МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП‘ЮТЕРНИХ НАУК

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЕКТУВАННЯ МОБІЛЬНИХ ІГРОВИХ ЗАСТОСУНКІВ

Здобувач вищої освіти гр. ІН.м-12ан/2у Дмитро ПОРОШИН

Науковий керівник,
кандидат фізико-математичних наук,
асистент кафедри комп’ютерних наук Ольга ШУТИЛЄВА

В. о. завідувача кафедри
кандидат технічних наук, доцент
доцент кафедри комп’ютерних наук Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
В.о.зав.кафедри _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Порошину Дмитру Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проектування мобільних ігрових застосунків*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд технологій, що застосовуються для створення мобільних застосунків; 2) Аналіз конкурентів та постановка завдання; 3) Огляд технології створення прототипу; 4) Аналіз інструменту створення кросплатформних застосунків Flutter та Stacked архітектури ; 5) Розробка додатку та прототипу; 6) Огляд функцій та аналіз подальшого розвитку проекту.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд технологій, що застосовуються для створення мобільних доданків		
2.	Постановка задачі та аналіз конкурентів		
3.	Опис архітектури додатку та бази даних		
4.	Розробка додатку з використанням Flutter та Firebase		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 91 сторінки, 33 рисунка, 10 джерел літератури, 3 таблиці, 1 додаток.

Об’єкт дослідження – Інформаційна технологія проектування мобільних ігрових застосунків.

Мета роботи – проектування та розробка асистента для настільних рольових через реалізацію мобільного додатку за допомогою технології Flutter.

Результат – зроблено аналіз літератури, інструментів та підходів для розробки кросплатформних доданків. Зроблено огляд клієнт-серверної взаємодії та наведено основні її типи, проаналізовані основні переваги та недоліки. Зроблено дослідження у важливості та методах прототипування мобільних застосунків. На основі досліджень було обрано інструменти для вирішення поставленої задачі та створена програмна реалізація застосунку для ОС iOS та Android. Дана розробка дозволяє створювати власні шаблони для настільних рольових ігор, відповідає за збереження даних користувача такі як ігрові персонажі та створені шаблони. Додаток був реалізований за допомогою фреймворку Flutter на мові Dart та бази даних Firebase Firestore.

FLUTTER, DART, ANDROID, IOS, МОБІЛЬНИЙ ДОДАТОК
ФРЕЙМОРК, FIREBASE, ТЕХНОЛОГІЯ РОЗРОБКИ
КРОСПЛАТФОРМНИХ ЗАСТОСУНКІВ

ЗМІСТ

ВСТУП

Мобільні девайси або смартфони стали невід'ємною частиною нашого життя. Зараз вони всюди і без них практично неможливо існувати у сучасному суспільстві. Ці пристрою використовуються у безліч різних сфер життя. Дехто використовує свій смартфон для пошуку інформації, спілкування у соціальних мережах, для інших – це спосіб зайняти себе у дорозі або чекаючи в черзі.

Одна з найважливіших функцій мобільних пристрої – це допомога користувачеві у пошуку інформації. Сучасний смартфон бере на себе роль кишенькового асистента. За допомогою нього можна дізнатися короткий шлях додому, швидко зробити нотатки або скористатися інтернет банкінгом замість походу у відділення.

Не зважаючи на те що мобільні пристрої зараз широко розповсюджені, люди все рівно потребують фізичного контакту з іншими. Один з способів провести час є настільні рольові ігри, але деякі з них мають очевидні проблеми з тим, що потрібно відволікатися від гри на додаткові розрахунки або на редагування інформації.

Основним завдання даної роботи є створення асистента для настільних рольових ігор. Цей застосунок візьме на себе роль по розрахунку змінних, редагування даних та синхронізації цієї інформації серед гравців.

Актуальність роботи полягає у відсутності універсального додатку, де гравці зможуть реалізувати свої ігри з можливістю синхронізувати свої данні у хмарному середовищі та серед інших гравців.

1 ЛІТЕРАТУРНИЙ ОГЛЯД ЗА ОБРАНОЮ ТЕМАТИКОЮ РОБОТИ

По-перше, необхідно розглянути сучасні методи розробки мобільних додатків. Розглянути як нативні так і кроплатформні інструменти та визначити основні переваги та недоліки кожного з методів.

По-друге, розглянемо принцип роботи сервер клієнтської архітектури. В цьому пункті потрібно приділити увагу тому як клієнт взаємодіє з сервером, виділити основні практики розробки таких додатків.

На останок, зробити аналіз застосунків зі схожою тематикою. Це необхідно для того, щоб виділити позитивні та негативні риси конкурентів та визначити мінімальний функціонал, що потрібно реалізувати у майбутньому проекті.

1.1 Методи побудови мобільних застосунків

Розробка мобільних додатків – це процес розробки програмного забезпечення для мобільних пристроїв, включаючи мобільні телефони та планшети. Типова мобільна програма покладається на мережеве з’єднання для роботи з віддаленими обчислювальними ресурсами. Під час процесу розробки мобільних додатків виникає необхідність у написанні програмного коду, створенні пакетів програмного забезпечення, які можна встановити, розгорнути серверні служби та протестувати додаток на цільових пристроях.

На сьогоднішній день існують дві найпопулярніші платформи на сучасному ринку смартфонів – це платформа iOS і Android від Google.

Платформа iOS створена компанією Apple для забезпечення своєї флагманської лінійки мобільних телефонів iPhone [1]. З іншого боку, операційна система Android розроблена Google і використовується не лише на пристроях від Google, але й багатьма іншими OEM-виробниками, які випускають власні мобільні девайси [2].

Розробка для платформ iOS і Android мають деякі спільні риси, але між ними є відмінності, які варто обговорити. По-перше, кожен із них включає різні комплекти розробки програмного забезпечення (SDK) та інструментарій розробки. По-друге, Apple використовує iOS виключно для своїх пристроїв, тоді як Android від Google доступний для інших компаній, якщо вони відповідають вимогам платформи.

Розробляючи мобільні програми на ці дві платформах розробники можуть поширити свій додаток для більшої кількості пристроїв – це дозволяє збільшити свою цільову аудиторію.

Зараз існує три найпопулярніші підходи при створенні мобільних додатків, а саме:

- нативні мобільні програми;
- кросплатформні мобільні програми;
- прогресивні веб-додатки.

Кожен із цих методів розробки мобільних додатків має як переваги, так і недоліки. Розглядаючи підходи до розробки мобільних додатків, вам потрібно звернути увагу на бажану взаємодію з користувачем, обчислювальні ресурси та власні функції, необхідні для додатка, бюджет, часові рамки, часові обмеження та інтелектуальні ресурси, що доступні для підтримки додатка.

Нативні мобільні додатки написані на мові програмування та фреймворками, наданими власником платформи. Програма розроблена за допомогою нативних інструментів працюють безпосередньо на операційній системі пристрою, наприклад iOS і Android.

Переваги така підходу наступні:

- найкраща продуктивність з точки зору часу роботи;
- прямий доступ до API пристрою.

Але такий метод розробки має і свої недоліки:

- вищі витрати на розробку та підтримку програм;
- різна кодова база для кожної платформи.

На противагу нативним додаткам, кросплатформні програми можуть бути написані на різних мовах програмування та використовувати різні фреймворках. Ці інструменти дозволяють писати код, який буде працювати яка працювати на декількох операційних системах [3].

Переваги кросплатформної розробки:

- єдина база коду для кількох платформ;
- легко розвивати та підтримувати.

Однак, цей метод має очевидні недоліки:

- використання нативних бібліотеки та функцій стає складнішим;
- обмеження продуктивності через використання мостів.

Останнім спосіб за допомогою якого можливо розробити мобільні додатки це прогресивні веб-додатки. Прогресивні веб-додатки або PWA – це альтернативний підхід до традиційної розробки мобільних додатків, який не вимагає від користувача встановлювати та завантажувати додаток на девайс. Технічно PWA – це веб-програми, які використовують можливості браузера, такі як робота в автономному режимі, запуск фонових процесів і додавання посилання на головний екран пристрою, щоб забезпечити роботу додатку, подібну до звичайної нативної програми [4].

PWA застосунки мають такі переваги:

- Додаток доступний як для інтернету через браузер, так і як мобільна програма;
- Інсталяція не потрібна, додаток доступний за URL – адресою.

Основні недоліки це:

- Обмежена підтримка нативних функцій пристрою;
- Можливості програми обмежені в залежності від браузера.

1.2 Аналіз клієнт-серверної архітектури

Клієнт-сервер описує зв’язок між двома комп’ютерними програмами, клієнтська програма робить запит на обслуговування до іншої серверної програми (Рисунок 1.1). Це дозволяє кільком користувачам мати доступ до однієї бази даних одночасно, а база даних може зберігати велику кількість даних. Стандартні мережеві функції, такі як обмін електронною поштою, доступ до Інтернету та доступ до бази даних, базуються на моделі клієнт-сервер [5].

Клієнт ініціює запити до серверу. Клієнт використовується для запуску програм і доступу до даних, які зберігаються на сервері. Зазвичай, клієнтські додатки легше створити, ніж сервери, оскільки вони зазвичай не потребують спеціальних системних привілеїв, але клієнтські додатки, вимагають наявності графічного інтерфейсу.

Сервер виконує запити від клієнта. Часто, сервер має окремі привілеї, як наприклад читання або редагування бази даних, тому, розробляючи їх, потрібно бути обережним, щоб не передати несанкціонований доступ до даних клієнту.

