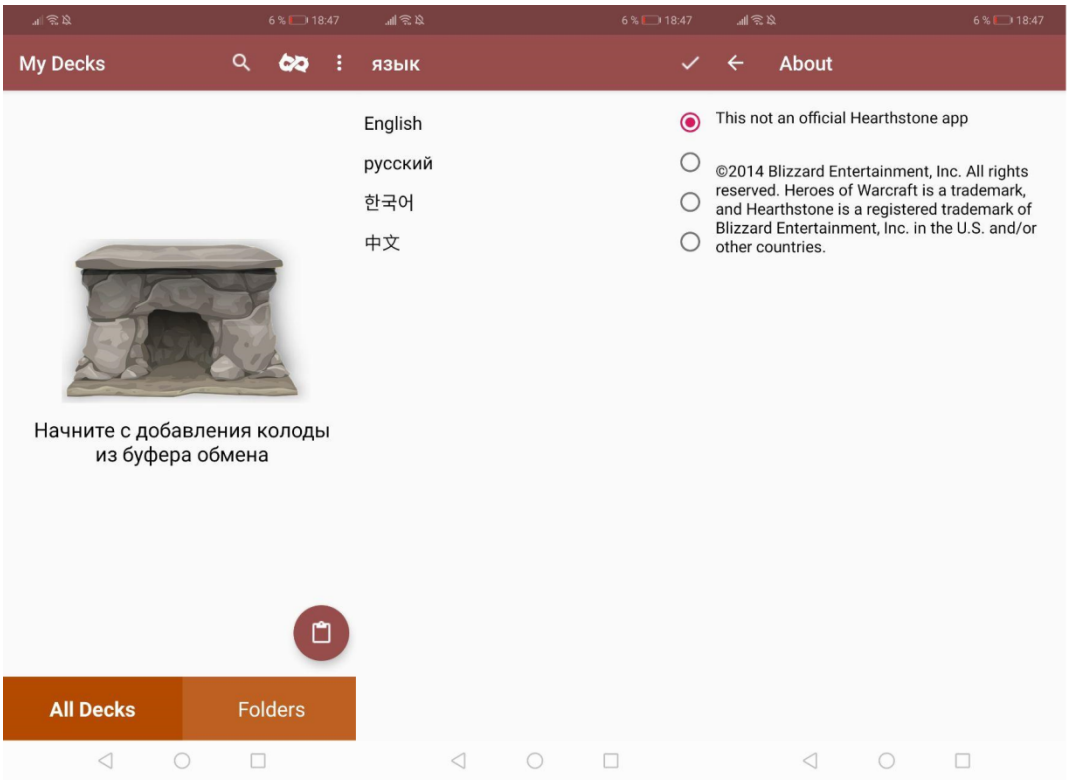


Рисунок 5 – Скріншоти додатку DeckBox

Загальний огляд:

\+ можна зберігати колоди;

\+ групування по папкам;

\+ є бекап;

– погана реалізація обирання мови;

– немає пошуку по колодам;

– не працює через інтернет;

– немає колод від ком’юніті.

1.1.6 Hearth Amino (Android, iOS)

Серед усіх переглянутих мною додатків, цей найближче схожий на додаток для перегляду користувацьких колод із інтернету. Але він має зайвий функціонал і звичайному юзеру, який хоче тільки подивитися колоди, він може здатися складним.

Останнє оновлення: 19.12.2019

Кількість завантажень: 10 000+

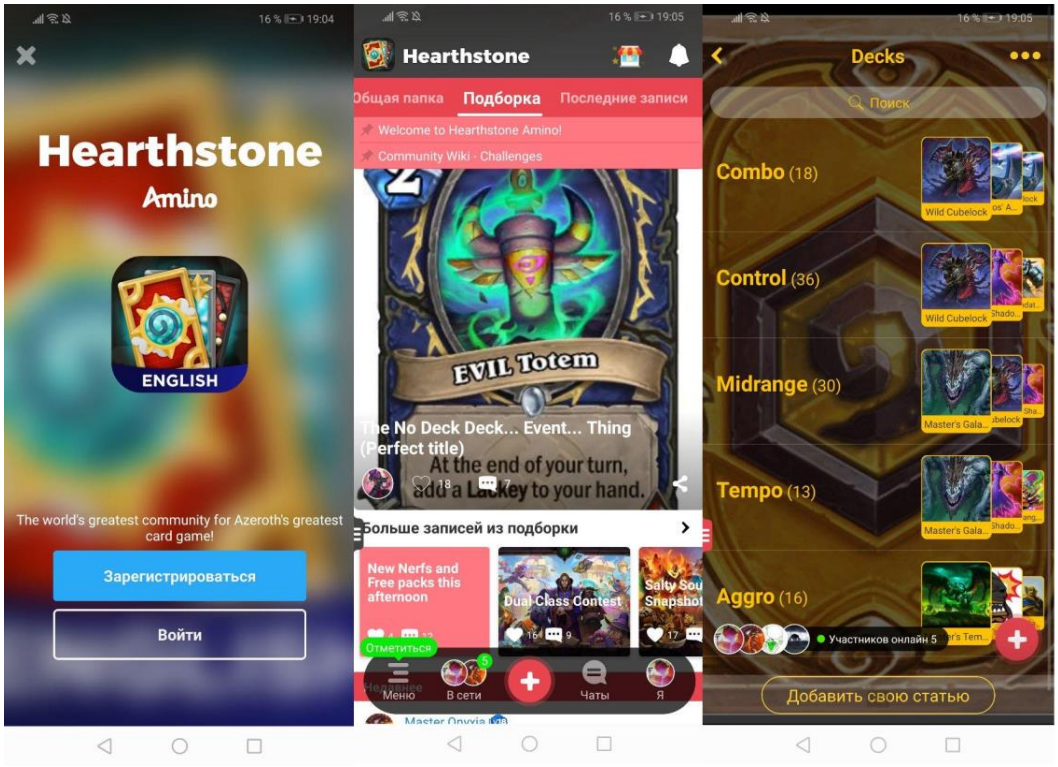


Рисунок 6 – Скріншоти додатку Hearth Amino

Загальний огляд:

+ можна зберігати колоди;
+ працює через інтернет;
+ є користувацькі колоди;
+ швидко працює та завантажує колоди;
– багато зайвого функціонау (це соціальна мережа);
– невелика аудиторія, тому в додатку мало колод;
– обов’язкова реєстрація;
– офіційно не підтримується з 2019 р.

1.2 Висновки до розділу 1.1

Отже, ми розглянули сайти та додатки на мобільні пристрої, які дозволяють знаходити нові колоди для гри. Проблема розробки сайта в тому, що вже існують монополісти у своїй сфері і новим проектам дуже важко створити їм конкуренцію. Один з сайтів має статистику для кожної деки та приватну статистику, інший сайт має форум і велике ком’юніті. Щодо мобільних пристроїв, ці додатки мають середню популярність, але з перелічених вище додатків і їх аналітики стає зрозуміло, що кожен з цих додатків має свої недоліки. Серед розглянутих додатків найбільш актуальним є **Deck Finder For HS** для платформи **Android**, який досі підтримується і оновляється на Google Play. Але в цьому додатці, як було зазначено вище, недостатній фукнціонал та застарілий дизайн. Важливо зазначити, що ця тематика є популярною – більша частина додатків має 100 000+ завантажень. Тому доцільним є рішення розробити власний додаток, який повинен мати зручний і інтуітивно зрозумілий інтерфейс і дизайн, впроваджуючий комфортну роботу з додатком. [1]

1.3 Постановка задачі

Необхідно підібрати інструменти для реалізації мобільного додатку для перегляду, пошуку та копіювання колод для гри «Харстоун» та спроектувати структуру додатку. Впровадження мобільного додатку в сферу взаємодії з колодами для гри необхідно для вирішення наступного спектру задач:

- перегляд існуючих колод;
- пошук нових колод;
- копіювання колод;
- збереження колод в колекцію;
- своєчасне отримання пуш-повідомлення про нові колоди.

Розробка та підтримка такого додатку має як недоліки, так і переваги. Зокрема з недоліків можна визначити необхідність постійної роботи сервера та підтримку додатку, виправлення багів. Але для користувачів використання такого додатку має чималу вигоду. Здебільшого маркетингові дослідження показують, що пошук інформації на офійіному джерелі через сторонні сервіси є ефективним впливом на користувача [2]. Також варто зазначити про перевагу використання додатку, спеціально розробленого під мобільні пристрої, над використанням веб-сайту, завдяки зручності і адаптивності.

Для досягнення мети потрібно розв'язати такі задачі:

- аналіз існуючих підходів щодо цієї теми;
- вибір цільової архітектури;
- вибір мови програмування;
- вибір середовища розробки;
- проектування базового дизайну;
- опис діаграми класів;

2 ПРОЕКТУВАННЯ

2.1 Варіанти використання системи

Діаграма варіантів використання представлена на рис.7.

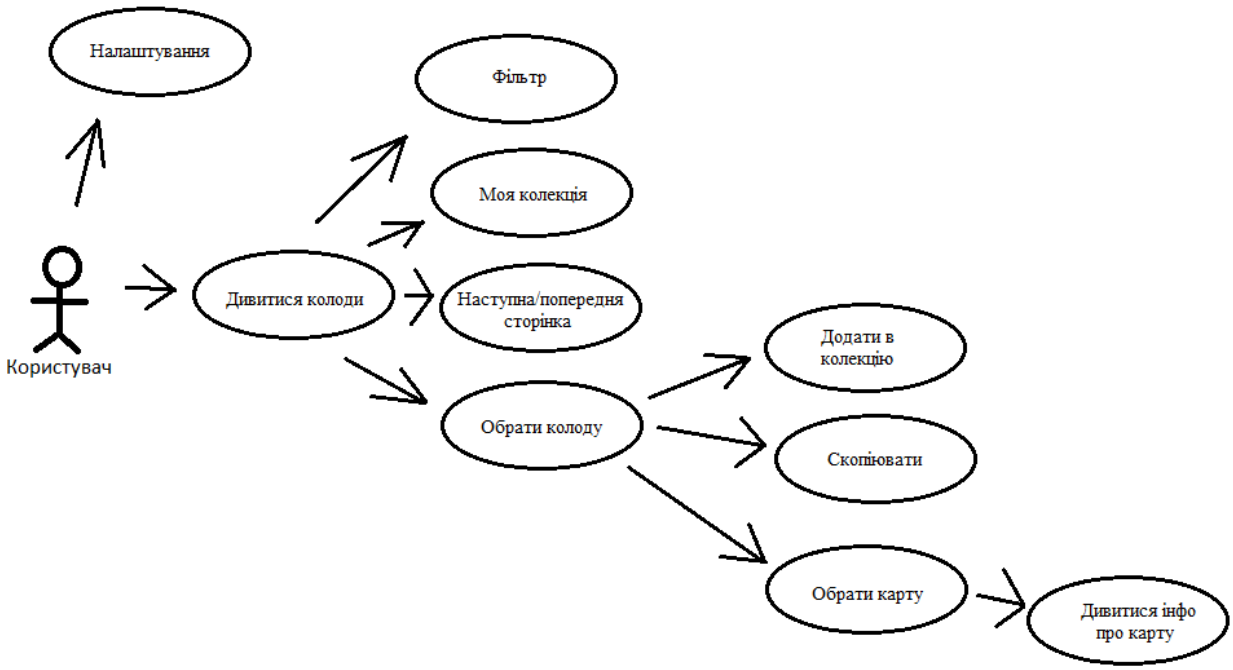


Рисунок 7 – Варіанти використання системи.

З додатком взаємодіє один актор (користувач) – це людина, яка використовує мобільний додаток. Користувач повинен мати можливість робити наступні дії:

– відкрити налаштування;

– дивитися список з колодами;

– застосувати фільтр пошуку (фільтр по класам/вартості колод/популярності або новизні);

– відкрити мою колекцію;

– гортати сторінку (наступна/попередня);

– обрати колоду зі списку;

– подивитися інформацію про колоду;

– додати колоду в колекцію;

– скопіювати код колоди;

– обрати карту (клікнути на неї);

– подивитися інформацію про карту.

2.2 Функціональні вимоги

Визначемо критерії, яким повинна задовольняти технологія і які вона повинна виконувати функції, щоб в подальшому нам було простіше визначити платформу та технології, які будемо використовувати в процесі розробки. Завданням додатку є реалізація найзручнішого спосібу перегляду, пошуку і копіювання колод.

Система повинна задовольняти наступним функціональним вимогам:

– відображати перелік колод;

– відображати перелік карт та мета-дані для колоди (автор, дата створення і т.д.) для кожної колоди;

– відображати арт-картинку для кожної карти;

– мати можливість фільтрувати колоди по ігровим класам та іншим параметрам;

– мати можливість додавати колоди у колекцію (зберігати для офлайн-доступу);

– оновляти список з колодами свайпом вниз;

– гортати сторінки з колодами (відображати макс. 10 колод на одній сторінці)

– своєчасно інформувати користувача про додання нових колод.

2.3 Інструменти реалізації

2.3.1 Платформа

По-перше, треба обрати ОС, на яку буде розроблятися додаток. Серед мобільних ОС будемо обирати серед iOS та Android. Щоб додаток став популярним, оберемо ту систему, на якій більша кількість потенційних користувачів. Звернемо увагу на додаток офіційної гри в офіційних магазинах та порівняємо кількість користувачів. Нажаль, App Store не надає інформації щодо кількості завантажень, тому покладемося на кількість відгуків.

App Store: 33 400 відгуків

Рисунок 8 – Додаток Hearthstone в App Store

Google Play: 1 717 425 відгуків

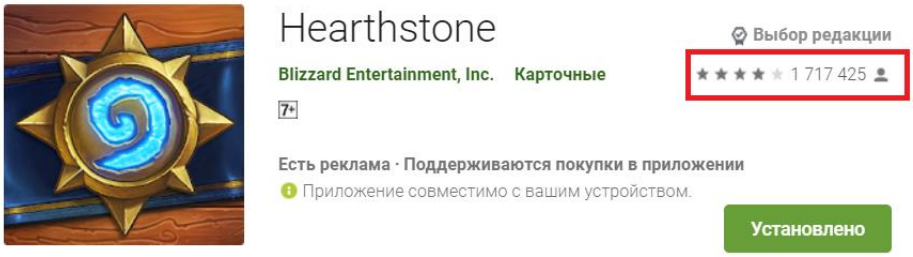


Рисунок 9 – Додаток Hearthstone в Google Play

На **iOS** ця гра є менш популярною, тому більш доцільно розроблювати додаток на платформу **Android**. Також андроїд виграє iOS за загальною популярністю серед користувачів [3]. Статистичні дані подані на рис.10.

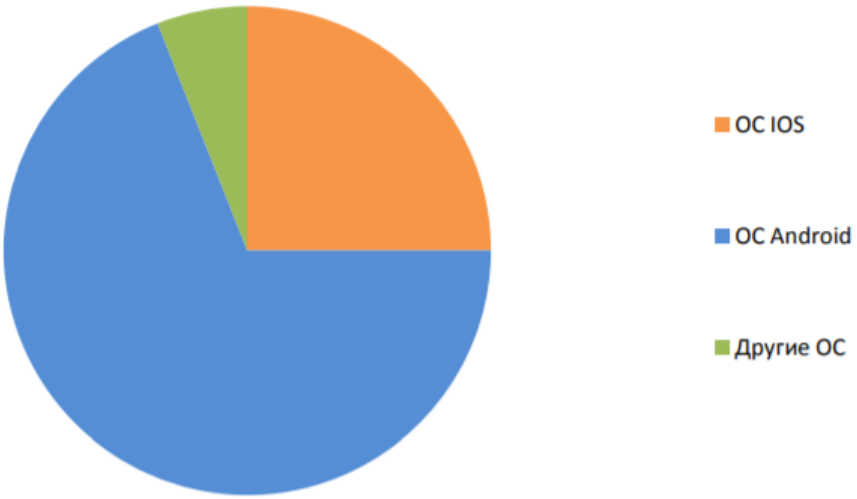


Рисунок 10 – Частка користувачів на різних ОС.

2.3.2 Мова програмування

Під андроїд існує дві основні мови програмування – Java та Kotlin.

Таблиця 1 – Опис мов програмування Java та Kotlin

Рисунок 11 – Логотип Java	***Java*** *— це об'єктно-орієнтована мова програмування високого рівня на основі класів, яка розроблена так, щоб мати якомога менше залежностей від реалізації. Це мова програмування загального призначення, призначена для того, щоб розробники програм могли писати один раз, запускати їх де завгодно, що означає, що скомпільований код Java може працювати на всіх платформах, що підтримують Java, без необхідності перекомпіляції.*
Рисунок 12 – Логотип Kotlin	***Kotlin*** *— це крос-платформна, статично типізована мова програмування загального призначення з виводом типу. Kotlin розроблений для повноцінної взаємодії з Java. Версія JVM стандартної бібліотеки Kotlin залежить від бібліотеки класів Java, але вивід типів дозволяє його синтаксису бути більш стислим.*

Щоб обрати більш кращу мову, слід звернути увагу на думку людей зі значним досвідом в цій сфері. Для прикладу був обраний ресурс «Хабрахабр» [4].