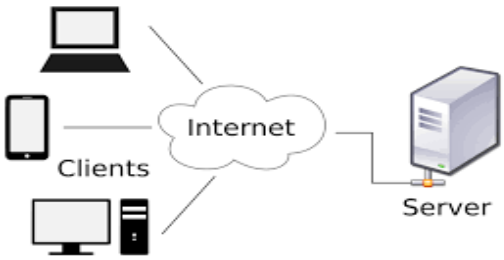

Рисунок 1.1 – Клієнт-серверна архітектура

Клієнт-серверна архітектура має безліч переваг. По перше, це централізація контролю, а саме доступ, ресурси та єдність даних контролюються виділеним сервером, щоб програма або неавторизований

клієнт не могли пошкодити систему. Це також дозволяє виконувати завдання оновлення даних або інших ресурсів [6].

По друге, дозволяє реалізувати правильне керування даними. Усі файли зберігаються в одному місці. Таким чином, керування та пошук файлів стає набагато легшим. Це відкриває можливість до резервного копіювання та відновлення. Створити резервну копію легко, оскільки всі дані зберігаються в одній базі. У разі поломки деякі дані можуть бути втрачені, і за допомогою резервного копіювання їх можна відновити. У той час як у однорангових обчисленнях ми повинні робити резервні копії на кожній робочій станції.

Архітектура відкриває можливості до масштабованості. Індивідуально можна покращити продуктивність клієнтів і серверів. Також доступ до даних можна отримати з різних платформ, як наприклад з браузера, мобільного додатка або десктопної програми.

Сервер може встановлювати правила, що визначають безпеку та права доступу, для того щоб сторонні програми не змогли отримати доступ до файлів.

У клієнт-серверної архітектури, є свої недоліки. Такі системи зазвичай є більш дорогими, адже виникає необхідність у хостингу. Налаштування серверу вимагає окремих навичок та вмінь.

Клієнт-серверна архітектура – це обчислювальна модель, у якій сервер розміщує, надає та організовує більшість ресурсів і послуг, які отримує клієнт. Ця архітектура має один або багато клієнтських комп’ютерів, підключених до центрального сервера через мережу Інтернет або локально.

Клієнт-серверна архітектура поділяється на декілька типів. Перший і найпростіший тип – це дворівнева архітектура. У цьому типі архітектури робоче навантаження розподіляється між сервером (хостом системи) і клієнтом (на якому розміщено інтерфейс користувача) (рис. 1.2). Ці компоненти можуть бути розташовані на окремих комп’ютерах, але абсолютної вимоги до цього немає, логічно розділені рівні, за деяких

умов(наприклад, під час розробки або тестування) можуть бути розміщені на одному комп’ютері.

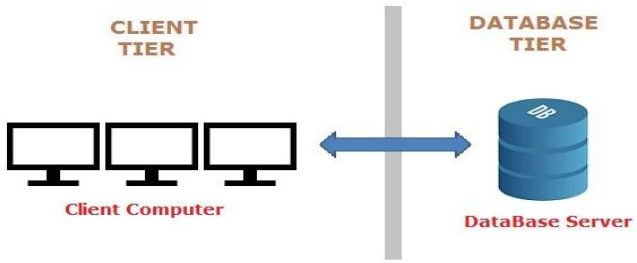

Рисунок 1.2 – Дворівнева архітектура

Така система має високу продуктивність так як сервер бази даних і бізнес-логіка фізично близькі, що забезпечує вищу продуктивність. Таку систему зазвичай легше реалізувати, через те, що не потрібно розробляти окремий сервер для отримання даних, а вся інформація отримується з бази, це також дає можливість швидко створювати прототипи програми.

Однак, дворівневий дизайн є хорошим рішенням, коли кількість користувачів зазвичай невелика, але при збільшенні користувачів виникають обмеження. У міру зростання юзерів продуктивність починає падати. Це пов’язано з тим, що кожен користувач має власне з’єднання, і навіть якщо запити не виконується, сервери мають підтримувати всі ці з’єднання.

Дворівневі додатки можуть мати проблеми з безпекою, бо кожен користувач повинен мати власний індивідуальний доступ до бази даних отримувати всі доступи, які необхідні для запуску клієнтської програми.

Через те, що клієнт напряму взаємодіє з базою, то при необхідності мігрувати на нову СУБД виникають проблеми та складнощі.

Щоб позбутися цих недоліків, було введено додатковий рівень. Така система називається трирівневою, вона призначена подолати обмеження дворівневої архітектури. Рівень між клієнтом та базою даних використовується для реалізації додатків і керування базою даних (рис.1.3), як і у дворівневій моделі рівні можуть бути реалізовані на різних фізичних

машинах або декілька рівнів можуть бути розміщені на одній машині в залежності від потреб.

У трирівневому дизайні такі робочі процеси як вхід у систему, доступу до інформації, зберігання даних системи та інтерфейс користувача розробляються та підтримуються як самостійні модулі на окремій платформі.

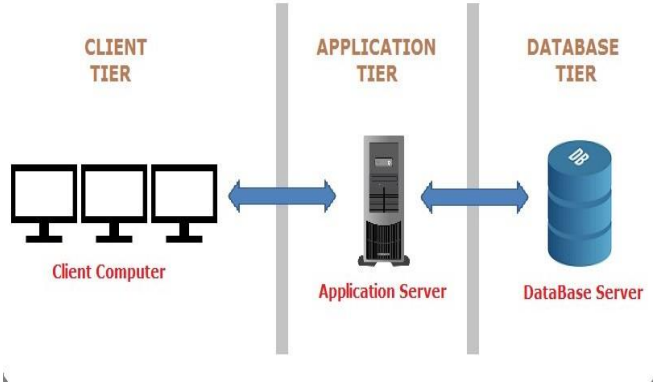

Рисунок 1.3 – Трирівнева архітектура

Клієнтський рівень займає найвищий рівень і відображає інформацію для користувача. Цей рівень взаємодіє з іншим рівнем, роблячи запити в на зміну, редагування та інше.

Рівень серверної програми або він також називається рівнем логіки. Він контролює запити від клієнтської програми, отримуючи інформацію від бази даних та виконуючи їх обробку.

Рівень бази даних відповідальний за збереження інформації. Цей рівень приймає запити від серверної програми й надає необхідні дані назад на логічний рівень для обробки, а потім, ця інформація передається назад до користувача.

Ця архітектура позбувається недоліків дворівневого дизайну. Вона забезпечує цілісність даних. Дублювання даних стає практично неможливим коли дані проходять через трирівневу базу даних для оновлення, що забезпечує її надійність.

Покращена безпека також є позитивною рисою архітектури. Дані стають більш захищеними завдяки розміщенню бізнес-логіки на централізованому

сервері, оскільки клієнт не взаємодіє з базою даних безпосередньо. Це дозволяє приховати структуру бази даних, щоб уникнути небажаних будь-які змін. Обмеженням даної системи є необхідність у створенні додаткового рівня та забезпечення з'єднання між усіма трьома рівнями.

Останнім типом клієнт-серверною реалізації є N-рівнева архітектура, або також багаторівневою архітектурою. Це клієнт-серверна архітектура, яка за своєю структурою схожа на трирівневу, але може мати декілька серверних реалізацій або баз даних (рис.1.4).

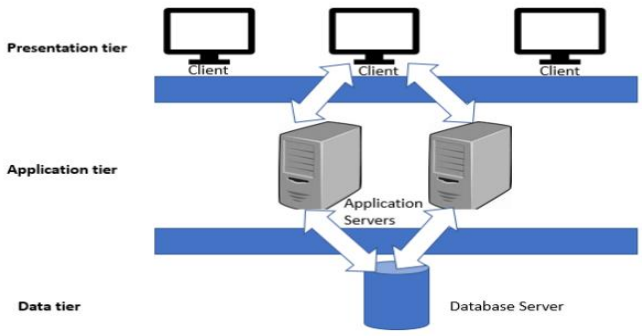

Рисунок 1.4 – Трирівнева архітектура

Цей рівень має всі ті переваги, що й попередній, але він також дозволяє полегшити масштабування системи, оскільки не впливаючи на інші рівні, можна додати більше програмних ресурсів до кожного рівня.

Це відкриває можливість для повторного використання компонентів. Оскільки програма розділена на незалежні рівні, ви можете комфортно використовувати кожен рівень для інших проектів програмного забезпечення. Наприклад, якщо ви хочете використовувати ту саму програму, але для іншого набору даних, ви можете просто рівень логіки та клієнта, а потім створити новий рівень даних.

Багаторівнева архітектура є найбільш затратною. Через велику кількість компонентів на рівнях цю складну структуру важко реалізувати або підтримувати.

Отже, клієнт-сервер відноситься до популярної моделі комп’ютерної мережі, яка використовує як клієнтські пристрої, так і сервери, кожен з яких має певні функції. Модель клієнт-сервер можна використовувати як у глобальній мережі, так і у локальній мережі (LAN). Клієнтські сервери «обслуговують» клієнтські комп’ютери цілодобово, надаючи ресурси для роботи файлів, друку, електронною пошти та інше. Малий бізнес, очевидно, може отримати вигоду від мережевої інфраструктури.

1.3 Аналіз конкурентів

На даний момент додаток має декількох конкурентів на ринку. Загалом, це закордонні розробки. Одними з найбільш популярних є «Character Sheet» та «D&D Beyond».

Перед початком аналізу конкурентів потрібно вибрати основні критерії які будуть виступати в якості порівняння різних програмою. Першим критерієм обираємо дизайн на UX застосунку. Це важливо у кожній програмі, бо від цього напряму залежить те, наскільки користувачам буде зручно користуватися додатком. Інтерфейс повинен бути сучасним, не перевантаженим, але в той же час зрозумілим та інтуїтивним. Наступним критерієм обираємо кастомізації. Цей критерій буде відповідати за те наскільки користувач зможе кастомізувати функції додатку для створення своєї власної рольової настільної гри. Останнім критерієм буде виступати допомога гравцеві. В цьому критерії розглянемо наскільки додаток справляється зі своєю задачею асистента та які дії гравців додаток дозволяє автоматизувати.

Першим розглянемо програму «Character Sheet». Ця програма є найпростішою з представлених додатків.

Головне вікно програми складається з кнопки налаштувань, кнопки створення нового персонажа та списку наявних (рис. 1.5).

Рисунок 1.5 – Головна сторінка програмі «Character Sheet»

Додаток має функції створення свого власного темплейту для персонажів гравців. Ця функція має широкий вибір кастомізації. Вона дозволяє створювати власні поля та атрибути, додавати залежності між навичками персонажа та редагувати формулу (рис.1.6).

Рисунок 1.6 – Створення свого темплейту у програмі «Character Sheet»

Також, навіть кастомний темплейт має обов’язкові властивості які неможливо видалити, що є мінусом даної програми, бо вони не завжди є потрібними (рис.1.7). Також треба зазначити що власні темплейти є платною опцією, але це не можна назвати однозначною негативною рисою, адже сам додаток є безкоштовним.

Програма має достатній функціонал для допомоги гравцеві. Вона дозволяє вести та редагувати інвентар гравця, також надає доступ до ігрового магазину, дозволяє автоматично керувати валютою гравця (рис.1.8). Є можливість обирати предмети з готового списку, або додавати свої власні, але ця функція є платною.

Рисунок 1.7 – Створення свого темплейту у програмі «Character Sheet»

Рисунок 1.8 – Інвентар гравця програми «Character Sheet»

«Character Sheet» дає можливість створювати опис свого героя, додавати йому історію (рис. 1.9). Дуже зручно робити нотатки у середині програми, а не перемикатися між вікнами.

Рисунок 1.9 – Сторінка з нотатками програми «Character Sheet»

Негативним фактором програми можна назвати інтерфейс. Він гармонічно стилізований під фентезійну тематику, але це викликає деякі складнощі у розумінні програми. Деякі кнопки є неочевидними і виглядають як елементи оформлення, а не інтерактивний елемент, а деякі накладаються одна на іншу, що викликає дискомфорт.

Основні переваги:

- Можливість кастомізації та створення свої теплейтів;
- Достатній функціонал, що здатний допомогти гравцю;
- Відсутність реклами;
- Основний функціонал доступний безкоштовно.

Основні недоліки:

• Незручний та непродуманий дизайн, що здатний вести в оману користувача.

Отже, «Character Sheet» – це непогана програма, яка може виступати асистентом гравцеві, так як має весь функціонал для цього, але інтерфейс програми може викликати дискомфорт під час користування.

Наступним розглянемо додаток «D&D Beyond». Ця програма базується на правилах D&D та призначена для допомоги гравцям, що використовують офіційні редакції D&D. Додаток вимагає реєстрації для користування.

Дизайн програми зручний та лаконічний, у дизайні присутні графічні елементи але вони не відволікають від основних функцій. Додаток має бібліотеку в якій можна прочитати будь яку інформацію, про класи, властивості та інше (рис.1.10). Якщо в бібліотеці не вийшло знайти необхідну статтю, то застосунок має функцію глобального пошуку, або пошуку по окремим категоріям для зручності.

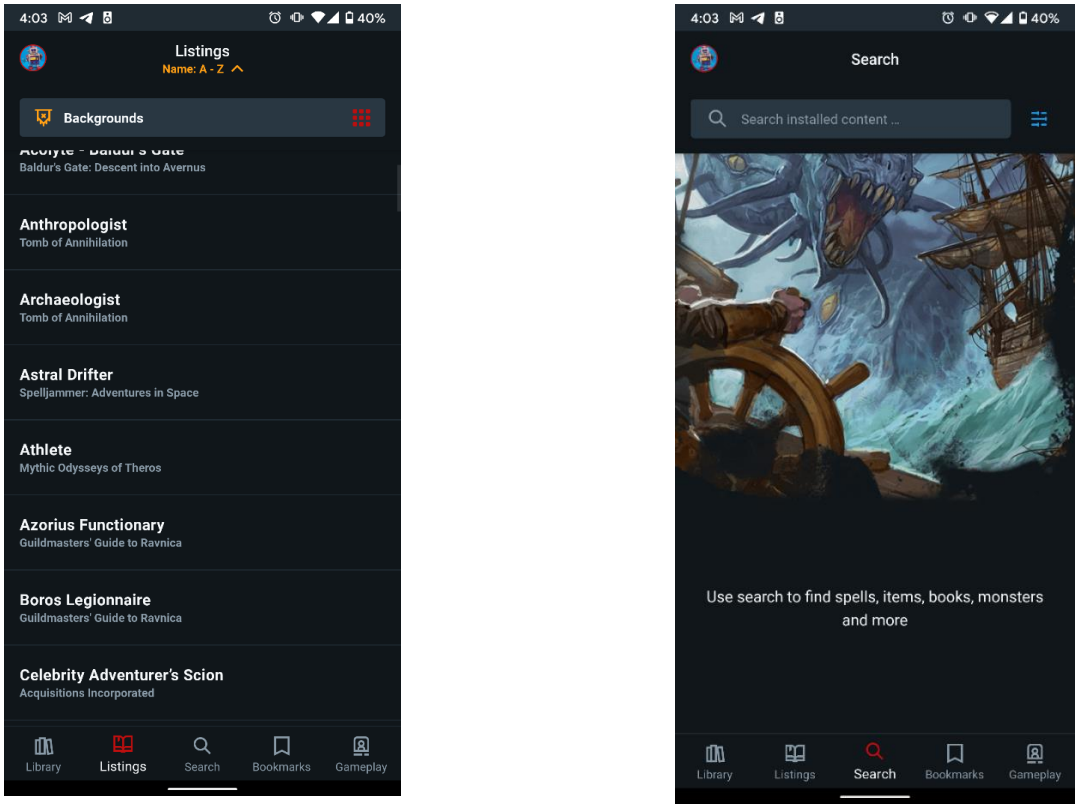

Рисунок 1.10 – Пошук у програмі «D&D Beyond»

Правила та ігри до них, можна завантажити з окремої вкладки (рис.1.11). Більшість ігор та сюжетів є платними, але базові правила D&D та гайд буки до них можна завантажити безкоштовною. Це дозволяє спробувати функціонали програми, перед покупкою платних доповнень.

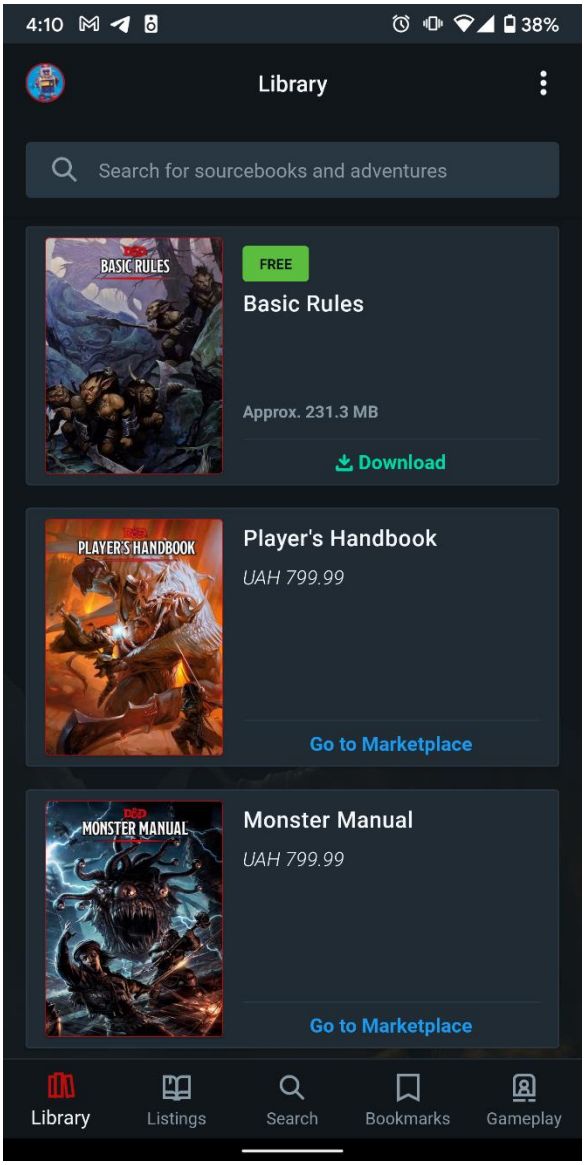

Рисунок 1.11 – Вкладка завантажень програми «D&D Beyond»

«D&D Beyond» дає можливість створити власного персонажа. Цей процес відбувається у окремому вікні через WebView, що викликає деякі складнощі та незручності. Сам процес створення героя інтуїтивний та покроковий (Рисунок 1.12). Функція створення свого власного теплейта відсутня, можна використовувати лише базовий шаблон за правилами D&D.