«Kotlin - безумовно, цікавий крок вперед, він скорочує час написання коду за рахунок більш коротких конструкцій.» (с) Дмитро Качалов, провідний програміст Kelly IT Solutions.

«Використовувати Kotlin простіше і зручніше, ніж Java. У ньому програмний код виходить в середньому на 40% коротше. А ще він дозволяє не допускати деяких помилок, які можуть виникнути в ході виконання програми. Коли код більш простий і зрозумілий, помилки складніше зробити і легше виявити, тому їх число стрімко знижується. Ви витрачаєте менше часу на розробку і тестування.» (с) Семен Пілунц, Android-розробник, експерт курсу Нетології.

Приклад синтаксису Kotlin у порівнянні з Java (табл. 2).

Таблиця 2 – Порівняння синтаксису Java та Kotlin

Java	**Kotlin**

`final TextView textView = (TextView) findViewById(R.id.text_view);`
 	
`textView.setText("text");`
	`textView.text = "text"`

`someList.filter({ text -> text.startsWith("aaa") })`	`someList.filter { it.startsWith("aaa") }`

`public final class User {`
`private String name;`
`private int age;`
`private int weight;`
 	
`public User(String name, int age, int weight) {`
	`data class User(`
	`val name: String,`
	`val age: Int,`
	`val height: Float = 1.8f)`

Продовження **Таблиці 2**

Java	Kotlin

this.name = name;
this.age = age;
this.weight = weight; }
 	
public Person(String name, int age) {
this.name = name;
this.age = age;
this.weight = 70; }
 	
public String getName() { return name; }
public int getAge() { return age; }
public int getWeight() { return weight; }	

У останньому прикладі - звичайний клас Java. Він не робить багато, він просто містить деякі дані. Він еквівалентний сусдньому прикладу на мові Kotlin, тому боляче бачити наскільки він великий. У Kotlin при оголошенні класу автоматично створюються необхідні функції геттерів, сетерів, copy(), toString(), hashcode() та equals() для цього класу даних. Звичайно, можно легко замінити ці функції, але в більшості випадків достатньо просто оголосити клас та його властивості. Однією з головних переваг Kotlin є його лаконічність. Ви отримуєте більше функціональних можливостей із меншим кодом. І чим менше коду ви пишете, тим менше помилок ви робите.

Null safety

Система типу Котліна спрямована на усунення небезпеки нульових посилань з коду, також відомого як помилка мільярда доларів.

Однією з найпоширеніших підводних каменів у багатьох мовах програмування, включаючи Java, є те, що доступ до члена нульового посилання призведе до винятку нульового посилання. У Java це буде еквівалент NullPointerException.

Система типу Kotlin спрямована на вилучення NullPointerException із нашого коду. Єдиними можливими причинами NullPointerException можуть бути:

– Явний виклик throw NullPointerException();

– Використання **!!** оператора, який описаний нижче;

– Неініціалізований this, який є в конструкторі, передається і десь використовується ("витік this");

– Конструктор суперкласу викликає відкритого члена, реалізація якого у похідному класі використовує неініціалізований стан;

У **Java** можуть бути ще такі додаткові причини:

– Спроби отримати доступ до члена за нульовим посиланням типу платформи;

– Загальні типи, що використовуються для взаємодії Java з неправильним дозволом, наприклад шматок коду Java може додати null до коду Kotlin MutableList <String>, що означає, що для роботи з ним слід використовувати MutableList <String?>;

– Інші проблеми, спричинені зовнішнім кодом Java.

Перевага Java лише у єдиному пункті – на даний момент, він у 5.7 разів популярніший, ніж Kotlin [5]. Це значить, що більша кількість бібліотек та окремих готових частин коду буде доступною в інтернеті.

Також на рисунку нижче подані статистичні дані, які показують рейтинг найпопулярніших мов програмування у другій половині 2020 року [6].

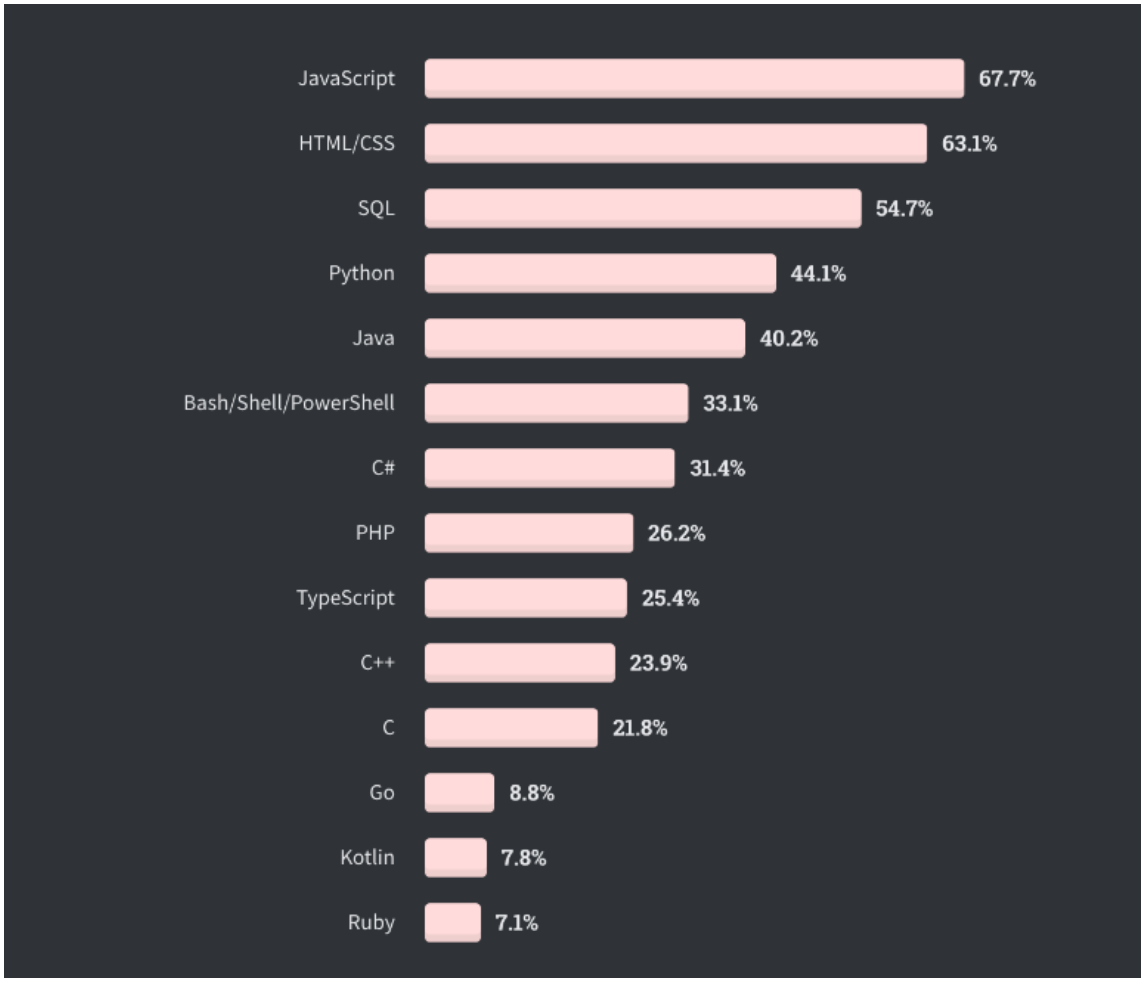


Рисунок 13 – діаграма популярності мов програмування

Слід звернути увагу на те, що розглядати рисунок вище не цілком правильно, адже мова Java використовується у значній більшій кількості сфер інформаційних технологій на відміну від Kotlin, який майже завжди використовується лише у андроїд програмуванні.

Тепер подивимося статистику швидкозростаючих мов програмування за версіями Octoverse GitHub і Stack Overflow - Kotlin на четвертому місці. Це показує, що розвиток Kotlin не впливає на популярність Java і поки Android підтримує дві мови - нові додатки будуть випускатися на них обох.

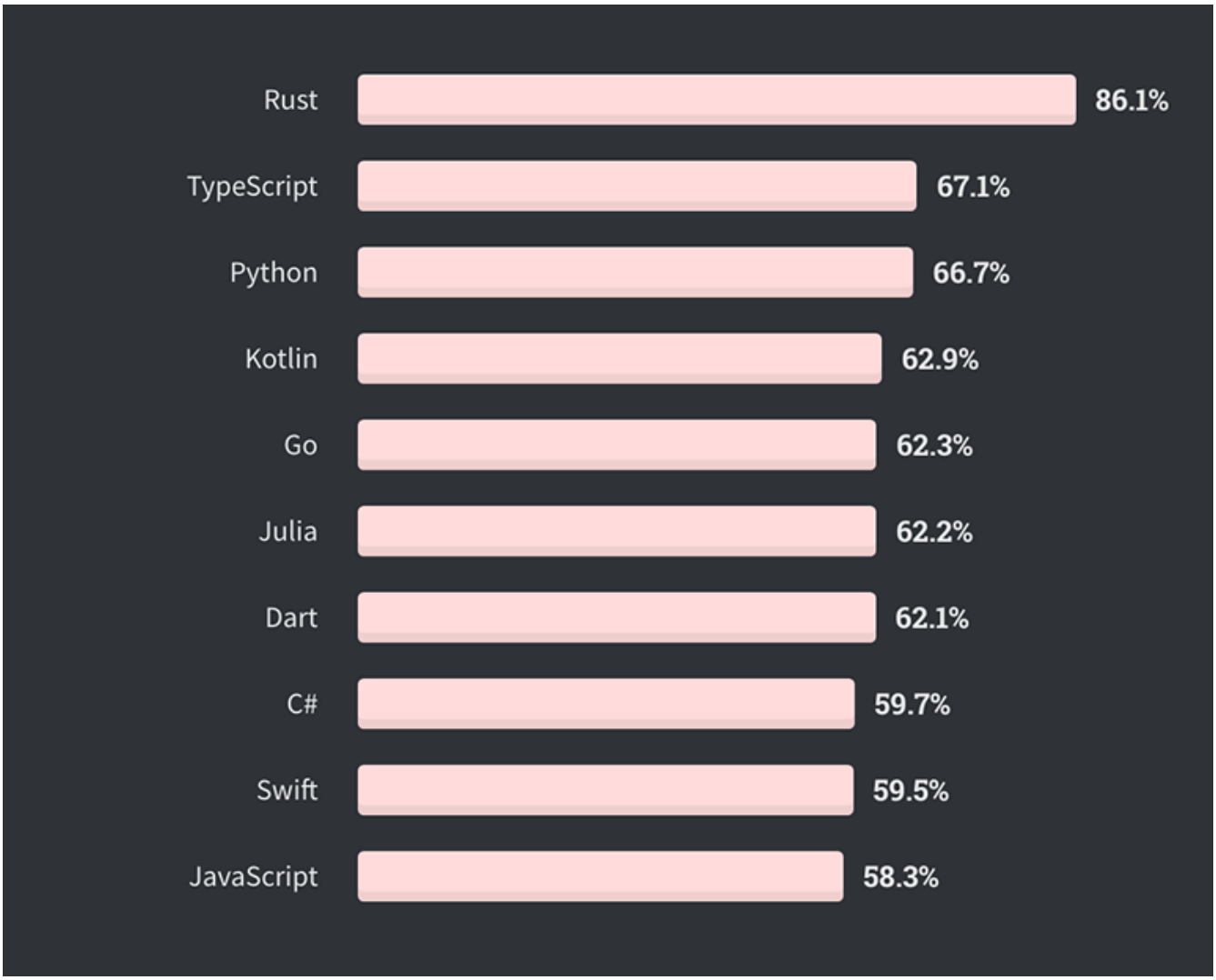


Рисунок 14 – Перелік найшвидше зростаючих мов програмування

Переваги Java:

– більше бібліотек;

Переваги Kotlin:

– лаконічніший код;
– Null Safety;
– новіша мова.

Отже, натомість більш застарілій мові Java був обраний Kotlin.

2.3.3 IDE

Для розробки програми операційної системи Android необхідно використовувати певні середовища програмування. Для розробки під ОС Android від компанії Google є офіційна середовище розробки, звана Android Studio. Крім офіційної IDE є кілька аналогів, не менше потужних і зручних в розробці мобільних додатків InlelijIDEA та Eclipse.

Android Studio - Це відносно нова середовище розробки Андроїд додатків, що базується на платформі Intellij IDEA компанії JetBrains (автори IntellijIdea, PhpStorm, AppCode, ReSharper), яка була анонсована на всесвітній конференції Google I\O 2013. На сьогоднішній день IntelijIDEA коштує $499.00/рік, тому ми розглянемо безкоштовні варіанти IDE і порівняємо основні відмінності в них в табл.3. [7].

Таблиця 3 – Порівняння IDE Android Studio та Eclipse

Android Studio	**Eclipse**
Офіційна IDE від Googe	Неофіційна
Безпосередньо підтримує андроїд	Підтримує андроїд-розробку через Google ADT пагін.
Має регулярну підтримку/оновлення	Не має регулярної підтримки андроїд-розробки
Підтримує тільки Java/Kotlin	Підтримує C, C++, PHP, Java, Perl, C#
Простий інтерфейс	Більш складніший інтерфейс
Більш швидка	Програє в швидкості
Потребує менше ресурсів	Більш вимогливий до CPU та RAM
Іноді крашиться	Майже не крашиться

Отже, порівнюючи усі переваги і недоліки обох IDE, доцільним є обрати Android Studio через її пряму підтримку Android та кращу оптимізацію щодо споживання ресурсів комп’ютера.

2.3.4 Сайт для парсингу

Серед сайтів з форумами, на яких викладують колоди для гри, є 2 найпопулярніших – Hearthpwn.com та Hsreplay.com.

HsReplay.com

Використовує технологію Javascript для підгрузки інформацію на сторінку. Значно програє в швидкості. Щоб спарсити цей сайт, потрібно емулювати роботу веб-браузера безпосередньо в додатку для того, шоб відправити сайту інформацію про девайс и прогрузити сторінку. З іншого боку, цей сайт має більшу популярність та відображає вірогідність перемоги на кожній колоді завдяки статистичним даним. Колоди оновляються один раз у тиждень.

Hearthpwn.com

Менш популярний, проте не використовує технологію Javascript для динамічного завантаження списка з колодами. Це означає, що його дуже легко парсити без емуляції роботи веб-браузера в додатку. Незважаючи на більш низьку популярність, на цьому сайті частіше викладують нові колоди. Проте він не показує вірогідність перемоги, а лише відображає інформацію, яку додав сам автор. Колоди оновляються у режимі реального часу.

Детально розглянувши обидва варіанти, я обрав другий сайт (Hearthpwn.com), тому що його легче парсити і в ньому частіше викладують деки – те що нам потрібно.

2.3.5 Допоміжні бібліотеки

Бібліотека для парсингу

У андроїді є метод, який дозволяє завантажувати Html-сторінки, та потім парсити їх «`Html.fromHtml(source)`».

Але для цього доведеться писати свій API, за допомогою якого будуть шукатися теги, класи в Html, та братися дані з них. На щастя є бібліотека, яка робить це відразу – Jsoup [8].

«Java-бібліотека jsoup призначена для розбору HTML-сторінок (парсинг), дозволяючи отримати необхідні дані, використовуючи DOM, CSS і методи в стилі jQuery. Бібліотека підтримує специфікації HTML5 і дозволяє аналізувати довільні сторінки, як це роблять сучасні браузери. Бібліотеці можна підсунути для аналізу URL, файл або рядок.» [9]

Бібліотека для завантаження фото

Існує 4 основні бібліотеки для завантаження фото з інтернету – Picasso, UImageLoader, Glide та Fresco [10]. Їх відмінності продемонстровані на рисунку 13.

	Picasso	UImageLoader	Glide	Fresco
Creator	Square Inc	Sergey Tarasevich	Bumptech	Facebook
Maturity(Oldest)	★★★★½	★★★★★	★★★★☆	★★★½☆
Volley and OKHTTP	✔	✔	✔	✔
Used Personally	✔	✔	✔	✘
Customizability	★★★★☆	★★★★½	★★★★½	★★★★☆
Network Image Use	★★★★★	★★★★☆	★★★★½	★★★★☆
GIF Support	✘	✘	✔	⚠
Image cropping	★★★★☆	★★★★☆	★★★★☆	★★★★★
Ease of use	★★★★★	★★★★½	★★★★★	★★★★☆
Speed	★★★★½	★★★★☆	★★★★½	★★★★½
Documentation	★★★★½	★★★★½	★★★★½	★★★★★
Disk + Mem Cache	✔	✔	✔	✔
Personal Rating	★★★★★	★★★★☆	★★★★½	★★★½☆
USP	Fast,most popular	Most mature	= Picasso	Ashmem, By Facebook

Рисунок 15 – Порівняння плагінів для завантаження фото

Як бачимо, найкращі варіанти це Picasso та UImageLoader. На практиці найшвидшою бібліотекою є Picasso. Це можна перевірити на практиці – завдяки проекту Github, який дозволяє порівняти кожну з цих 4-ох бібліотек [11] .

Бібліотека для дизайну

Material Design - це адаптивна система вказівок, компонентів та інструментів, що підтримують найкращі практики проектування інтерфейсу користувача. За підтримки відкритого коду Material впорядковує співпрацю між дизайнерами та розробниками та допомагає командам швидко створювати красиві продукти. У цій бібліотеці використовуються наступні принципи проектування дизайну [12]:

Поверхні та тіні, що використовуються як метафори, елементи інтерфейсу повинні бути індивідуально шаруватими поверхнями, укладеними один на одного або поруч. Тіні слід використовувати для передачі, які поверхні перебувають перед іншими, допомагаючи зосередити увагу та встановити ієрархію.

Зображення повинні бути повноцінними. Мінімізуйте інтервал між фотографіями, а також поля між фотографіями та краями екрана для більш захоплюючого та багатого вмісту інтерфейсу.

Кольори слід використовувати сміливо, щоб підкреслити брендинг та важливі елементи інтерфейсу. Обов’язково виберіть основний колір та колір акценту, щоб забезпечити стабільний досвід роботи у вашому додатку. Розгляньте можливість використання API палітри для витягування кольорів безпосередньо із вмісту для отримання більш захоплюючого досвіду.

Для вирівнювання та візуальної послідовності слід використовувати метрики та ключові лінії для вирівнювання та розміру вмісту відповідно до базової сітки 8dp. Для більш точного позиціонування тексту можна використовувати сітку 4dp. Там, де це доречно, вирівняйте такі елементи, як основний текст, ескізи, заголовки панелі додатків тощо, за стандартними лініями.

Бібліотека для серіалізації/десеріалізації даних

Gson - це бібліотека Android, яку можна використовувати для перетворення об'єктів Java у їх представлення JSON. Він також може бути використаний для перетворення рядка JSON в еквівалентний об'єкт Java або Kotlin. Gson може працювати з довільними об'єктами Java/Kotlin, включаючи вже існуючі об'єкти, для яких у вас немає вихідного коду.

Є кілька проектів з відкритим кодом, які можуть перетворити об’єкти Java у JSON. Однак більшість з них вимагають розміщення анотацій Java у своїх класах; те, що ви не можете зробити, якщо у вас немає доступу до вихідного коду. Більшість також не повністю підтримують використання Java Generics. Gson розглядає обидва ці фактори як дуже важливі цілі проектування [13].

Цілі

– Надає прості методи toJson () та fromJson () для перетворення об'єктів Java у JSON та навпаки

– Дозволяє перетворення існуючих немодифікованих об’єктів у JSON і з нього

– Широка підтримка Java Generics

– Дозволяє власні подання для об’єктів

– Підтримка довільно складних об'єктів (з глибокою ієрархією успадкування та широким використанням загальних типів)

2.3.1 База даних

База даних Firebase Realtime є хмарною базою даних. Дані зберігаються у форматі JSON і синхронізуються в режимі реального часу з кожним підключеним клієнтом. Коли ви створюєте крос-платформні програми з нашими SDK для iOS, Android та JavaScript, усі ваші клієнти використовують один екземпляр бази даних

у реальному часі та автоматично отримують оновлення з найновішими даними [14].

Основні відмінності від інших баз даних:

– У реальному часі

Замість типових запитів HTTP, база даних Firebase Realtime використовує синхронізацію даних - кожен раз, коли дані змінюються, будь-який підключений пристрій отримує це оновлення протягом мілісекунд.

– Офлайн-програми

Firebase залишаються чуйним навіть у режимі офлайн, оскільки пакет SDK Firebase Realtime Database зберігає ваші дані на диску. Після відновлення зв’язку клієнтський пристрій отримує будь-які пропущені зміни, синхронізуючи його з поточним станом сервера.

– Доступ з клієнтських пристроїв

До бази даних Firebase Realtime можна отримати доступ безпосередньо з мобільного пристрою або веб-браузера; немає необхідності в сервері додатків. Безпека та перевірка даних доступні за допомогою правил безпеки бази даних Firebase у реальному часі, правил на основі виразів, які виконуються під час читання чи запису даних.

– Масштабування між кількома базами даних

За допомогою бази даних Firebase Realtime Database за ціновим планом Blaze можливо масштабно підтримувати потреби у своїй програмі, розподіляючи дані між кількома екземплярами баз даних в одному проекті Firebase. Оптимізуйте автентифікацію за допомогою автентифікації Firebase у вашому проекті та автентифікуйте користувачів у всіх екземплярах бази даних. Контролюйте доступ до даних у кожній базі даних за допомогою спеціальних правил бази даних Firebase у реальному часі для кожного екземпляра бази даних.

2.3.2 Контроль версій

Для контролю версій було використане рішення використовувати GitHub, адже це найбільш популярний сервіс для зберігання коду.

Github - один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome). Сервіс безкоштовний для проєктів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL), а для окремих індивідуальних проектів пропонуються різні платні тарифні плани.

2.4 Висновки до розділу 2.3

Отже, ми оглянули найпопулярніші інструменти розробки та ресурси, які потрібні нам для створення додатку. Також, щоб спростити собі задачу, ми будемо використовувати кастомні бібліотеки від офіційних розробників гугл (такі як Firebase Analytics), та від сторонніх розробників (такі як Jsoup, Picasso). В підсумку ми обрали такі інструменти (див. табл. 4):

Таблиця 4 – Перелік обраних інструментів

Цільова ОС:	Android
Мова програмування:	Kotlin
IDE:	Android Studio
Сайт для парсингу:	hearthpwn.com
Бібліотека для парсингу:	Jsoup
Бібліотека для завантаження фото:	Picasso
Бібліотека для дизайну:	Material Design
Бібліотека для серіалізації/десеріалізації даних:	Gson

2.5 Архітектура

Додаток буде розроблено на основі архітектури MVP.

Потрібно сказати, що не зовсім коректно називати MVP патерном, так як це швидше загальні підходи до розробки системи, які засновані на комбінації декількох патернів. Але в подальшому ми не будемо загострювати на цьому увагу і залишимо назву "патерни". Основна ідея будь-якого з патернів MVP, MVC, MVVM полягає в поділі логіки і UI-частини програми так, щоб їх можна було тестувати окремо. При цьому самі патерни досить сильно відрізняються між собою, тому розглянемо їх, щоб зрозуміти, чому в Android в першу чергу використовується MVP [15].

Почнемо розгляд патернів з їхніх спільних частин - View і Model:

– Є два підходи до розуміння цієї сутності. Хтось оперує поняттям Model в сенсі всього шару даних в додатку: це і бізнес-об'єкти, що містять логіку, і спосіб їх отримання (Repository), і якісь менеджери та інші елементи, що відносяться до даних. Такий підхід доречний, якщо говорити, що ваша система використовує виключно патерн MVP і більше ніяких елементів. Але ми вирішили зберегти шар даних в тому вигляді, в якому він був викладений в принципах "чистої" архітектури, тому під Model ми будемо розуміти звичайні класи об'єктів, які використовуються при взаємодії View з делегатом. Плюс такого підходу полягає в тому, що ми поділяємо суті, що може спрощувати розуміння. На кінцевий результат використання різних патернів ніяк не впливає, але це потрібно враховувати при вивченні інших джерел.

– View відображає дані, одержувані або від Model, або від делегата, що залежить від конкретного патерна. View - це та частина системи, яку видно користувачеві і яка взаємодіє з ним. При цьому View не повинна містити логіку, а передавати результати взаємодії делегату, який буде керувати цією View.

Схема патерну MVP виглядає наступним чином:

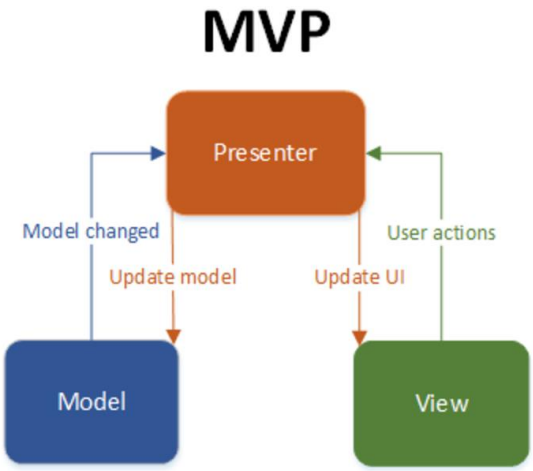


Рисунок 16 – Схема патерну MVP

Патерн MVP знайшов куди більше застосування в Android. MVP має кілька основних відмінностей від MVC. По-перше, Presenter управляє тільки однієї View і взаємодіє з нею через спеціальний інтерфейс. По-друге, View управляється тільки за допомогою Presenter-а, а не відстежує зміну Model. Presenter отримує всі дані з Model (або з шару даних в нашому викладі), обробляє їх відповідно до необхідної логікою і управляє View. Схема патерну MVP у порівнянні з MVC виглядає наступним чином:

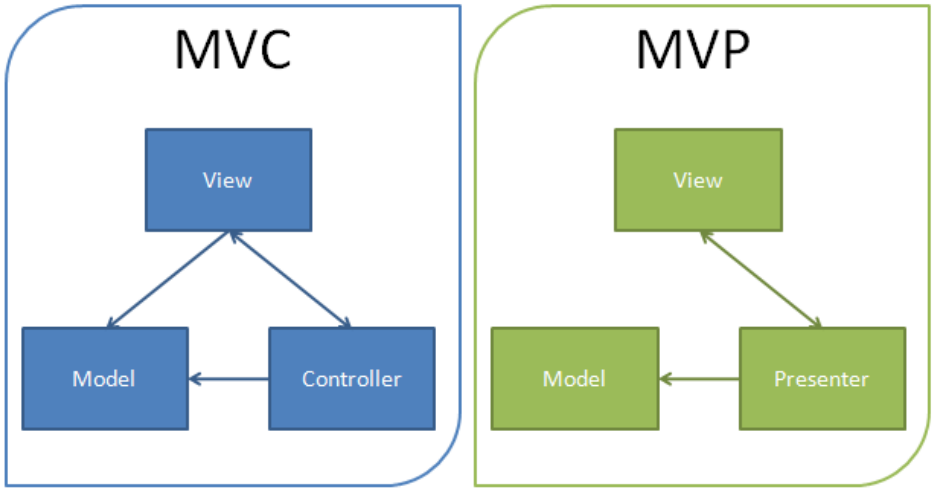


Рисунок 17 – Порівняння MVC з MVP

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Процес розробки

Для початку напишемо класи з даними.

Класи з даними (Data Class) – це класи, єдиним призначенням яких є зберігання даних. Функціонал таких класів залежить від самих даних, які в них зберігаються. У Kotlin клас може бути позначений словом data:

```
data class User(val name: String, val age: Int)
```

Компілятор автоматично формує наступні члени даного класу з властивостей, оголошених в основному конструкторі:

– пару функцій equals () / hashCode (),

– функцію toString () у формі "User (name = John, age = 42)",

– компонентні функції componentN (), які відповідають властивостям, відповідно до порядку їх оголошення,

– функцію copy ()

Якщо будь-яка з цих функцій явно визначена в тілі класу (або успадкована від батьківського класу), то генеруватися вона не буде.

Для забезпечення узгодженості і осмисленого поведінки згенерованого коду класи даних повинні відповідати таким вимогам [16]:

– Основний конструктор повинен мати як мінімум один параметр;

– Всі параметри основного конструктора повинні бути відзначені, як val або var;

– Класи даних не можуть бути абстрактними, open, sealed або inner;

– Класи даних не можуть успадковуватися від інших класів (але можуть реалізовувати інтерфейси).

Почнемо зі створення класу Card. У карти є такі поля з даними:

– назва;

– кількість цієї карти в колоді;

– рідкість;

– кількість мани;

– посилання на сторінку з картою в інтернеті;

```
data class Card(
    val name: String = "",
    val amount: String = "",
    val rarity: CardRarity = CardRarity.COMMON,
    val cost: String = "",
    val linkOnCard: String = ""
)
```

Рисунок 18 – клас User.kt

В свою чергу, нам також потрібно написати класи-перерахування (Enum) для типів CardRarity, GameClasses та GameFormat.

```
5   enum class CardRarity(
6       val colorRes: Int
7   ) {
8       FREE(R.color.black),
9       EPIC(R.color.purple),
10      RARE(R.color.blue),
11      COMMON(R.color.green),
12      LEGENDARY(R.color.orange);
13  }
```

Рисунок 19 – клас CardRarity.kt

Enum-класи в котліні – це класи перерахування з безпекою типів. Кожна enum-константа є об'єктом. При оголошенні константи розділяються комами. Так як константи є екземплярами enum-класу, вони можуть бути ініційовані.

Enum-константи також можуть оголошувати свої власні анонімні класи як з їх власними методами, так і з перевантаженими методами базового класу [17].

Після написання дата-класів, наша ієрархія моделей програми виглядає наступним чином.

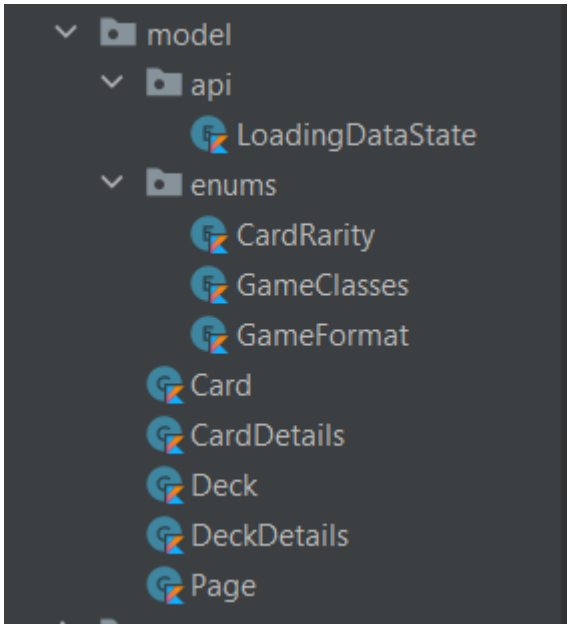


Рисунок 20 – ієрархія класів з даними

Також створимо допоміжні класи (Extension), які будуть містити в собі глобальні функції, які розширюють функціонал інших класів.

Також по ходу написання додатку ми будемо додавати інші Extension класи. У кінці роботи список extenstion-класів буде виглядати наступним чином:

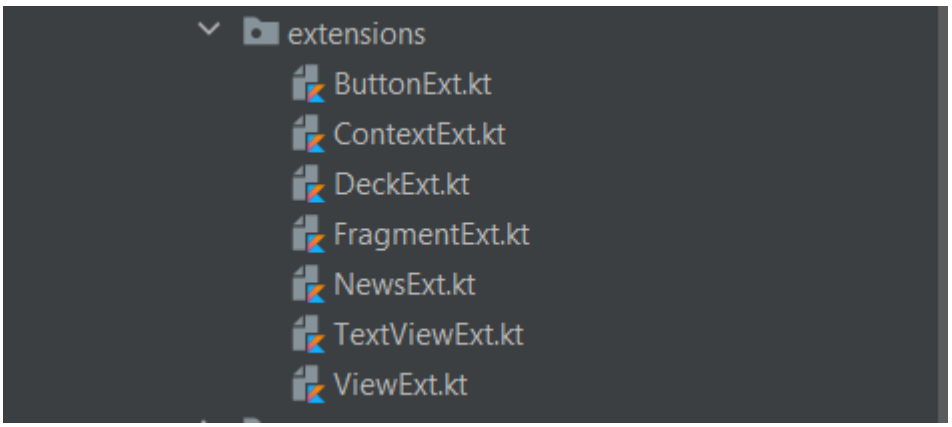


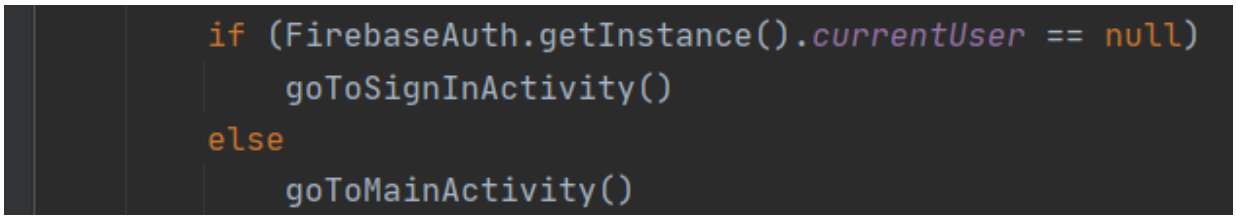
Рисунок 21 – перелік Extension класів

```
package com.cyberquick.hearthstonedecks.other.extensions

import ...

fun Context.toast(message: String) {
    Toast.makeText( context: this, message, Toast.LENGTH_SHORT).show()
}

tailrec fun Context.getActivity(): AppCompatActivity? = this as? AppCompatActivity
    ?: (this as? ContextWrapper)?.baseContext?.getActivity()

fun Context.color(color: Int) = ContextCompat.getColor( context: this, color)

fun Context.drawable(drawable: Int) = ContextCompat.getDrawable( context: this, drawable)
```

Рисунок 22 – приклад Extension-класу

Тепер можна почати писати функціональну логіку програми. Почнемо з класу SplashActivity, який успадковує клас AppCompatActivity. Цей клас буде створений першим під час життєвого циклу програми.