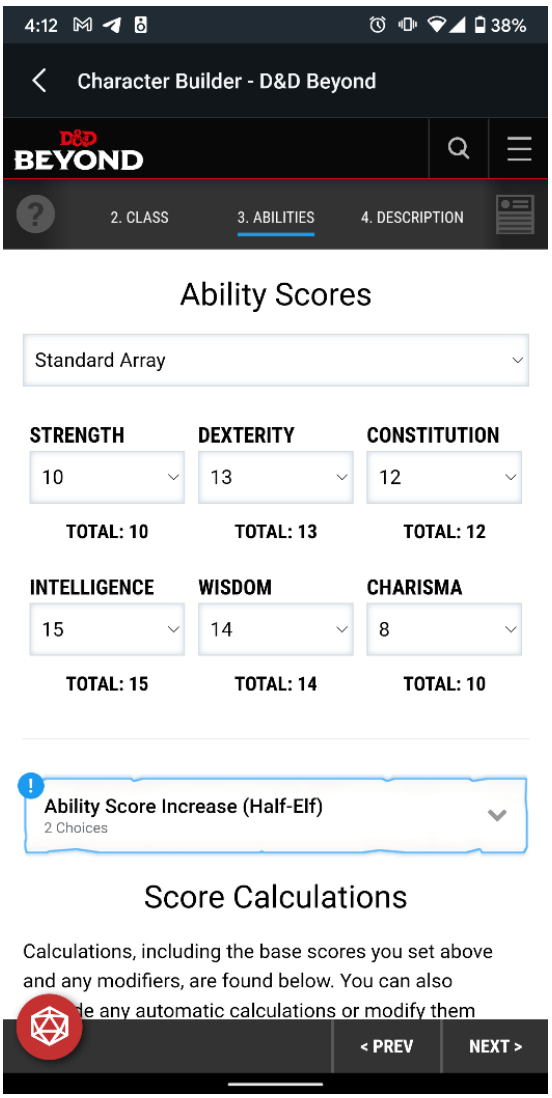

Рисунок 1.12 – Створення персонажа у програмі «D&D Beyond»

Після створення свого героя, юзер може переглянути всі необхідні данні в середині додатку. На вікні персонажа є всі наявні функції, такі як опис, нотатки, інвентар та властивості (рис.1.13). Однією з переваг додатку можна виділити можливість кинути кубик різних типів.

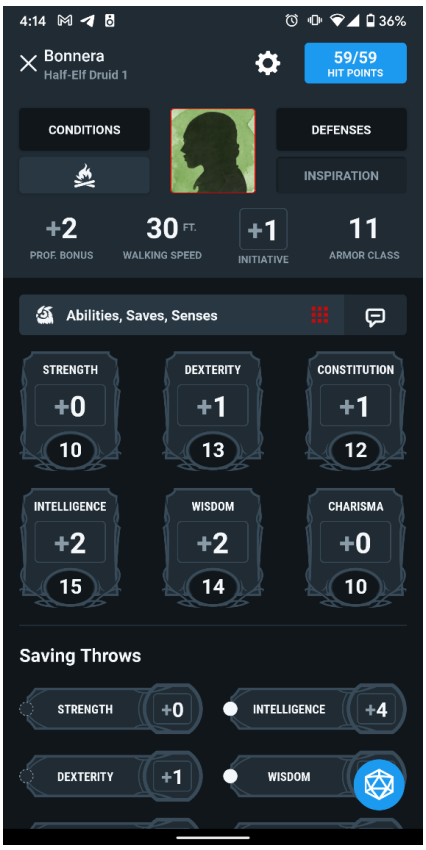

Рисунок 1.13 – Вікно персонажа у програмі «D&D Beyond»

Основні переваги:

- Зручний та сучасний інтерфейс;
- Функціонал є достатнім для гри;
- Бібліотека та пошук;
- Можливість рандомної генерації чисел.

Основні недоліки:

- Відсутня можливість створення свого шаблону;
- Створення персонажа браузер.

Отже, “D&D Beyond” – програма з гарним інтерфейсом яка добре підходить під ігри які базуються на стандартних правилах рольових ігор D&D. Відсутність кастомних шаблонів не дозволяє використовувати цей додаток для своїх власних настільних ігор та правил.

1.4 Постановка задачі

Рольові настільні ігри є досить популярним видом розваг. Такі ігри здатні розвинути у гравців уяву та креативність, адже більшість дій у грі потрібно додумувати.

Хоча зараз існує безліч різних настільних RPG зі своїми правилами та сюжетами, але часто виникає бажання створити свою гру з різними правилами. У цьому випадку гравцям доводиться всі розрахунки робити власноруч, що у більшості випадків відволікає від самого процесу гри.

Тому виникає потреба у універсальному додатку, за допомогою якого можна зробити власний шаблон для гри, вести розрахунки та зберігати всі данні на відділеному сервері для синхронізації між гравцями.

Програма повинна містити такі функції:

- Створення власних шаблонів для ігор за допомогою готових конструкцій;
- Синхронізація даних між пристроями, збереження персонажів користувача;
- Можливість редагувати властивості героїв.

Для досягнення поставленої мети треба вирішити наступні задачі:

- Провести аналіз предметної області;
- Обрати засоби для вирішення поставленої задачі;
- Виконати прототипування дизайну майбутнього додатка
- Зробити програмну реалізацію.

2 МЕТОДИКА ВИКОНАННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Прототипування мобільних застосунків

Існує безліч програм для прототипування комп’ютерних програм. Вони можуть відрізнятися за функціями які має програма, за призначенням та формою розповсюдження. Більшість програм мають однакові функції, але деякі з них, які націлені на професійне використання мають більший список можливостей, але водночас й більш складний користувацький інтерфейс [7].

Для створення прототипу даної програми було обрано інструмент Figma. Figma – це універсальний інструмент, який спрощує співпрацю та доступність для UX-дизайнерів, розробників та будь-кого іншого в команді за допомогою хмарної платформи на основі браузера [8]. Якщо ви раніше працювали зі Sketch, ви побачите, що Figma має схоже відчуття, що полегшує роботу з ним.

Figma орієнтована не лише на веб-дизайн, її пріоритетом також є розробка прототипів для мобільних платформ. Можа використовувати гнучкі стилі Figma, щоб контролювати вигляд тексту, створювати спільні елементи, використовувати сітки та інших елементи програми. Figma у своїй базі має достатньо функцій для створення власних прототипів, але для розширення функцій існують різноманітні корисні плагіни, такі як Autoflow для ілюстрації потоків користувачів, Figmotion для створення анімації та багато інших [9].

Основними критеріями за використання саме цієї програми було її доступність та простота у використанні. Figma є безкоштовною програмою, увесь функціонал для прототипування доступний одразу без додаткової плати. Також, програма має необхідний набір функцій та неперевантажений інтерфейс користувача. Figma доступна як окрема програма для персональний комп’ютерів, але також можливе використання веб-версії, адже прогрес зберігається автоматично.

2.2 Кросплатформна розробка мобільних застосунків

Основним інструментом написання мобільного застосунку обрано Flutter. Flutter – це між платформний набір інструментів із відкритим вихідним кодом, який використовується для створення програм для мобільних ОС, вебу та комп'ютера з однаковою кодовою базою. Це означає, що розробники пишуть код один раз і застосовують його на всіх платформах – iOS, Android, веб-платформі, macOS і навіть Wear OS [10].

Flutter не є єдиним кросплатформний фреймворк. Xamarin, React Native, Cordova – вони також дозволяють розробникам написати код один раз і використовувати його на кількох платформах, але Flutter поєднує в собі якість нативних програм із гнучкістю кросплатформної розробки, що відрізняє цей фреймворк. Flutter має такий самий вигляд, як нативна програма. Він відображає інтерфейс користувача з нуля, а не працює як оболонка поверх компонентів рідного інтерфейсу, як це роблять інші фреймворки.

Якщо подивитися на популярність фреймворку на GitHub, то можна побачити що розробники – як індивідуальні, так і з IT компаній будь-якого розміру – відносяться позитивно до Flutter і роблять свій внесок у його покращення.

Зараз, все більше компаній починають переносити існуючи або створювати нові проекти саме з використанням цієї технології. Ось деякі приклади, коли компанії обирають основним інструментом Flutter:

- Вони мають обмежений час і/або бюджет для створення програм;
- Хочу дві програми за ціною однієї;
- Створення та обслуговування старих нативних додатків коштує занадто дорого;
- Мали проект на RN/Cordova/Ionic/Xamarin, але шукав кращий варіант.

Flutter — це не просто фреймворк, це повний SDK для створення програм, він містить усе необхідне для створення інтерфейсу користувача (UI),

включаючи віджети для Material Design для систем на базі Android і Cupertino для iOS систем. Вони дозволяють розробникам легко відтворювати інтерфейс, що буде виглядати та працювати як нативний.

Програми, створені за допомогою Flutter, використовують Dart. Dart — це об’єктно-орієнтована мова програмування з відкритим кодом, також створена Google [11]. Ця мова спочатку пропонувалась як одна з альтернатив JavaScript, але дуже швидко втратила популярність.

Поряд з Dart, у ядрі Flutter є високошвидкісний C++ (рис.2.1). Отримана програма створює таку високу частоту кадрів, що виглядає як нативна.