Основна його функція – заповнити пустий екран повноекранним фоном з логотипом додатку, поки загружаються ресурси і класи. Також в ньому ми перевіряємо, чи авторизований юзер в системі.



```
if (FirebaseAuth.getInstance().currentUser == null)
    goToSignInActivity()
else
    goToMainActivity()
```

Рисунок 23 – перевірка стану авторизації юзера

Якщо так – то йдемо далі, якщо не авторизований – то на сторінку авторизації. Після процесу авторизації SplashActivity отримає повідомлення про те, чи авторизувався юзер вдало. На екрані з’являється повідомлення і додаток завершує роботу у разі, якщо виникла помилка.

Після цього створимо клас MainActivity, який буде містити в собі:

– бокову навігацію
– контейнери з фаргментами
– тулбар

Усі інші екрани будуть знаходитися у середині MainAcitivity, тому що ми використовуємо принцип Single-activity. Звісно, не беручи до уваги друге наше активіті – SplashActivity, тому що воно лише показує фоновий екран і запускає MainAcitivty, де й відбувається уся логіка додатку. Тому цим активіті можна знехтувати.

Коротко кажучи, архітектура з одним активіті - це архітектура, яка має лише одну активіті або відносно невелику кількість видів активіті. Замість того, щоб одне Activity представляло один екран, ми розглядаємо Activity як великий контейнер з фрагментами всередині Activity, що представляють екран [18].

Далі створимо фрагменти:

– усі колоди з інтернету
– мої колоди
– перегляд однієї колоди
– перегляд карти
– «про додаток»

Запит до бази даних може виконуватись для доступу до списку «Мої колоди» і виконанням дій з ним, а саме для чотирьох випадків:

– зберегти колоду у список
– видалити колоду з списку
– завантажити список
– перевірити, чи є колода у списку

Дані з бази даних без багів десеріалізуються, навіть якщо деякі поля не заповнені, з формату JSON у формат Deck.

Рисунок 24 – приклад запису до бази даних

Робота з базою даних може виконуватись у двох режимах: постійна синхронізація даних або одноразовий запит до бази даних. Постійна синхронізація виконується, наприклад, під час спілкування у чаті, коли є потреба у моментальному доставленні повідомлення до співрозмовника.

У нашому разі буде достатньо одноразових запитів, щоб при вході в додаток намагатись завантажити базу даних. Також при кожному вході в додаток дані будуть кожен раз оновлятися, щоб підтримувати актуальність даних. Але дані не будуть завантажуватися повністю – Realtime Database працює таким чином, що спочатку він перевіряє локальне сховище, і якщо буде помітно різницю з даними на сервері, то тільки тоді почнеться оновлення даних.

Порядок запитів до бази даних буде відбуватись наступним чином: спочатку ми спробуємо завантажити дані з серверу. Якщо сервер бачить, що дані на сервері і в локальному сховищі відрізняються – то відбувається синхронізація з сервером. Після того, як дані завантажились, ми кладемо їх у локальне сховище. Якщо ж виникла помилка при завантаженні (наприклад, був вимкнений інтернет) – то

запит повторюється до тих пір, поки вони не будуть завантажені. Діаграма, яка показує принцип завантаження даних з бази даних, представлена на рисунку нижче.

Дизайн будемо створювати за допомогою стандартних макетів (Layout), які пишуться на мові XML.

Макет визначає структуру для користувальницького інтерфейсу у вашому додатку, наприклад, в діяльності. Усі елементи в макеті побудовані з використанням ієрархії об'єктів View та ViewGroup. View зазвичай малює те, що користувач може бачити та взаємодіяти. Тоді як ViewGroup - це невидимий контейнер, який визначає структуру макета для View та інших об’єктів ViewGroup, як показано на рисунку нижче.

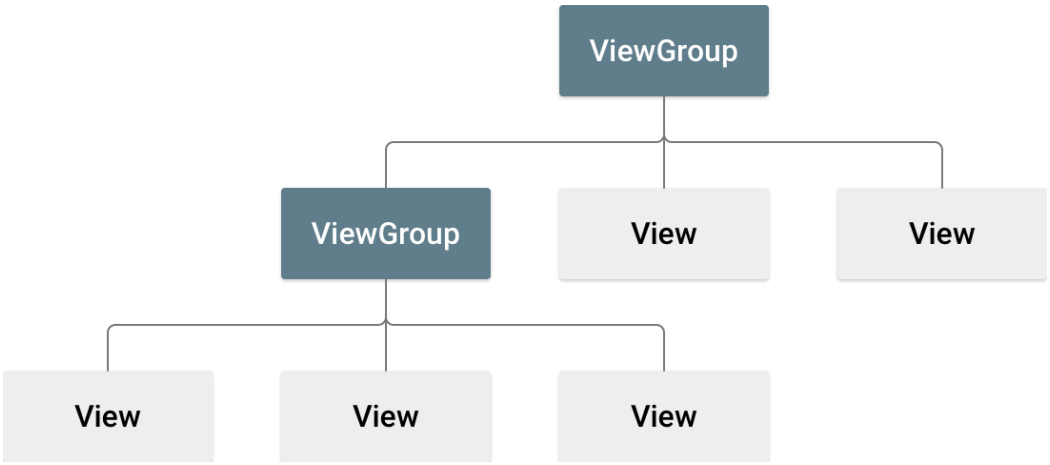


Рисунок 25 – схема роботи View та ViewGroup

Використовуючи XML для Android, можна розробляти макети інтерфейсу та елементи екрану, які вони містять, таким же чином, як ви створюєте веб-сторінки в HTML - із низкою вкладених елементів.

Кожен файл макета повинен містити рівно один кореневий елемент, який повинен бути об'єктом View або ViewGroup. Визначивши кореневий елемент, ви

можете додати додаткові об’єкти макета або віджети як дочірні елементи, щоб поступово будувати ієрархію подання, яка визначає ваш макет. Щоб наблюдати результати дизайну у реальному часі, ми використаємо стандартний інструмент для перегляду макетів у андроїд. Знизу можна побачити деякі макети додатку.

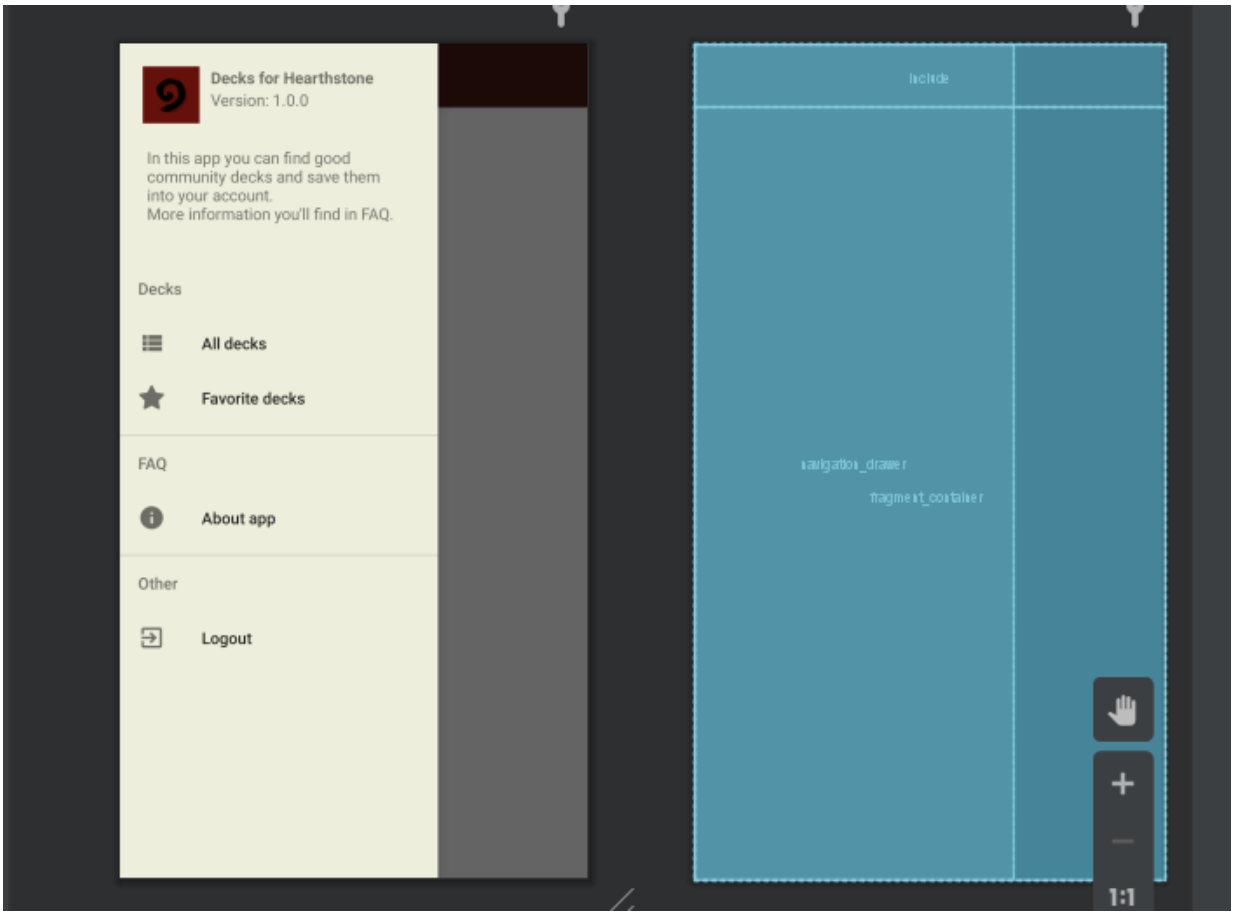


Рисунок 26 – дизайн activity_main.xml

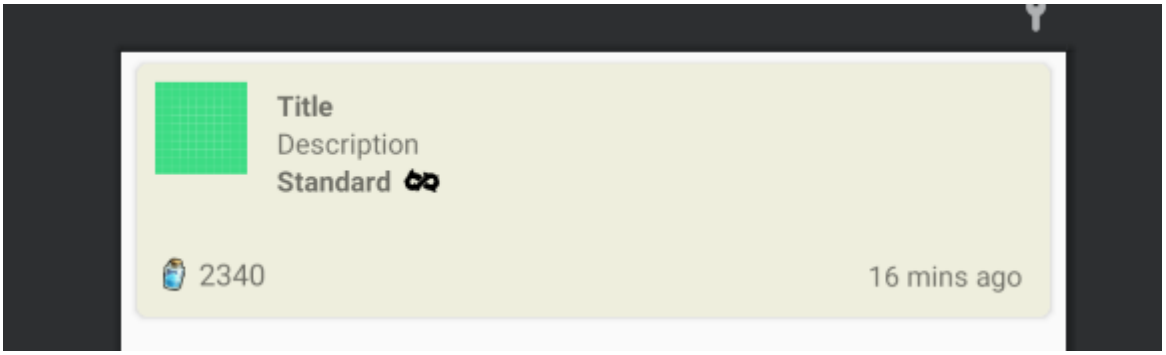


Рисунок 27 – дизайн item_deck_preview.xml

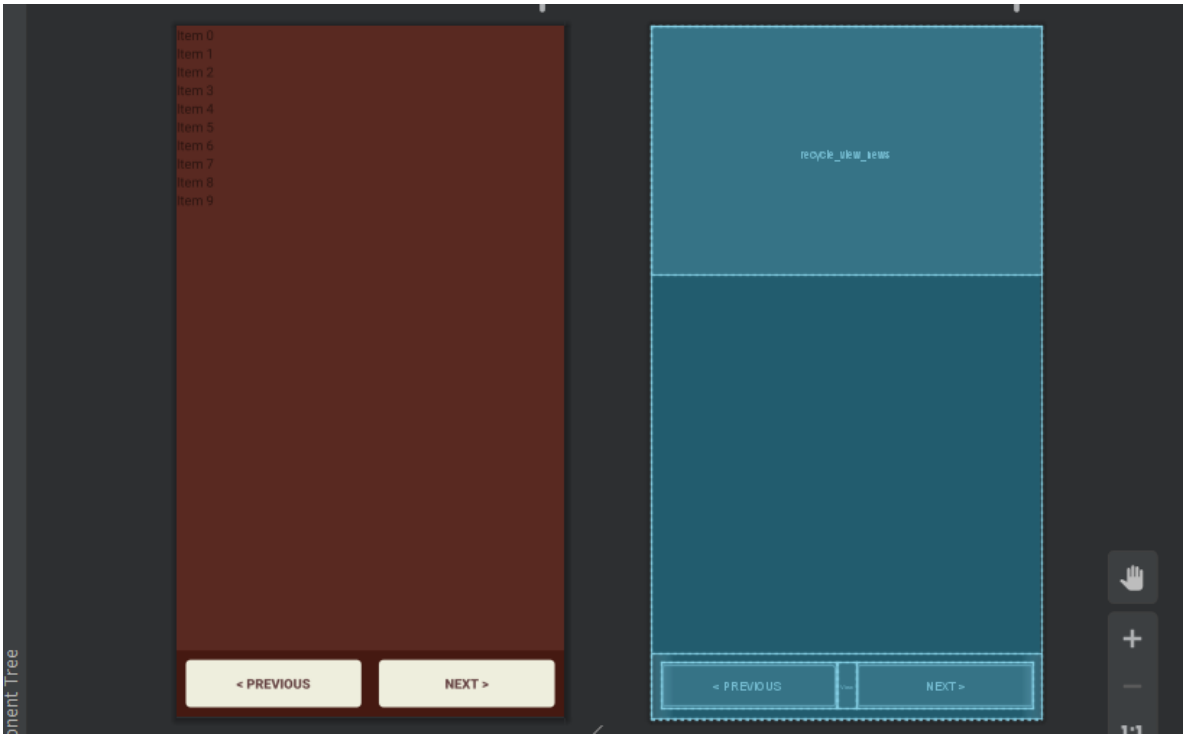


Рисунок 28 – дизайн fragment_all_decks.xml

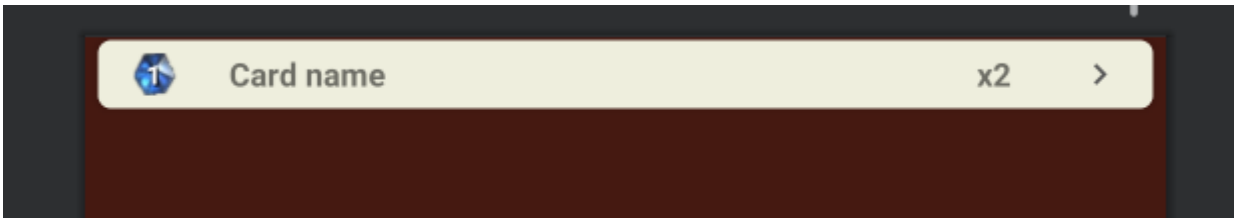


Рисунок 29 – дизайн item_card.xml

У кінці, наша структура файлів з UI-логікою буде мати наступний вигляд:

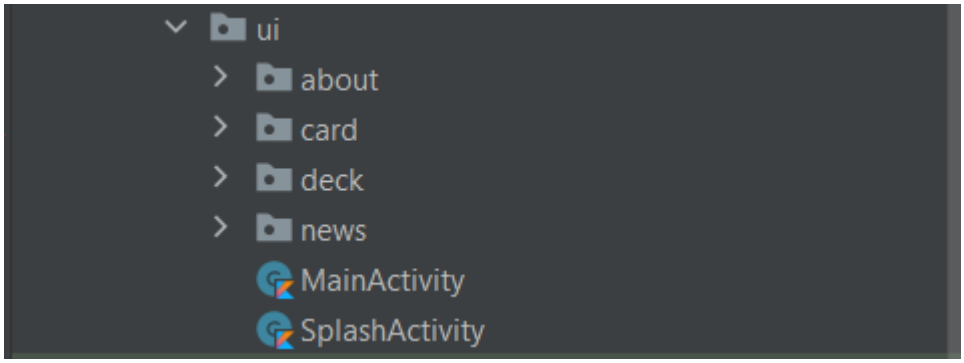


Рисунок 30 – ієрархія класів з UI-логікою

3.2 Опис функцій

Під час запуску додатку на екрані користувача відображається екран запуску програми, під час якого перевіряється авторизація юзера та відбувається рендер наступного екрану. Якщо юзер не авторизований в системі, то перед ним відкриється екран реєстрації.

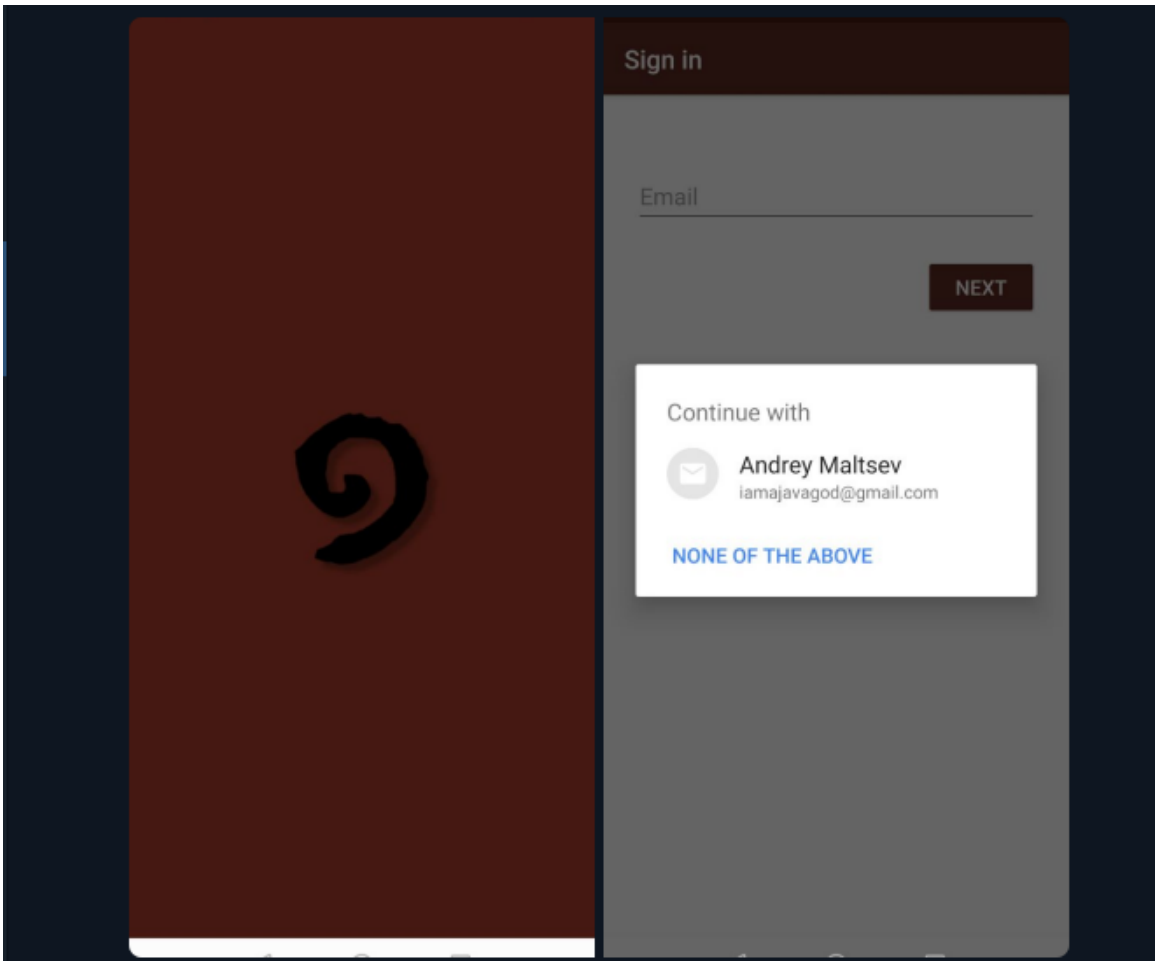


Рисунок 31 – перший екран запуску додатку та екран авторизації

Якщо користувач натисне на свій акаунт, то поле з e-mail заповниться обраним адресом. Далі юзер побачить екран з реєстрацію або з входом в додаток, в залежності від того, чи зареєстрований він в системі.

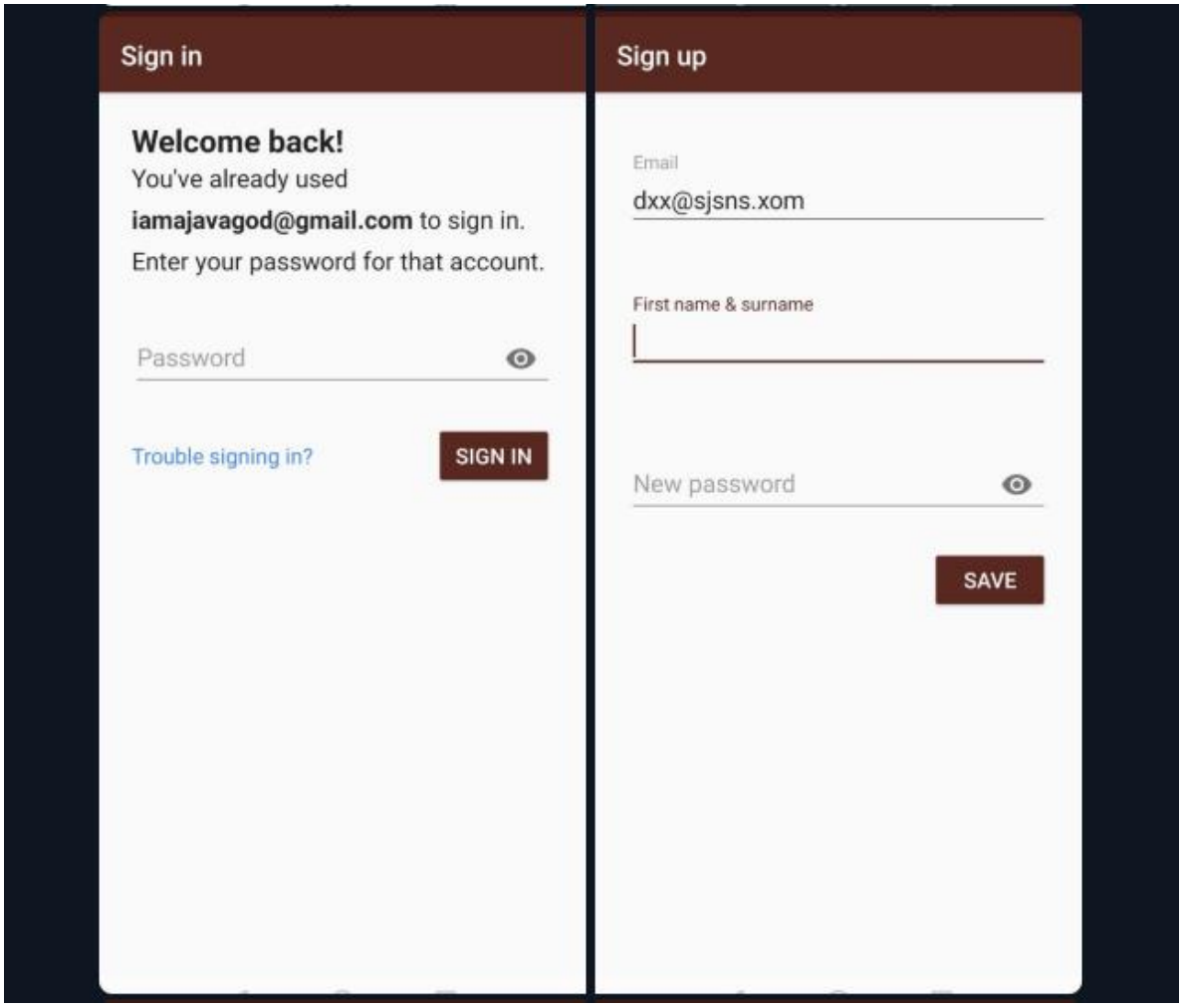


Рисунок 32 – екран входу та екран реєстрації

На екрані входу юзер повинен буде ввести свій пароль та натиснути кнопку “Sign In”. Якщо дані правильні – він перейде на MainActivity, якщо ні – то виведеться повідомлення про помилку. Також на цій сторінці є функція відновлення паролю у разі, якщо юзер його забув.

На екрані реєстрації юзер буде повинен ввести свій E-mail, ім’я, прізвище та придумати пароль. Пароль повинен бути від 6 символів та мати латинські літери з цифрами. Після вдалої авторизації юзер побачить перед собою головний екран програми.

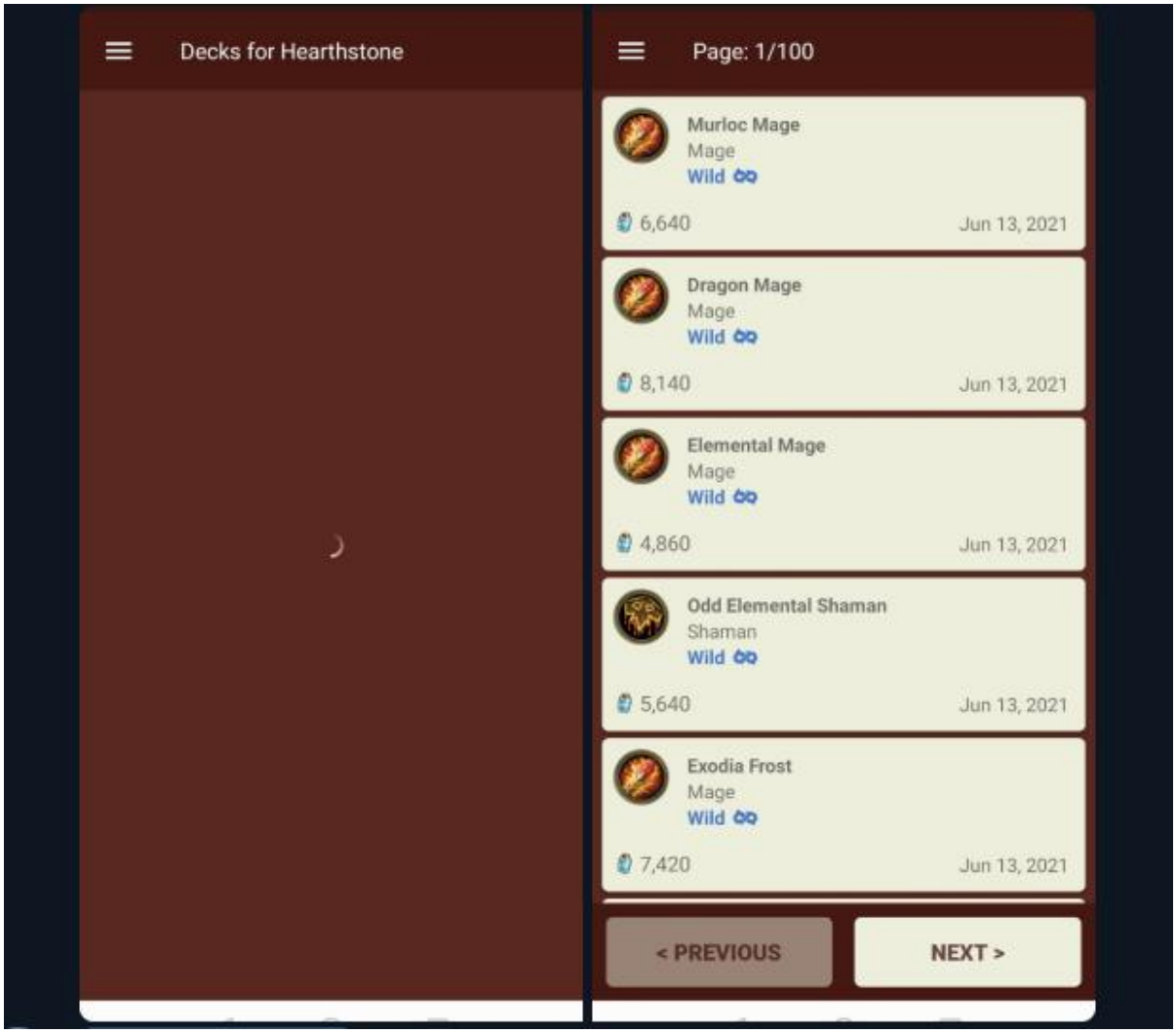


Рисунок 33 – головний екран

На головному екрані юзер спочатку бачить перед собою прогрес бар, після якого юзер може побачить або помилку, або список з колодами з інтернету (якщо дані завантажились вдало).

Тут юзер може натиснути на будь-яку колоду, щоб перейти до її детально перегляду, або перегорнути сторінку кнопками «Previous», «Next». Також тут юзер може відкрити головне меню з усіма основними функціями додатку та коротким описом про додаток.

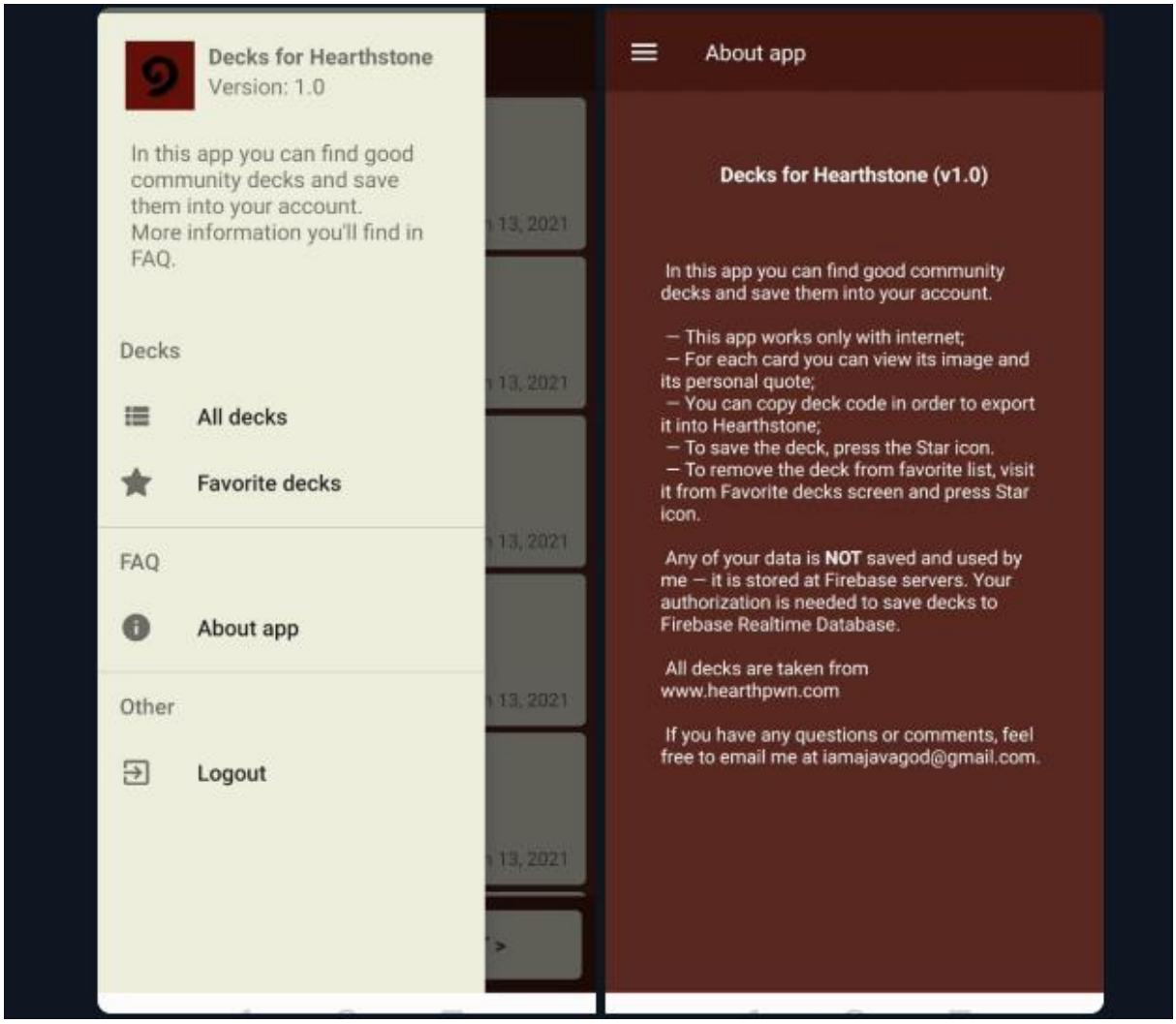


Рисунок 34 – бокове меню та екран «About app»

На боковій навігації є короткий опис про додаток, його версія, а також зручні кнопки для навігації:

– Кнопка «All decks», яка відкриває список з усіма колодами
– Кнопка «Favorite decks», яка відкриває збережені колоди
– Кнопка «About app», яка відкриває екран з інформацією про додаток
– Кнопка «Logout», яка виходить з акаунту

На екрані «About app» можна побачити коротку інформацію про додаток, а також короткий опис його функцій.