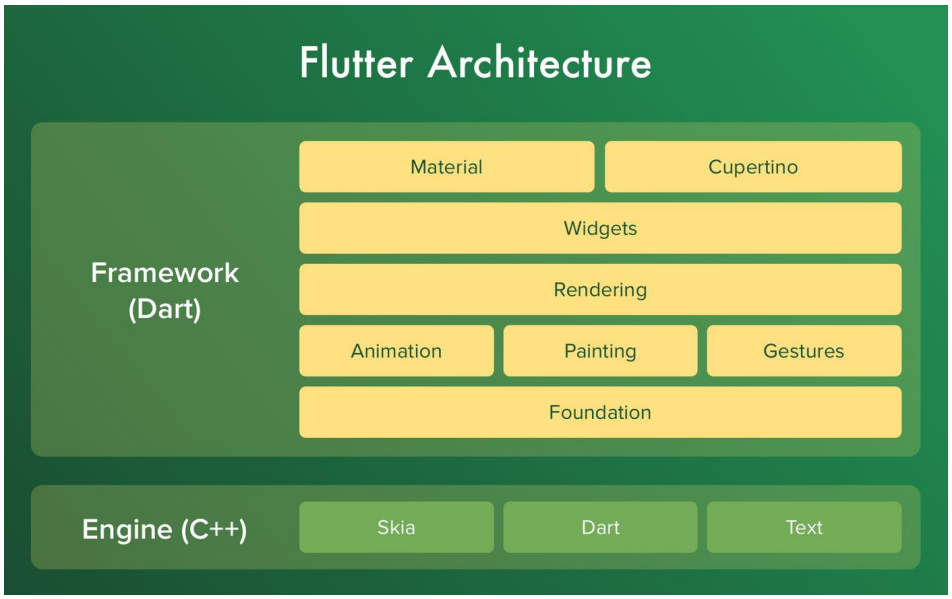

Рисунок 2.1 – Архітектура фреймворку Flutter

Dart використовує графічний механізм Skia, що створений на мові C++, який має всі протоколи, композиції та канали. Завдяки вбудованому движку Skia, Flutter мінімізує взаємодію з компонентами ОС — і йому не потрібен міст, який уповільнює роботу програми, як у React Native (рис. 2.2).

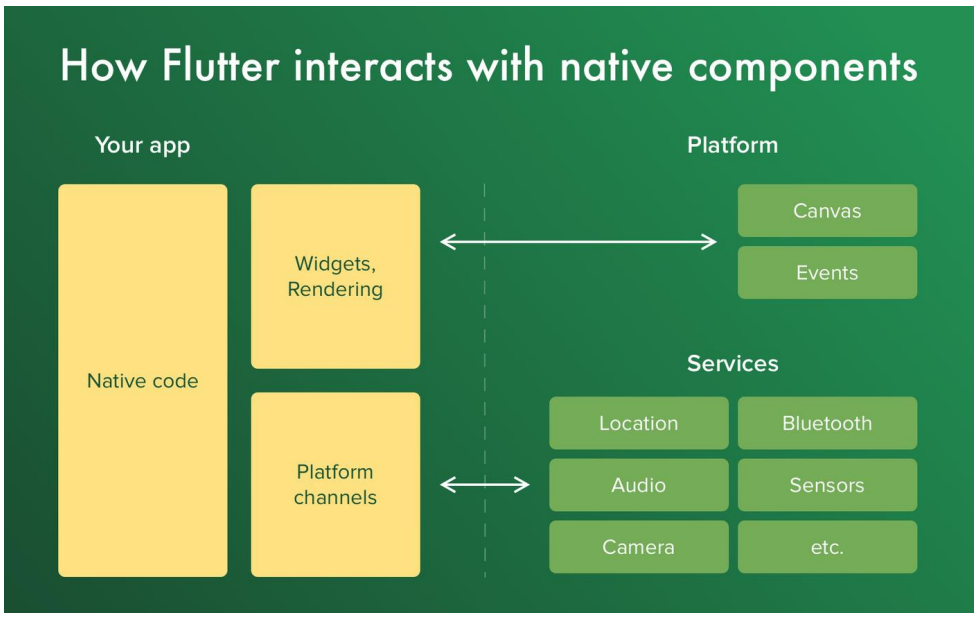

Рисунок 2.2 – Схема роботи Flutter з нативними компонентами

Натомість, подібно до ігрових движків, таких як Unity або Unreal, Flutter малює інтерфейс самостійно. Кнопки, текст, медіа-елементи, фон – все це малюється всередині графічного движка Skia [12].

На даний момент, Flutter має низку переваг, першою з яких є те, що додатки Flutter є бюджетними. Програми, розроблені у Flutter, не потребують багато часу для створення, і тому вони не такі дорогі, як нативні застосунки. Він ідеально підходить для стартапів, створення MVP, коли вам потрібно швидко перевірити свою бізнес-модель. Вартість розробки програми Flutter буде нижчою порівняно з нативною розробкою для двох платформ.

Другою й основною перевагою є швидкодія інструменту. Як було вже згадано раніше, архітектура фреймворку забезпечує продуктивність 60 кадрів/с або 120 кадрів/с на пристроях, які підтримують оновлення екрану 120 Гц. Це можливо за рахунок того, що програмам, створеним у Flutter, не потрібен міст для взаємодії з рідними компонентами, як це потрібно іншим фреймворкам.

Щоб отримати вигляд і продуктивність нативних програм, розробники мобільних пристроїв створювали дві окремі програми на різних мовах – Swift/Objective-C для iOS або Java/Kotlin для Android. Flutter дозволяє

програмістам використовувати єдину кодову базу, об’єднувати свої команди, зменшувати ризики та пришвидшувати час виходу на ринок. І все це, отримуючи переваги оригінального вигляду та продуктивності.

Серед переваг Flutter для самих розробників є функція під назвою Hot Reload. Коли розробник натискає кнопку перезавантаження, усі зміни коду миттєво відображаються на гаджетах, емуляторах і симуляторах. Якщо програмісти змінюють деякий код у програмах, створених за допомогою Flutter, після перекомпіляції, їм не потрібно повертатися назад або вручну відтворювати стан, щоб побачити, що змінилося. Hot Reload дає розробникам Flutter можливість надавати більше функцій за менший час. Крім того, простіше додавати нових функцій, щоб підтримувати активність користувачів.

Flutter не ідеальний. Він розвивається дуже швидко, але створення міцної основи, яка охоплює все, що роблять нативні платформи — це справа не одного дня.

Хоча Flutter стрімко набирає популярність, але мова Dart залишається вкрай не популярною поза межами фреймворку. Dart – швидка, об’єктно-орієнтована мова програмування, що підтримується величезною технологічною компанією, але Dart не може конкурувати з такими гігантами, як Java, Objective-C або Kotlin. Небагато розробників вибирають Dart, тому існує обмежена кількість компаній, які пропонують Flutter як альтернативу іншим інструментам.

Розмір програм написаних з використанням даної технології не є оптимальним. Базовий додаток, створений Flutter, тепер має 4,7 МБ на Android і 10,9 МБ на iOS [13]. Це багато порівняно з базовими програмами Java і Kotlin. Тим не менш, інші кросплатформні фреймворки мають таке ж слабке місце: мінімальна програма, написана на Xamarin, займає ~16 МБ, а React Native – 7 МБ.

Проаналізувавши переваги та недоліки технології Flutter, можна зробити висновок, що цей інструмент підходить для вирішення даної задачі, коли потрібно за мінімальну кількість часу реалізувати додаток для двох платформ,

що матиме широкий спектр функцій. Недоліки цього інструменти є незначними для даної роботи.

2.3 Розробка бази даних

В якості основного сховища даних обрано набір інструментів Firebase. Він включає в себе повний перелік сервісів які дозволяють завантажувати файли, зберігати дані, працювати з аналітикою та інші. За збереження інформації відповідає сервіс Firestore.

Firestore – це NoSQL база даних, створена для автоматичного масштабування, високої продуктивності та простоти розробки програм. Хоча інтерфейс Firestore має багато тих самих функцій, що й традиційні бази даних, як NoSQL база даних він відрізняється від них способом опису зв'язків між об’єктами даних [14].

У Firestore одиницею зберігання є документ. Документ — це невеликий запис, який містить поля, які зіставляються зі значеннями. Кожен документ ідентифікується назвою. Кожен документ містить набір пар ключ-значення. Firestore оптимізовано для зберігання великих колекцій, що складаються з невеликих документів. Усі документи повинні зберігатися у колекціях. Документи можуть містити під колекції та вкладені об’єкти, обидва з яких можуть містити примітивні поля, як рядки, або складні об’єкти, як списки (рис.2.3).

Рисунок 2.3 – Приклад документу Firestore

Документи Firestore за структурою схожі на формат JSON. Насправді вони в основному є [15]. Існують деякі відмінності (наприклад, документи підтримують додаткові типи даних і обмежені розміром до 1 МБ), але загалом можна розглядати документи як полегшені записи JSON.

Документи живуть у колекціях (Рисунок 2.4), які є просто контейнерами для документів. Наприклад, може бути колекція користувачів, яка містить різних користувачів, кожен з яких представлений документом. Колекції та документи створюються неявно у Firestore. Просто призначте дані документу в колекції. Якщо колекція або документ не існує, Firestore створює їх.

Firestore не має схем, тому є повна свобода вибору, які поля розмістити в кожному документі та які типи даних зберігати в цих полях. Усі документи в одній колекції можуть містити різні поля або зберігати різні типи даних у цих полях. Однак доцільно використовувати однакові поля та типи даних у кількох документах, щоб легше запитувати документи.

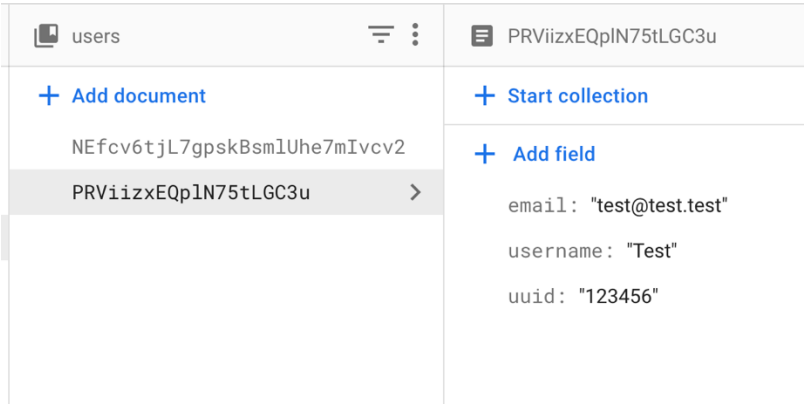

Рисунок 2.4 – Приклад колекції у Firestore

Назви документів у колекції є унікальними. Можна надати власні ключі, наприклад ідентифікатори користувачів, або дозволити Firestore автоматично створювати для документу випадкові ідентифікатори.

2.4 Прототипування інтерфейсу користувача

Проектування майбутнього додатку є важливим етапом у створенні будь якого застосунку. Від цього етапу залежить те наскільки програма буде легко підтримуватися у майбутньому, процес інтеграції нових функцій, наскільки програма буде надійною й навіть час практичної реалізації. Проектування застосунку у даній роботі складається з наступних етапів:

- Створення прототипу інтерфейсу користувача;
- Огляд архітектури, що буде використана у додатку;
- Створення та опис об’єктів бази даних.

Прототип було створено й розроблено у програмному середовищі Figma. Перши етапом у розробці прототипу – було створення базових елементів з яких він буде складатися (рис.2.5).

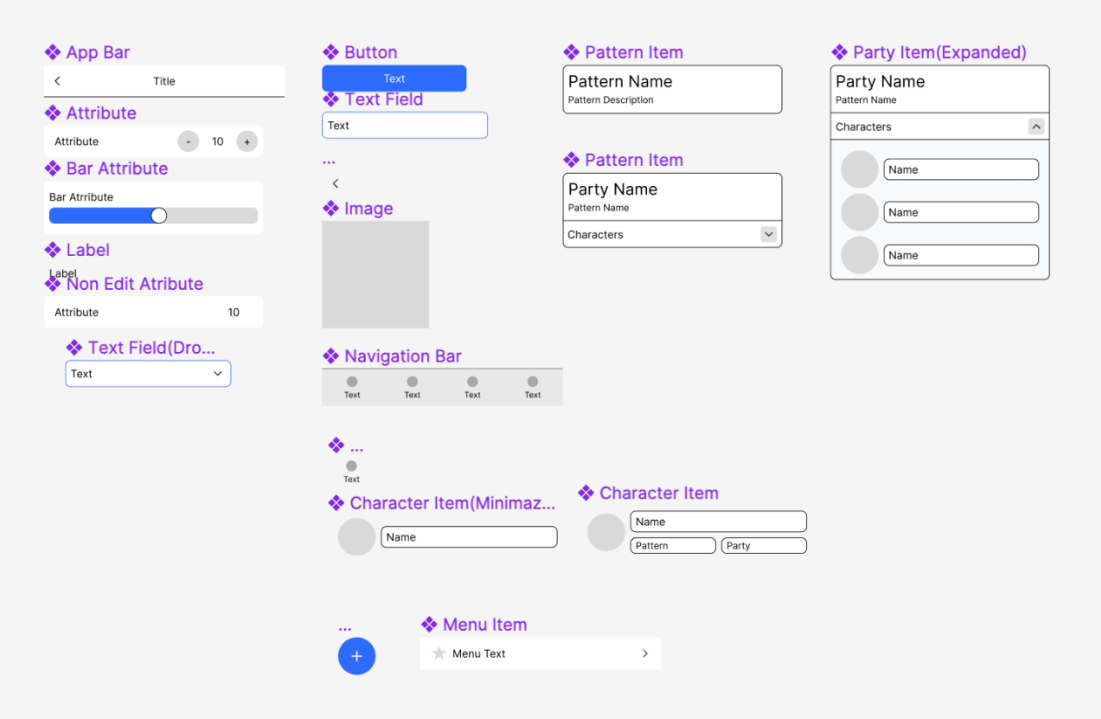

Рисунок 2.5 – Базові елементи прототипу

Набір включає в себе такі елементи як: текстові поля, кнопки, елементи списків, панель навігації та інші. Ці всі елементи були використані для створення прототипу застосунку.

Перши вікном прототипу є Splash Screen. Далі, користувач має вибір з двох опцій увійти в існуючий акаунт або створити новий. При натисканні на кожну з кнопок відкривається відповідне вікно (рис.2.6).

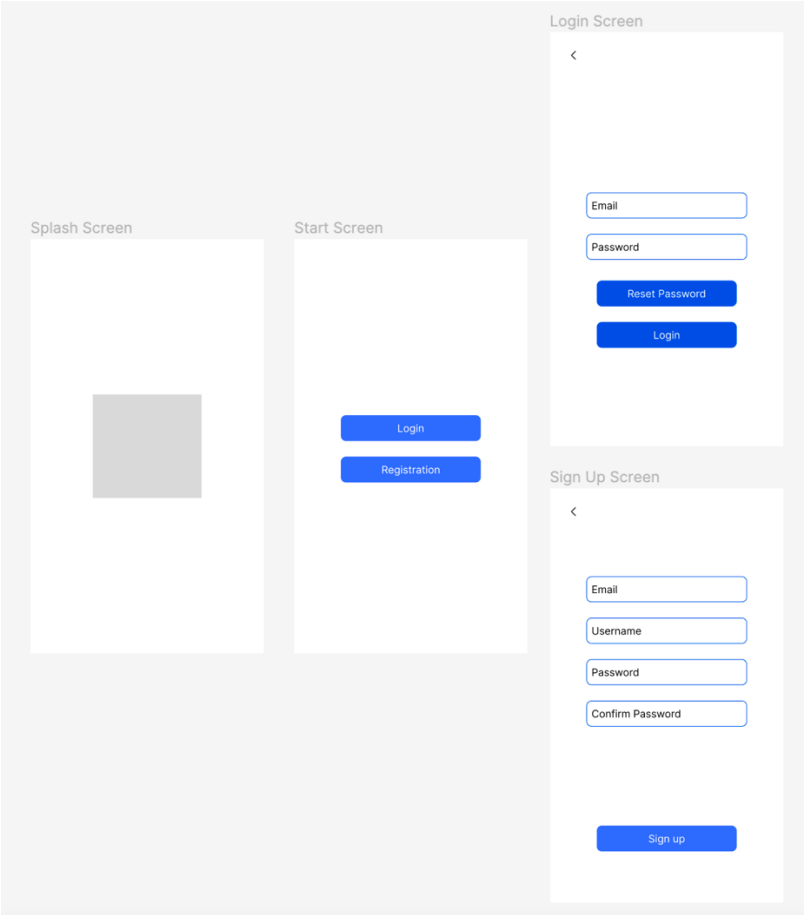

Рисунок 2.6 – Прототип авторизації в додаток

Після виконання входу у додаток, користувач має головне вікно програми на якому доступна навігаційна панель з чотирма оптаціями: Characters, Parties, Patterns, Account (рис.2.7).

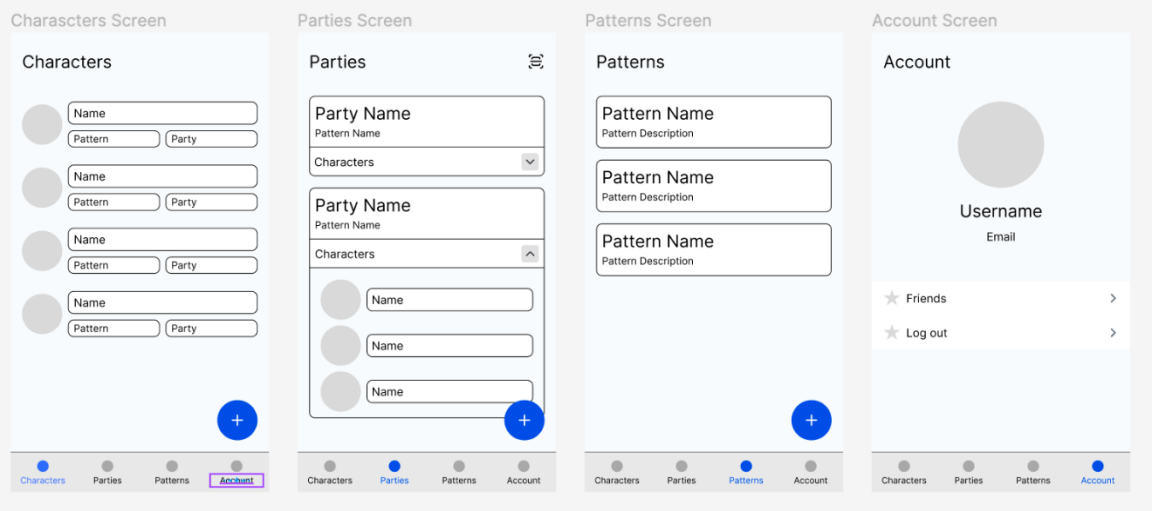

Рисунок 2.7 – Головне вікно програми

На вкладці Characters користувач матиме змогу створити власних персонажів з використанням різних доступних для нього шаблонів та елементів (рис.2.8).