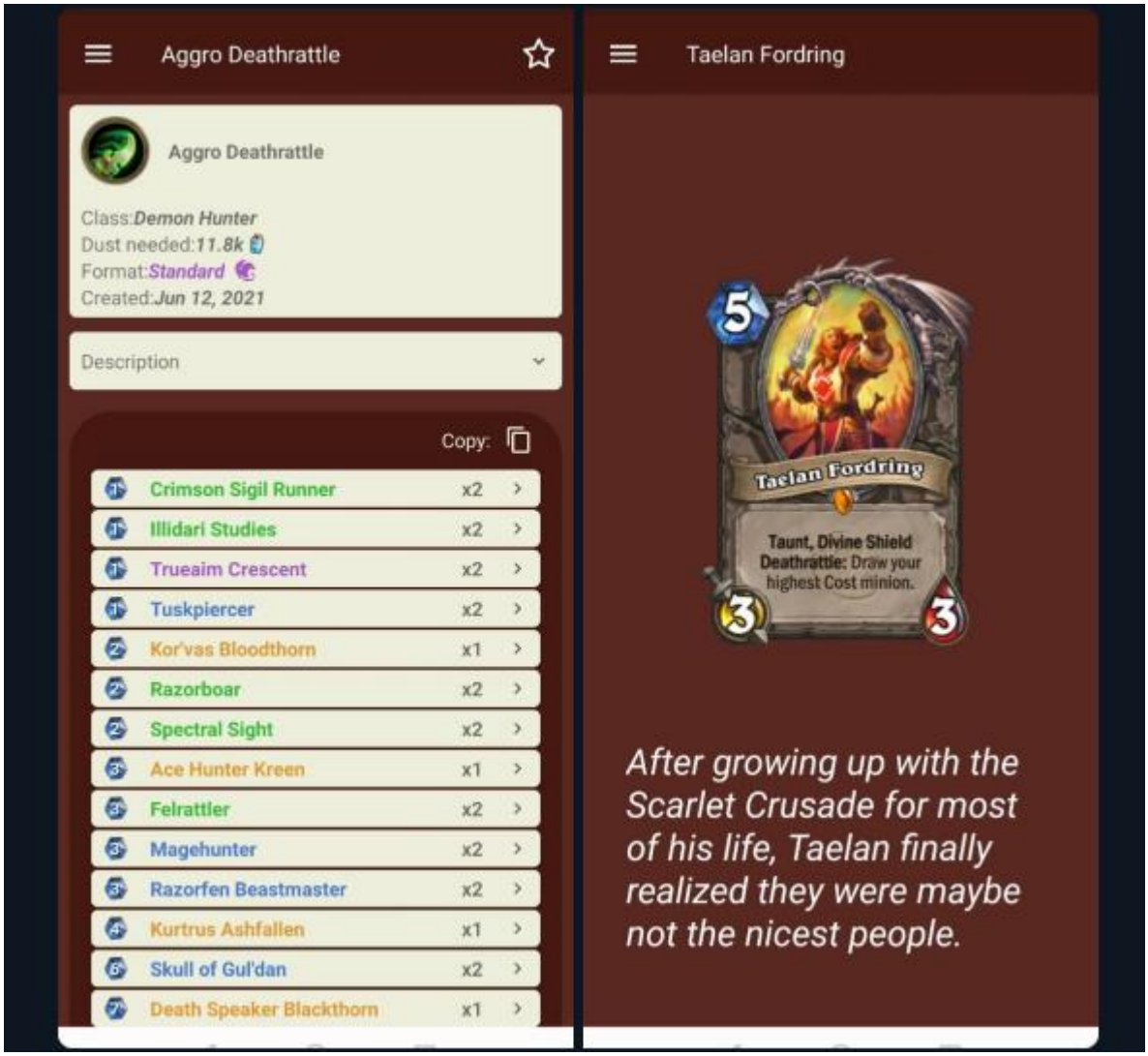


Рисунок 35 – детальна інформація про колоду і про карту

На сторінці з переглядом колоди можна побачити основну інформацію про неї, описання, кнопку для копіювання колоди в буфер, кнопку в правому куті для збереження колоди до «Мої колоди», а також перелік карт.

Після натискання на одну з карт, відкривається екран, на якому юзер бачить зображення карти та її цитату. Також з усіх екранів додатку можна відкрити бокове меню, з якого перейти на інший екран.

3.3 Захист бази даних

Також нам треба все ж таки мати на увазі, що можливий розвиток подій, коли зловмисник яким би то ні було чином отримає доступ до бази даних. Хоча й це малоймовірно, але не буде зайвим перестрахуватися й від такої ситуації. Отже, все що ми можемо зробити, це заборонити права прямого редагування бази даних для всіх юзерів.

Але одразу ж це рішення здається дивним, адже юзери не зможуть додавати свої дані в поля, та і встає питання, як взагалі у такому разі вони будуть додавати нові записи (наприклад, ім'я то пароль) до бази даних.

Ця проблема вирішується завдяки використанню так званих conditions in Realtime Database Rules (умов) [19]. До впровадження бази даних права доступу виглядають так, як на скріншоті нижче:

```
{
  "rules": {
        ".read": "true",
        ".write": "true"
  }
}
```

Рисунок 36 – стандартні права доступу до бази даних

Основним елементом правил безпеки баз даних у реальному часі є умова. Умова - це логічний вираз, який визначає, чи слід дозволити чи заборонити певну операцію. Для основних правил використання істинних та помилкових літералів як умов працює переважно добре. Але мова правил безпеки баз даних у реальному часі дає вам способи писати більш складні умови, які можуть:

– Перевірити автентифікацію користувача
– Оцінити наявні дані щодо нещодавно поданих даних
– Перевірити різні частини бази даних

– Перевірити вхідні дані

– Використовувати структуру вхідних запитів для логіки безпеки

Для того, щоб дати права на читання та редагування полів бази даних, ми маємо перевірити, по-перше, чи зареєстрований юзер у системі, по-друге, чи співпадає його ID з тим, до якого поля він звертається. Якщо не співпадає – він не зможе прочитати або змінити дані цього поля. Виходить, що для юзера доступні лише ті поля в системі, які стосуються саме його. Відредагуємо наші правила доступу до бази даних наступним чином, як показано на рисунку нижче:

```
{
  "rules": {
    "users": {
      "$user_id": {
        // grants read access to the owner of this user account
        // whose uid must exactly match the key ($user_id)
        ".read": "$user_id === auth.uid",
        // grants write access to the owner of this user account
        // whose uid must exactly match the key ($user_id)
        ".write": "$user_id === auth.uid"
      }
    }
  }
}
```

Рисунок 37 – Права доступу до бази даних після впровадження Conditions

3.4 Безпарольна аутентифікація

Для того щоб повисити рівень безпеки при автентифікації та зменшити ризик витіку пароля, ми можемо використовувати автентифікацію Firebase для входу в систему користувача, надіславши йому електронне повідомлення із посиланням, на яке вони можуть натиснути, щоб увійти. У процесі електронна адреса користувача також підтверджується.

Існує безліч переваг щодо входу електронною поштою:

Реєстрація та вхід із низьким тертям.

– Менший ризик повторного використання пароля в усіх додатках, що може підірвати безпеку навіть добре підібраних паролів.

– Можливість аутентифікації користувача, одночасно перевіряючи, що користувач є законним власником адреси електронної пошти.

– Користувачеві для входу потрібен лише доступний обліковий запис електронної пошти. Не потрібно володіти номером телефону або обліковим записом соціальних мереж.

– Користувач може надійно ввійти в систему без необхідності вводити (або пам’ятати) пароль, який може бути громіздким на мобільному пристрої.

– Існуючого користувача, який раніше входив за допомогою ідентифікатора електронної пошти (пароля або об'єднаного), можна оновити, щоб увійти лише за допомогою електронної пошти. Наприклад, користувач, який забув свій пароль, все одно може ввійти в систему без необхідності скидати пароль.

Для цього, по-перше, треба ввімкнути безпарольну автентифікацію в Firebase.

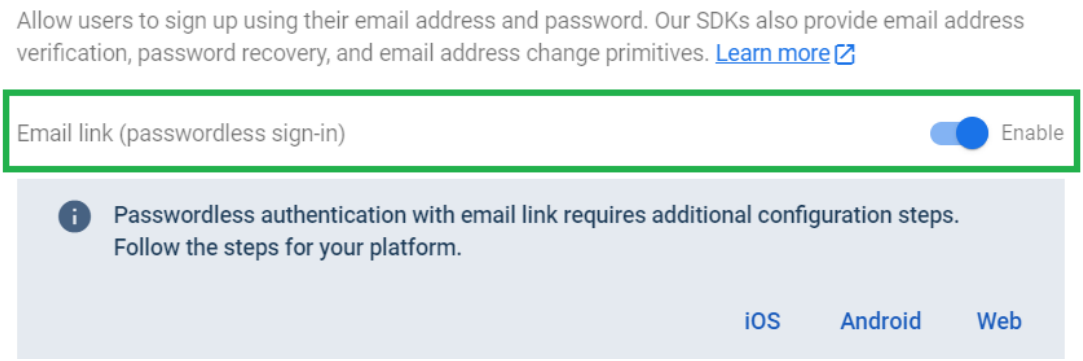


Рисунок 38 – Ввімкнення безпарольного входу в консолі Firebase

Потім створюємо динамічне посилання, яке буде грати роль сервера для зберігання посилання для підтвердження e-mail.

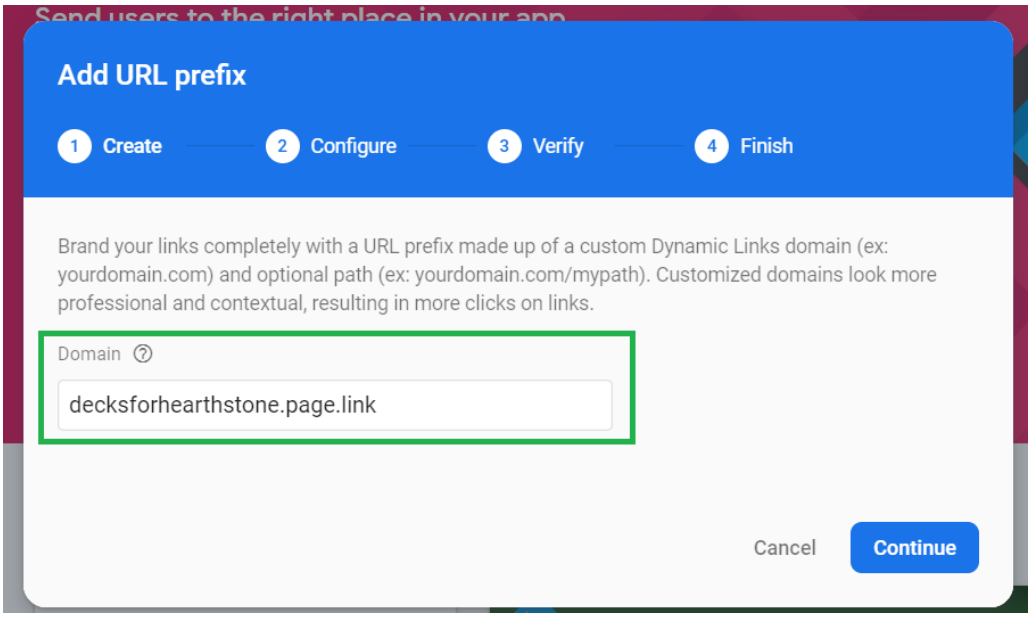


Рисунок 39 – Налаштування динамічного посилання.

Після цього додаємо це посилання до «білих» доменів в Firebase (це домени, з яких наш додаток може отримувати трафік).

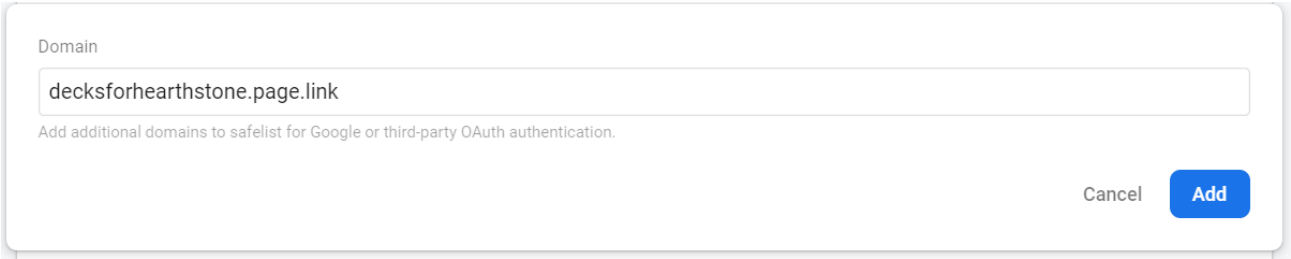


Рисунок 40 – Додаємо посилання до дозволених доменів

Коли все готово, додаємо наше динамічне посилання до сторінки авторизації.

```
    val actionCodeSettings = ActionCodeSettings.newBuilder()
        .setAndroidPackageName(packageName, installIfNotAvailable: true, minimumVersion: null)
        .setHandleCodeInApp(true)
        .setUrl("https://decksforhearthstone.page.link")
        .build()

    return arrayListOf(
        AuthUI.IdpConfig.EmailBuilder().enableEmailLinkSignIn()
            .setActionCodeSettings(actionCodeSettings).build(),)
}
```

Рисунок 41 – налаштування безпарольної автентифікації

Після того, як юзер натиснув посилання в E-mail, перед ним знов відкриється додаток, в який автоматично передається саме посилання, де ми в свою чергу перевіряємо його на валідність наступним чином:

```
39
40          val link = intent.data?.toString()
41          if (link != null && AuthUI.canHandleIntent(intent)) {
42              startActivityForResult(
43                  AuthUI.getInstance()
44                      .createSignInIntentBuilder()
45                      .setEmailLink(link)
46                      .setAvailableProviders(getAvailableProviders())
47                      .build(), RC_SIGN_IN)
48          }
```

Рисунок 42 – перевірка отриманого посилання на валідність

Щоб запобігти використанню посилання для входу іншим користувачем або на іншому пристрої, Firebase Auth вимагає введення електронної адреси користувача під час завершення входу. Для успішного входу ця електронна адреса повинна відповідати адресі, на яку спочатку було надіслано посилання для входу.

Після вдалої перевірки ми отримаємо результат процесу автентифікації від бібліотеки FirebaseAuth і якщо все добре, то переходимо на MainActivity.