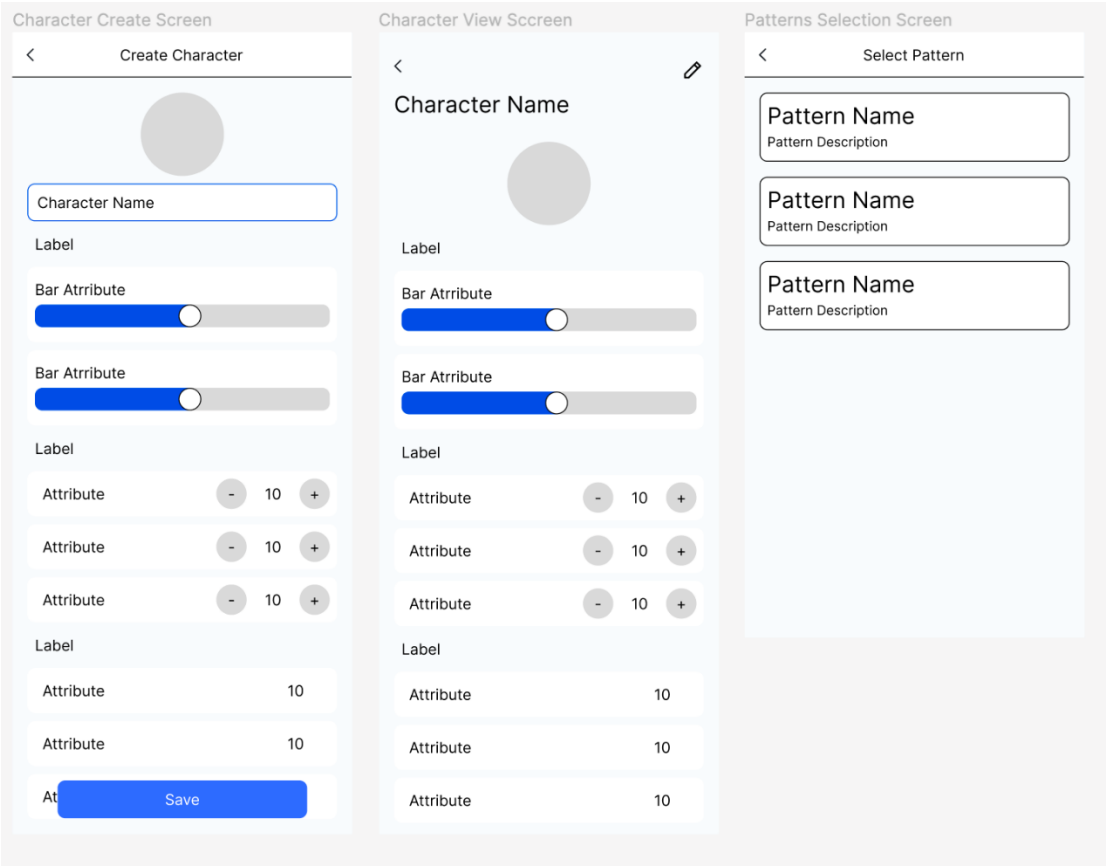

Рисунок 2.8 – Створення та редагування персонажа

Вікно Patterns дозволятиме користувачу переглядати або редагувати паттерни для різних ігор. Юзер також може створити свій власний шаблон використовуючи готові елементи (рис.2.9).

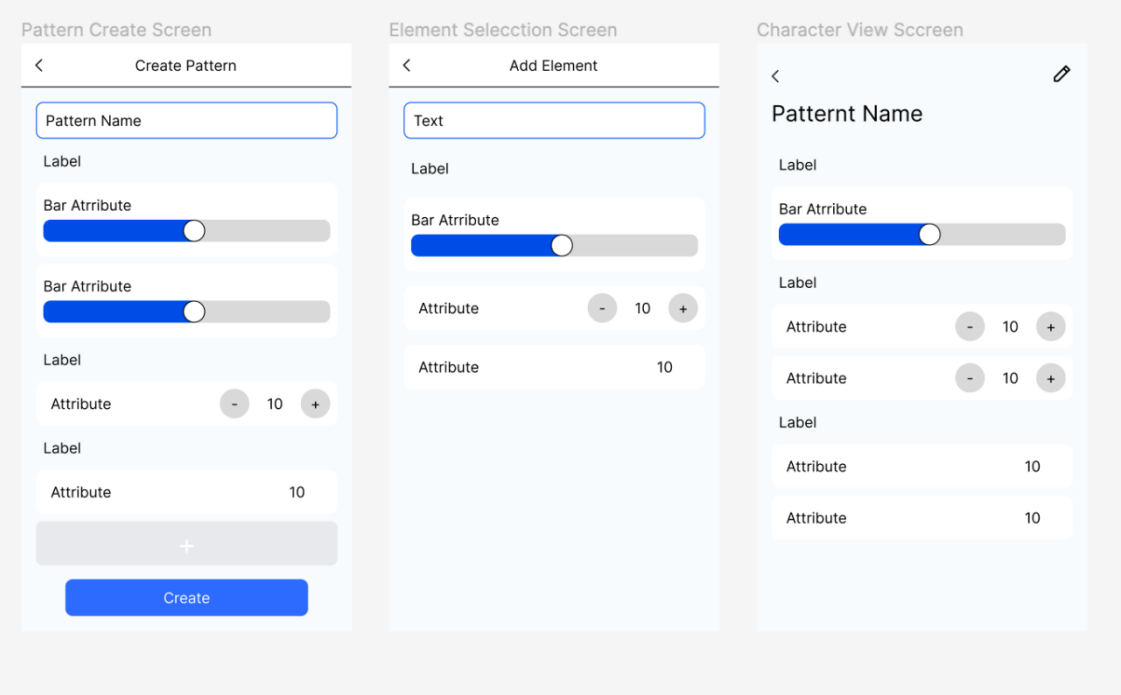

Рисунок 2.9 – Створення та редагування шаблону

Сторінка Account дозволятиме користувачу переглядати його акаунт та виконувати вихід з додатку. В майбутньому, цей екран слугуватиме для змін налаштування додатку або редагування інформації.

2.5 Проектування архітектури додатку

Концепція архітектури є однією з найактуальніших тем у сучасному світі програмування. Було розроблено багато рішень, які спрямовані на усунення того чи іншого недоліку. Архітектура додатка – одна з тем, до якої потрібно підходити з урахуванням особистих уподобань.

В даному проекті основною архітектурою виступає архітектура Stacked. Stacked, архітектура, яка пропонує чисті та ефективні рішення для архітектури програми створеної на Flutter. Від ін’єкції залежностей до готових служб та

багаторівневих структур, які дотримуються принципів чистої архітектури та багато іншого [9].

Базою для цієї архітектури є MVVM. Від неї Stacked отримав поняття моделі, представлення та класу, що з’єднує бізнес логіку з візуальним представленням або ViewModel.

У проекті, кожен екран або представлення має свою ViewModel. ViewModel бере на себе всю логіку яка присутня на екрані. Цей клас може відповідати за взаємодію з різними сервісами для відображення даних на екрані або здійснення навігації по додатку.

Для навігації Stacked має окремий вбудований сервіс, для якого потрібно задати екрани, що будуть використані у програмі, але після цих налаштувань, сервіс дозволяє здійснювати навігацію з будь якого рядку коду. Це усуває проблему, коли для навігації потрібні додаткові зміні які не доступні для ViewModel.

У проект було інтегровано ін'єкцію залежностей. Основною перевагою додання цього інструменту є прискорення розробки додатку. Ін'єкція залежностей або DI (Dependency Injection) виконує автоматичне створення моделей або ViewModel, передаючі всі необхідні параметри та зміні. Зникає потреба у ручному додані нового сервісу до класу, це особливо актуально якщо від цього класу залежать інші об’єкти.

Для зручності роботи з віддаленою базою використовуються сервіси. Кожен сервіс відповідальний за свою колекцію та за свій тип даних. Такі сервіси легко підтримувати, вони легко читаються а разом з DI не виникає складнощів у використанні їх на проекті.

2.6 Архітектура віддаленої бази даних

У базі Firestore було створено три основні колекції для збереження об’єктів. Кожна колекція відповідальна за свій тип даних. Колекція «users»

відповідальна за збереження інформації про користувачів. Вона має поля, що наведені у таблиці 2.1.

Таблиця 2.1 – Модель User

Поле	**Тип**	**Опис**
email	Текст	Зберігає електрону пошту
username	Текст	Зберігає ім’я користувача
id	Текст	Зберігає унікальний ідентифікатор користувача

Колекція «patterns» зберігає шаблони, які були створені юзерами програми. Структура документу що містить ця колекція наведена у табл.2.2.

Таблиця 2.2 – Модель Pattern

Поле	**Тип**	**Опис**
id	Текст	Унікальний ідентифікатор шаблону
name	Текст	Назва шаблону
description	Текст	Короткий опис шаблону
creatorId	Текст	Унікальний ідентифікатор юзера, що створив шаблон
elements	Масив	Масив, що зберігає інформацію про елементи з яких складається шаблон

Колекція «characters» включає у себе персонажів, що створені користувачами. Структура документу, що містить ця колекція наведена у таблиці 2.3.