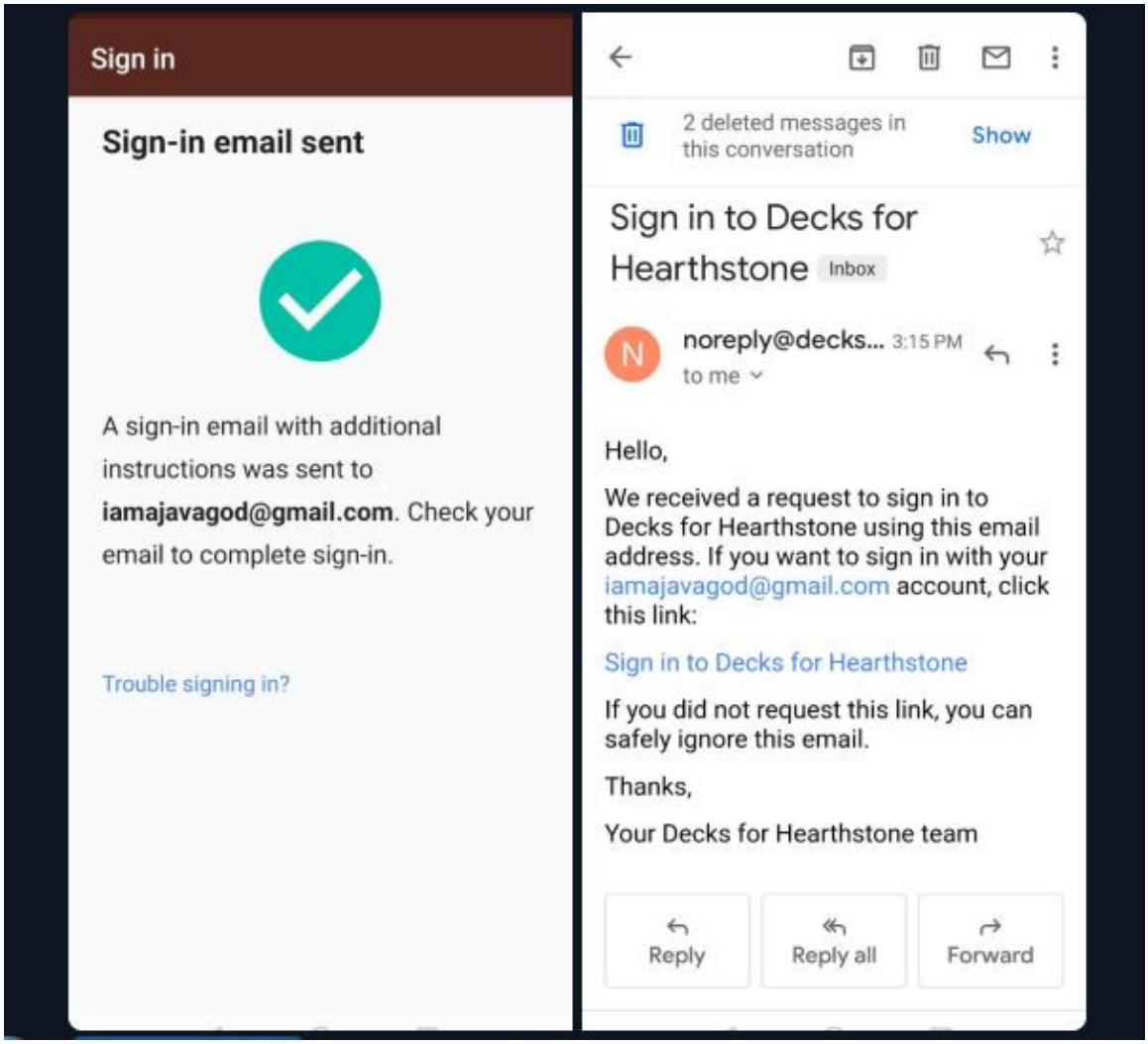


Рисунок 43 – отримання повідомлення на E-mail

Після завершення входу будь-який попередній неперевірений механізм входу буде видалено з система Firebase, а всі існуючі сеанси будуть визнані недійсними. Наприклад, якщо хтось раніше створив неперевірений обліковий запис із однаковими адресами електронної пошти та паролем, пароль користувача буде видалено, щоб запобігти тому, що перший користувач, який претендував на право власності та створив цей неперевірений обліковий запис, знову входив із неперевіреною електронною адресою та паролем.

3.5 Google Sign-in

Google Sign-in – це новий набір API, який забезпечує користувачам легкий та безпечний вхід та реєстрацію. Цей спосіб відрізняється від попереднього тим, що не має мети змусити юзера заходити на почту, а навпаки, здійснює авторизацію в один клік.

Деякі сервіси Google Play (наприклад, вхід в Google та запрошення в програми) вимагають надання SHA-1 сертифіката, щоб створити клієнт OAuth2 та ключ API для додатка.

Почнемо з отримання цього сертифікату. Вводимо наступну команду в консоль gradle нашого проекту в Android Studio:

```
$ ./gradlew signingReport
```

Рисунок 44 – команда для отримання SHA-1

Далі ми отримуємо такий вивід на екран, як на скріншоті (скріншот взятий для прикладу з офіційного сайту, він не мій):

```
> Task :app:signingReport
Variant: debug
Config: debug
Store: ~/.android/debug.keystore
Alias: AndroidDebugKey
MD5: A5:88:41:04:8D:06:71:6D:FE:33:76:87:AC:AD:19:23
SHA1: A7:89:E5:05:C8:17:A1:22:EA:90:6E:A6:EA:A3:D4:8B:3A:30:AB:18
SHA-256: 05:A2:2C:35:EE:F2:51:23:72:4D:72:67:A5:6C:8C:58:22:2A:00:
Valid until: Wednesday, August 10, 2044
```

Рисунок 45 – вивід на екран після введення команди

Далі ми беремо цей сертифікат (нам потрібний рядок з SHA1 підписом) і додаємо його в наш проект в консолі Firebase. Тепер ми налаштували консоль і потрібно лише налаштувати код у самій програмі.

Там, де ми передаємо список провайдерів автентифікації в бібліотеку Firebase Auth, нам потрібно додати провайдер Google. Цей процес показаний на скріншоті.

```
return arrayListOf(
    AuthUI.IdpConfig.GoogleBuilder().build(),
    AuthUI.IdpConfig.EmailBuilder().enableEmailLinkSignIn()
        .setActionCodeSettings(actionCodeSettings).build(),)
```

Рисунок 46 – дописуємо провайдер в список

На реальному пристрої автентифікація через гугл працює без багів. Отже, ми ще спростили автентифікацію для юзера, не зменшивши при цьому рівень безпеки додатка.

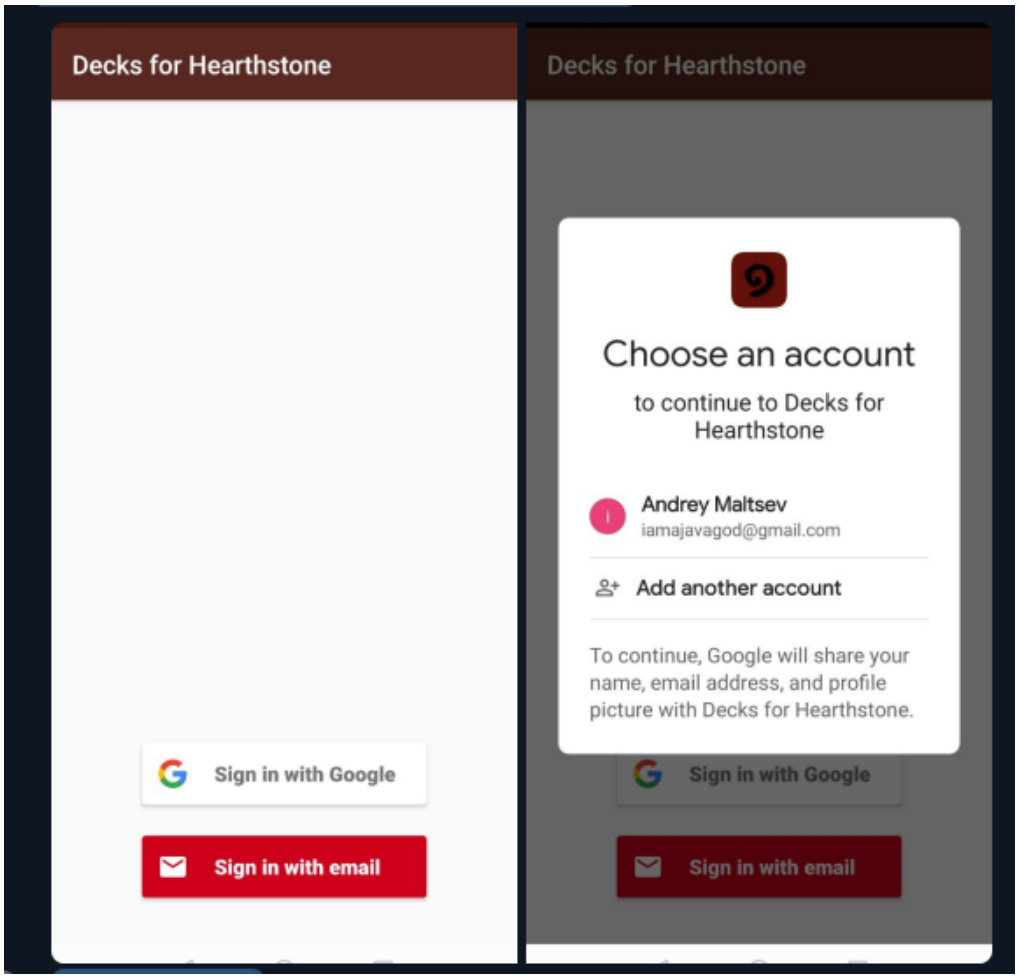


Рисунок 47 – Результат впровадження Google Sign-In

3.6 Тестування

У процесі розробки програми проводилося поетапне тестування з метою виявлення програмних помилок. Для цього нами були створені емулятори смартфона і планшета з різними діагоналями екрана для різних версій Android. Тестований програмний продукт послідовно запускався на цих емуляторах, його поведінка аналізувалося, і при необхідності по результатами аналізу вносилися зміни в код.

Для тестування окремих модулів роботи с базою даних в текст програми були внесені спеціальними функціями, які аналізувати базу даних і, при підозрі на помилку, що виводять повідомлення в системний журнал. Наприклад, при змінах в базі даних проводилася перевірка цілісності бази даних (перевірка на відповідність ключів - індексам), після чого при необхідності виводилося повідомлення в системний log.

Були проведені наведені нижче тести.

1. У базу даних навмисно вносилися неприпустимі дані в відповідні поля, які могли бути невірно інтерпретовані програмою. Потім мною аналізувалася поведінка активності у час обробки неприпустимих даних.
2. Додаток було запущено на пристроях, що працюють під керуванням різних версій Android з метою виявлення особливостей роботи програми, запущеної в різних операційних системах.
3. Після завершення циклу розробки, програмний продукт тестувався на реальних пристроях.

3.7 Можливі варіанти розвитку програмного додатка

Можливими шляхами розвитку додатка є:

1. Фільтр по ігровому режиму (Standard/Wild)
2. Пошук по ігровим колодам
3. Аутентифікація через сторонні соціальні мережі
4. Push-повідомлення про те, що вийшли нові колоди
5. Збереження завантажених колод в кеш
6. Можливий реліз в гугл плей

ВИСНОВКИ

Отже, ми закінчили розробку та тестування застосунку. Програмна система дає можливість швидкого та зручного доступу до списку актуальних колод та їх збереження. Також були додані функції, які покращують інтерфейс та навігацію по програмі, наприклад бокове меню.

В рамках дипломної роботи були досягнуті перераховані нижче результати.

1. Був оглянутий ринок існуючих застосунків на цю тему, врахували їхні недоліки (немає української мови, неактуальна центрів для України, неінтуїтивний інтерфейс) та спробували вирішити їх
2. Намальовано діаграму використання програми
3. Було обрано цільову архітектуру (Андроїд), мову програмування (Kotlin), IDE (Android Studio), допоміжні бібліотеки для парсингу сайтів, для завантаження фото, для дизайну, для серіалізації/десеріалізації даних, для аналітики, для розпізнавання тексту з фото та для зйомки фото.
4. Була обрана база даних (Firebase Realtime Database) та система контролю версій (GitHub).
5. Був обраний архітектурний патерн проектування (MVP).
6. Були враховані речі, які стосуються безпеки – в маніфесті був прописаний доступ тільки до захищеного трафіку, був розроблений безпечний механізм автентифікації користувачів (Безпарольний вхід та Google Sign-in) і також ми попіклувалися про безпеку бази даних – через застосування Rules Conditions.
7. Був написаний і протестований код програми.

Застосунок був протестований і він показав свою працездатність як на стандартних емуляторах, взятих з SDK Android, так і на реальних пристроях на платформі Android (смартфоні і планшетному ПК).

СПИСОК ЛІТЕРАТУРИ

1. Deck Finder for HS. *GooglePlay* : веб-сайт. URL: https://play.google.com/store/apps/details?id=com.neobile.hearthstonebestdecks&hl=ru
2. Малышкина Е.А. Совершенствование маркетинговых инструментов в интернет-бизнесе как фактор наиболее эффективного воздействия на потребителя: социально-экономические явления и процессы. Тамбов: Тамбовский государственный университет им. Г.Р. Державина, 2012. 136–146 с.
3. Розробка додатку для смартфону: що краще, андроїд чи iOS?. *Cosmoweb* : веб-сайт. URL: https://cosmoweb.kz/astana/blog/16-razrabotka-prilozheniya-dlya-smartfona-chto-luchshe-android-vs-ios
4. Java vs Kotlin для Android: думки розробників. *Habarahabr* : веб-сайт. URL: https://habr.com/ru/post/461877/
5. Рейтинг мов програмування 2020. *Dou* : веб-сайт. URL: https://dou.ua/lenta/articles/language-rating-jan-2020/
6. Порівняння мов програмування 2020. *Skillbox* : веб-сайт. URL: https://skillbox.ru/media/code/java_ili_kotlin_chto_vybrat_nachinayushchemu_android_razrabotchiku/
7. Android Studio Vs Eclipse – Which is Better for Android Developers? *Data Flair:* веб-сайт. URL: https://data-flair.training/blogs/android-studio-vs-eclipse/
8. Jsoup – Java HTML Parser (official site). *Jsoup*: веб-сайт. URL: https://jsoup.org/
9. Бібліотека jsoup. *AlexandrKalimov* : веб-сайт. URL: http://developer.alexanderklimov.ru/android/library/jsoup.php
10. Comparison Picasso vs UImageLoader vs Glide vs Fresco. *StackOverflow* : веб-сайт. URL: https://stackoverflow.com/questions/29363321/picasso-v-s-imageloader-v-s-fresco-vs-glide
11. Comparison of libraries downloading speed. *GitHub* : веб-сайт. URL: https://github.com/facebook/fresco/tree/master/samples/comparison

12. Bring your app to life with Material Design. *Android.com* : веб-сайт. URL: https://developer.android.com/distribute/best-practices/develop/use-material-design

13. GSON. *GitHub* : веб-сайт. URL: https://github.com/google/gson

14. Realtime Database. *Firebase* : веб-сайт. URL: https://firebase.google.com/docs/database

15. Model-View-Presenter. *Wikipedia* : веб-сайт. URL: https://uk.wikipedia.org/wiki/Model-View-Presenter

16. Antonio Leiva, «Kotlin for Android Developers», 2016 : книга, С. 54.

17. Dmitry Jemerov and Svetlana Isakova, «Kotlin in Action», 2021 : книга, С. 87.

18. Варакин М.В. Розробка мобільних додатків під Android. УЦ «Спеціаліст» при МГТУ ім. Н. Э. Баумана, 2012 : книга, С. 82-83.

19. Firebase Database Rules. *Firebase* : веб-сайт. URL: https://firebase.google.com/docs/database/security

ДОДАТОК А

Клас Card.kt

```
data class Card(
    val name: String = "",
    val amount: String = "",
    val rarity: CardRarity = CardRarity.COMMON,
    val cost: String = "",
    val linkOnCard: String = ""
)
```

Клас CardDetails.kt

```
data class CardDetails(
    val quote: String,
    val linkOnImg: String
)
```

Клас Deck.kt

```
data class Deck(
    val title: String = "",
    val gameClass: GameClasses = GameClasses.DemonHunter,
    val dust: String = "",
    val timeCreated: String = "",
    val linkDetails: String = "",
    val gameFormat: GameFormat = GameFormat.Standard,
    var deckDetails: DeckDetails? = null
)
```

Клас DeckDetails.kt

```
data class DeckDetails(
    val description: String = "",
    val code: String = "",
    val listOfCards: List<Card> = emptyList()
)
```

Клас Page.kt

```
data class Page(
    val pageNumber: Int,
    val listOfDecks: List<Deck>
)
```

Клас CardRarity.kt

```
enum class CardRarity(
    val colorRes: Int
) {
    FREE(R.color.black),
    EPIC(R.color.purple),
    RARE(R.color.blue),
    COMMON(R.color.green),
    LEGENDARY(R.color.orange);
}
```

Клас GameClasses.kt

```
enum class GameClasses(
    val titleInEnglish: String,
    val titleRes: Int,
    val imageRes: Int
) {
```

```
    DemonHunter("Demon Hunter", R.string.demon_hunter, R.drawable.demonhunter),
    Druid("Druid", R.string.druid, R.drawable.druid),
    Hunter("Hunter", R.string.hunter, R.drawable.hunter),
    Paladin("Paladin", R.string.paladin, R.drawable.paladin),
    Priest("Priest", R.string.priest, R.drawable.priest),
    Shaman("Shaman", R.string.shaman, R.drawable.shaman),
    Rogue("Rogue", R.string.rogue, R.drawable.rogue),
    Warlock("Warlock", R.string.warlock, R.drawable.warlock),
    Warrior("Warrior", R.string.warrior, R.drawable.warrior),
    Mage("Mage", R.string.mage, R.drawable.mage);
}
```

Клас GameFormat.kt

```
enum class GameFormat(val colorRes: Int, val iconRes: Int) {
    Standard(R.color.purple, R.drawable.ic_standard),
    Wild(R.color.blue, R.drawable.ic_wild);
}
```

Клас LoadingDataState.kt

```
enum class LoadingDataState {
    LOADING, LOADED, FAILED
}
```

Клас FirebaseHelper.kt

```
class FirebaseHelper {
    companion object {

        private val decksStorage get() = Firebase.database.reference
            .child("decks")
            .child(FirebaseAuth.getInstance().currentUser!!.uid)
```

```
        fun saveDeckToFavorite(deck: Deck, callback: (successful: Boolean) ->
Unit) {
            decksStorage
                .child(deck.id())
                .setValue(deck)
                .addOnCompleteListener {
                    callback(it.isSuccessful)
                }
                .addOnCanceledListener {
                    callback(false)
                }
        }

        fun removeFromFavorite(deck: Deck, callback: (successful: Boolean) ->
Unit) {
            decksStorage
                .child(deck.id())
                .removeValue()
                .addOnCompleteListener {
                    callback(it.isSuccessful)
                }
                .addOnCanceledListener {
                    callback(false)
                }
        }

        fun getFavoriteDecks(callback: (list: List<Deck>?) -> Unit) {
            decksStorage
                .get()
                .addOnSuccessListener {
                    val list = it.children.map { dataSnapshot ->
                        dataSnapshot.getValue(Deck::class.java)
                    }
                    callback(list.filterNotNull())
                }
```

```
                .addOnFailureListener {
                    callback(null)
                }
        }

        fun isInFavoriteList(deck: Deck, callback: (exists: Boolean) -> Unit) {
            decksStorage
                .child(deck.id())
                .addListenerForSingleValueEvent(object : ValueEventListener {
                    override fun onDataChange(snapshot: DataSnapshot) {
                        callback(snapshot.exists())
                    }

                    override fun onCancelled(error: DatabaseError) {}
                })
        }
    }
}
```

Клас SplashActivity.kt

```
class SplashActivity: AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        if (FirebaseAuth.getInstance().currentUser == null)
            goToSignInActivity()
        else
            goToMainActivity()
    }

    private fun goToSignInActivity() {
        val providers = arrayListOf(AuthUI.IdpConfig.EmailBuilder().build())
```

```
        startActivityForResult(AuthUI.getInstance()
                .createSignInIntentBuilder()
                .setAvailableProviders(providers)
                .build(),
            RC_SIGN_IN)
    }

    private fun goToMainActivity() {
        startActivity(Intent(this, MainActivity::class.java))
        finish()
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data:
Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == RC_SIGN_IN) {
            if (resultCode == Activity.RESULT_OK) {
                goToMainActivity()
            } else {
                toast("Sign-in failed")
                finish()
            }
        }
    }

    companion object {
        private const val RC_SIGN_IN = 12424
    }
}
```

Клас MainActivity.kt

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        simpleNavigate(AllDecksFragment())

        setSupportActionBar(topAppBar)
        initNavigationDrawer()
    }

    override fun onBackPressed() {
        if (supportFragmentManager.backStackEntryCount > 1)
            supportFragmentManager.popBackStack()
        else
            showExitWindow()
    }

    @SuppressLint("SetTextI18n")
    private fun initNavigationDrawer() {
        topAppBar.navigationIcon = drawable(R.drawable.ic_baseline_menu_24)

        topAppBar.setNavigationOnClickListener {
            drawerLayout.openDrawer(navigation_drawer)
        }

        navigation_drawer.setNavigationItemSelectedListener { menuItem ->
            drawerLayout.closeDrawer(navigation_drawer)
            when (menuItem.itemId) {
                R.id.drawer_menu_item_all_decks -> {
                    simpleNavigate(AllDecksFragment())
```

```
                }
                R.id.drawer_menu_item_my_decks -> {
                    simpleNavigate(FavoriteDecksFragment())
                }
                R.id.drawer_menu_item_about -> {
                    simpleNavigate(AboutAppFragment())
                }
                R.id.drawer_menu_item_logout -> {
                    showLogoutWindow()
                }
            }
            return@setNavigationItemSelectedListener true
        }

        navigation_drawer.addOnLayoutChangeListener(object :
View.OnLayoutChangeListener {
            override fun onLayoutChange(v: View?, left: Int, top: Int, right: Int,
bottom: Int,
                                        oldLeft: Int, oldTop: Int, oldRight: Int,
oldBottom: Int
            ) {
                navigation_drawer.removeOnLayoutChangeListener(this)
                tv_app_version.text = "Version: " + BuildConfig.VERSION_NAME
            }
        })
    }

    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        return super.onCreateOptionsMenu(menu)
    }

    private fun goToSplashActivity() {
        startActivity(Intent(this, SplashActivity::class.java))
        finish()
    }
```

```
private fun showExitWindow() {
    val dialog = MaterialAlertDialogBuilder(this)
        .setTitle("Quit app?")
        .setPositiveButton("Quit") { dialog, _ ->
            dialog.dismiss()
            finish()
        }
        .setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
        .show()

    dialog.getButton(DialogInterface.BUTTON_POSITIVE)
        .setTextColor(color(R.color.colorPrimary))

    dialog.getButton(DialogInterface.BUTTON_NEGATIVE)
        .setTextColor(color(R.color.colorPrimary))
}

private fun showLogoutWindow() {
    val dialog = MaterialAlertDialogBuilder(this)
        .setTitle("Logout?")
        .setPositiveButton("Yes") { dialog, _ ->
            dialog.dismiss()
            AuthUI.getInstance()
                .signOut(this)
                .addOnCompleteListener {
                    toast("Signed out successfully!")
                    goToSplashActivity()
                }
        }
        .setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
```

```
            .show()

        dialog.getButton(DialogInterface.BUTTON_POSITIVE)
            .setTextColor(color(R.color.colorPrimary))

        dialog.getButton(DialogInterface.BUTTON_NEGATIVE)
            .setTextColor(color(R.color.colorPrimary))
    }
}
```

Клас AllDecksFragment.kt

```
class AllDecksFragment : Fragment(R.layout.fragment_all_decks) {

    private var currentPage = Page(1, emptyList())
    private val totalPages = 100
    private lateinit var deckAdapter: DeckAdapter

    private var _loadingDataState = LoadingDataState.LOADING
    private fun setLoadingDataState(state: LoadingDataState) {
        _loadingDataState = state
        when (state) {
            // loading
            LoadingDataState.LOADING -> {
                progress_bar_in_main.show()
                ll_nav_bottom_btn.hide()
                recycle_view_news.hide()
                ll_error_loading_data.hide()
            }
            // loaded
            LoadingDataState.LOADED -> {
                progress_bar_in_main.hide()
                ll_nav_bottom_btn.show()
                recycle_view_news.show()
```

```
            ll_error_loading_data.hide()
        }
        // failed
        LoadingDataState.FAILED -> {
            progress_bar_in_main.hide()
            ll_nav_bottom_btn.hide()
            recycle_view_news.hide()
            ll_error_loading_data.show()
        }
    }
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    initControl()
    configureRecycler()
    restoreData()
}

private fun initControl() {
    btn_reload_data.setOnClickListener {
        loadPageFromInternet(currentPage.pageNumber)
    }
}

private fun configureRecycler() {
    deckAdapter = DeckAdapter()
    recycle_view_news.layoutManager = LinearLayoutManager(context)
    recycle_view_news.adapter = deckAdapter
}

private fun restoreData() {
    if (currentPage.listOfDecks.isNotEmpty()) {
        showPage(currentPage)
```

```
            return
        }

        loadPageFromInternet(currentPage.pageNumber)
    }

    private fun loadPageFromInternet(pageNumber: Int) {
        setLoadingDataState(state = LoadingDataState.LOADING)

        HearthstoneApi.loadPage(requireActivity(), pageNumber) { page ->
            if (viewDestroyed()) return@loadPage

            if (page.listOfDecks.isEmpty()) {   // error occurred
                setLoadingDataState(state = LoadingDataState.FAILED)
            } else {    // all good
                currentPage = page
                showPage(page)
            }
        }
    }

    private fun showPage(page: Page) {
        setLoadingDataState(state = LoadingDataState.LOADED)

        setTitle("Page: ${page.pageNumber}/${totalPages}")

        deckAdapter.set(page.listOfDecks) { deckClicked ->
            requireActivity().simpleNavigate(
                DeckDetailsFragment(deckClicked)
            )
        }

        initLowerButtons(page)
    }
```

```
    private fun initLowerButtons(page: Page) {
        btn_previous.setOnClickListener {
            loadPageFromInternet(page.pageNumber - 1)
        }

        btn_next.setOnClickListener {
            loadPageFromInternet(page.pageNumber + 1)
        }

        btn_previous.setActive(active = page.pageNumber > 1)
        btn_previous.setActive(active = page.pageNumber < 100)
    }
}
```

Клас DeckAdapter.kt

```
class DeckAdapter : RecyclerView.Adapter<DeckViewHolder>() {

    private val listDecks = mutableListOf<Deck>()
    private lateinit var onClickListener: (Deck) -> Unit

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
DeckViewHolder {
        val view =
LayoutInflater.from(parent.context).inflate(R.layout.item_deck_preview, parent,
false)
        return DeckViewHolder(view)
    }

    override fun getItemCount(): Int {
        return listDecks.size
    }

    override fun onBindViewHolder(holderDeck: DeckViewHolder, position: Int) {
```

```
        holderDeck.bind(listDecks[position], onClickListener)
    }

    fun set(list: List<Deck>, onClickListener: (Deck) -> Unit) {
        this.listDecks.clear()
        this.listDecks.addAll(list)
        this.onClickListener = onClickListener
        notifyDataSetChanged()
    }
}
```

Клас DeckViewHolder.kt

```
class DeckViewHolder(val view: View) : RecyclerView.ViewHolder(view) {

    fun bind(deckItem: Deck, onClickListener: (Deck) -> Unit) {
        deckItem.bindToView(
            tv_title = view.row_tv_title,
            tv_gameClassText = view.row_tv_class,
            img_gameClassIcon = view.row_img_game_class,
            tv_dustText = view.row_tv_dust,
            tv_timeCreated = view.row_tv_time,
            tv_gameFormat = view.row_tv_standard_wild,
            img_gameFormatIcon = view.row_tv_standard_wild_icon
        )
        view.setOnClickListener {
            onClickListener(deckItem)
        }
    }
}
```

Інтерфейс DeckDetailsContract.kt

```
interface DeckDetailsContract {
    interface View {
        fun showLoadingScreen()
        fun showError(errorString: String? = null)

        fun showDeckPreview(deck: Deck)
        fun showDeckDetails(deckDetails: DeckDetails)

        fun getActivityInstance(): AppCompatActivity

        fun setFavoriteIconIcon(res: Int)
        fun setFavoriteIconClickable(clickable: Boolean)

        fun outputMessage(message: String)
    }

    interface Presenter {
        fun onInit(deck: Deck)
        fun onFavoriteIconClick()
        fun onCopyButtonClick()
        fun configureFavoriteIcon()
    }
}
```

Клас DeckDetailsFragment.kt

```
class DeckDetailsFragment(
    private val deck: Deck,
) : Fragment(R.layout.fragment_deck), DeckDetailsContract.View {

    private val presenter = DeckDetailsPresenter(this)

    private var menuStarItem: MenuItem? = null
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setHasOptionsMenu(true)
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    presenter.onInit(deck)
}

override fun showLoadingScreen() {
    if (viewDestroyed()) return

    layout_progress_bar.show()
    layout_failed.hide()
}

override fun showError(errorString: String?) {
    if (viewDestroyed()) return

    layout_failed.show()
    errorString?.let { layout_failed_text.text = it }
}

override fun showDeckPreview(deck: Deck) {
    if (viewDestroyed()) return

    setTitle(deck.title)
    deck.bindToView(
        tv_title = det_tv_title,
        tv_gameClassText = det_tv_deck_class,
        img_gameClassIcon = det_img,
        tv_dustText = det_tv_dust,
        tv_timeCreated = det_tv_time,
```

```
            tv_gameFormat = det_tv_format,
            img_gameFormatIcon = det_tv_format_icon
        )
    }

    override fun showDeckDetails(deckDetails: DeckDetails) {
        if (viewDestroyed()) return

        layout_progress_bar.hide()
        layout_failed.hide()

        // description
        if (deckDetails.description.isBlank()) {
            card_view_btn_description.hide()
        } else {
            card_view_btn_description.show()
            det_description.configureByDefault(det_description_layout,
det_img_arrow_up_down)
            det_description.text = deckDetails.description
        }

        // cards
        val cardAdapter = CardAdapter()
        recycle_view_deck.apply {
            layoutManager = LinearLayoutManager(context)
            adapter = cardAdapter
            isNestedScrollingEnabled = false
        }
        cardAdapter.set(deckDetails.listOfCards)

        // btn copy
        btn_copy_deck.setOnClickListener {
            presenter.onCopyButtonClick()
        }
    }
```

```
override fun getActivityInstance() = requireActivity() as AppCompatActivity

override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
    inflater.inflate(R.menu.menu_with_star, menu)
    menuStarItem = menu.findItem(R.id.btn_favorite)
    presenter.configureFavoriteIcon()
    super.onCreateOptionsMenu(menu, inflater)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    presenter.onFavoriteIconClick()
    return true
}

override fun setFavoriteIconIcon(res: Int) {
    if (viewDestroyed()) return

    menuStarItem?.setIcon(res)
}

override fun setFavoriteIconClickable(clickable: Boolean) {
    if (viewDestroyed()) return

    menuStarItem?.apply {
        isEnabled = clickable
        icon.alpha = if (clickable) 255 else 127
    }
}

override fun outputMessage(message: String) {
    if (viewDestroyed()) return

    requireContext().toast(message)
```

```
        }
    }
```

Інтерфейс DeckDetailsPresenter.kt

```
class DeckDetailsPresenter(val view: DeckDetailsContract.View)
    : DeckDetailsContract.Presenter {

    private lateinit var currentDeck: Deck

    private var isDeckInFavorite: Boolean = false
    set(value) {
        field = value
        view.setFavoriteIconIcon(if (value) R.drawable.ic_star_filled else
R.drawable.ic_star)
    }

    override fun onInit(deck: Deck) {
        currentDeck = deck

        view.showDeckPreview(deck)

        showDeckDetailsOrLoadThem()
    }

    private fun showDeckDetailsOrLoadThem() {
        currentDeck.deckDetails?.let {
            view.showDeckDetails(it)
            return
        }

        view.showLoadingScreen()

        HearthstoneApi.loadDeckDetails(view.getActivityInstance(), currentDeck) {
deckDetails ->
            if (deckDetails == null)
                view.showError()
```

```
            else {
                currentDeck.deckDetails = deckDetails
                view.showDeckDetails(deckDetails)
            }
        }
    }

    override fun onCopyButtonClick() {
        (view.getActivityInstance().getSystemService(Context.CLIPBOARD_SERVICE) as
ClipboardManager)
            .setPrimaryClip(ClipData.newPlainText("copy",
currentDeck.deckDetails!!.code))

        view.outputMessage("Copied to clipboard!")
    }

    override fun configureFavoriteIcon() {
        view.setFavoriteIconClickable(false)

        FirebaseHelper.isInFavoriteList(currentDeck) { exists ->
            isDeckInFavorite = exists
            view.setFavoriteIconClickable(true)
        }
    }

    override fun onFavoriteIconClick() {
        view.setFavoriteIconClickable(false)

        if (isDeckInFavorite) {
            FirebaseHelper.removeFromFavorite(currentDeck) { successful ->
                proceedResultFromFirebase(
                    resultIsSuccessful = successful,
                    isInFavorite = false,
                    messageWhenSuccessful = "Removed from favorite"
                )
```

```
            }
        } else {
            FirebaseHelper.saveDeckToFavorite(currentDeck) { successful ->
                proceedResultFromFirebase(
                    resultIsSuccessful = successful,
                    isInFavorite = true,
                    messageWhenSuccessful = "Added to favorite"
                )
            }
        }
    }

    private fun proceedResultFromFirebase(
        resultIsSuccessful: Boolean,
        isInFavorite: Boolean,
        messageWhenSuccessful: String
    ) {
        if (resultIsSuccessful) {
            isDeckInFavorite = isInFavorite
            view.outputMessage(messageWhenSuccessful)
        } else view.outputMessage("Failed")

        view.setFavoriteIconClickable(true)
    }
}
```