Таблиця 2.3 – Модель Pattern

Поле	**Тип**	**Опис**
id	Текст	Унікальний ідентифікатор персонажа
name	Текст	Ім’я персонажа
patternId	Текст	Унікальний ідентифікатор шаблона до якого належить персонаж
creatorId	Текст	Унікальний ідентифікатор юзера, що створив персонажа
elementsData	Масив	Масив, що зберігає інформацію про дані які містяться в елементах шаблону

Для бази даних були задані окремі правила доступу до неї. Це необхідно для збереження цілісності даних та запобігання несанкціонованого доступу до інформації користувачів.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

Розробка мобільного додатку була розбита на наступні етапи

1. Створення проекту у програмному середовищі Android Studio
2. Створення інтерактивного прототипу мобільного додатку
3. Реалізація дизайну безпосередньо у проекті
4. Налаштування системи Firebase та створення локальної бази даних
5. Підключення мобільного додатку до програмного забезпечення Firestore, реалізація запитів.
6. Тестування мобільного додатку
7. Огляд перспективи розвитку додатку

3.1 Огляд функціоналу застосунка

При запуску додатку, першим відкривається Splash Screen. Це вікно необхідне для перевірки авторизації користувача та актуалізації необхідної інформації для запуску доданку.

Після перевірки, якщо користувач ще не авторизований у програмі відкривається вікно з опціями «Увійти» та «Зареєструватися» (рис.3.1).

При натисканні а кнопку «Увійти» відкривається вікно з тестовими полями у які користувач може ввести дані існуючого акаунту та здійснити вхід (рис. 3.2). Якщо користувач не має свого облікового запису, він може його створити натиснувши на кнопку реєстрація та заповнити необхідні поля (рис.3.3).

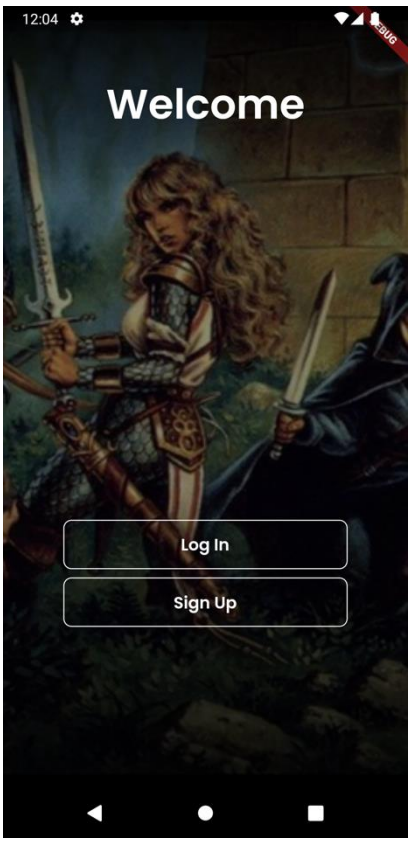

Рисунок 3.1 – Вікно «Welcome»

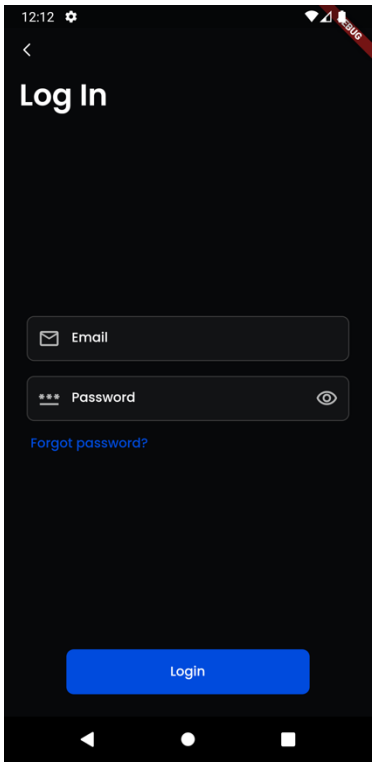

Рисунок 3.2 – Вікно авторизації

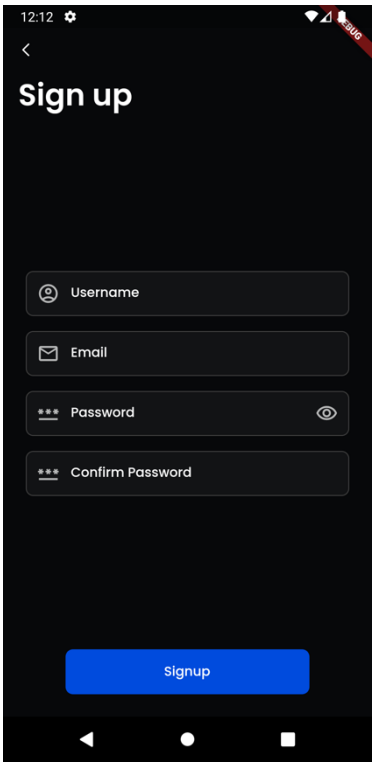

Рисунок 3.3 – Вікно реєстрації

Після здійснення авторизації або якщо юзер вже авторизований у застосунку відкривається головне вікно програми. Це вікно складається з навігаційного меню з чотирма різними вкладками (рис. 3.4).

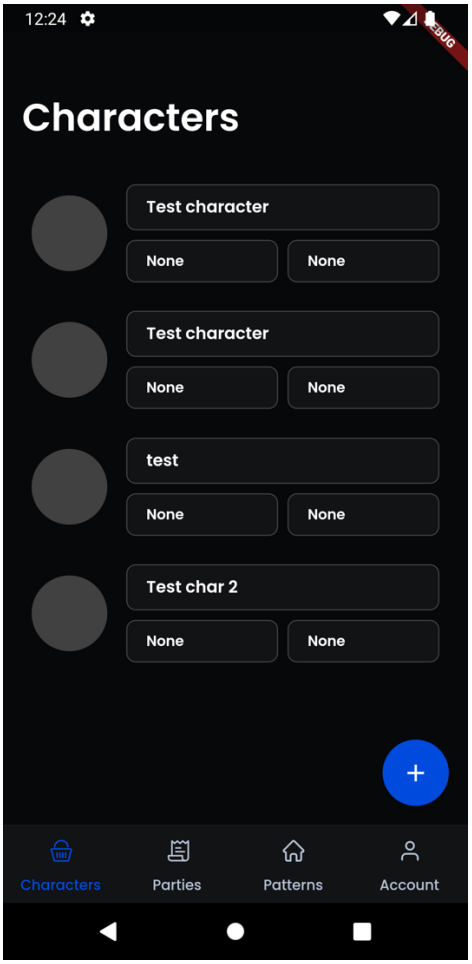

Рисунок 3.4 – Головне вікно програми

На вкладці «Characters» міститься список персонажів які були створені користувачем. На цьому ж екрані розташована кнопка для створення нового ігрового персонажа. При натисканні на неї, відкривається список з вибором доступних шаблонів, шаблони використовуються як заготовка для спрощеного внесення даних (рис.3.5). Після вибору шаблону, користувач вносить дані про свого ігрового персонажа, поле «Name» є обов'язковим для заповнення, форма даних інших полів залежить від конкретного шаблону (рис.3.6).

Рисунок 3.5 – Вікно вибору шаблона

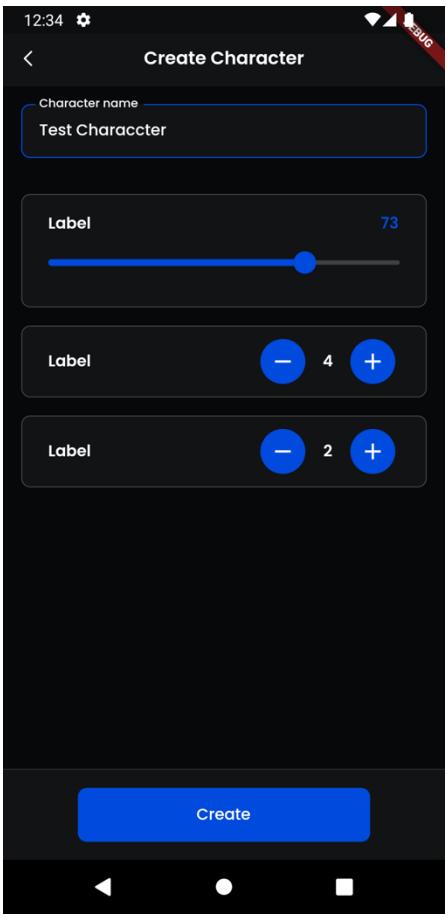

Рисунок 3.6 – Вікно створення персонажі

Вкладка «Pattern» відповідає за відображення доступних шаблонів. На цьому вікні користувач може вносити зміни до шаблонів, переглядати детальну інформацію та за потреби створити нові паттерни (рис. 3.7).

При натисканні на створення нового шаблону відкривається вікно з полями «Name», «Description» та кнопки додання нового елементу до шаблону (рис. 3.8), при натисканні на неї відкривається бібліотека наявних елементів які можна застосувати до шаблону (рис.3.9). Кожен елемент шаблону також має додаткові налаштуванні(рис.3.10). Ці налаштування можуть відрізнятися в залежності від типу елементу та обраних опцій.

Якщо користувач завершив створення або налаштування шаблону він повинен натиснути на кнопку збереження у кінці екрану, щоб зберегти або оновити дані.

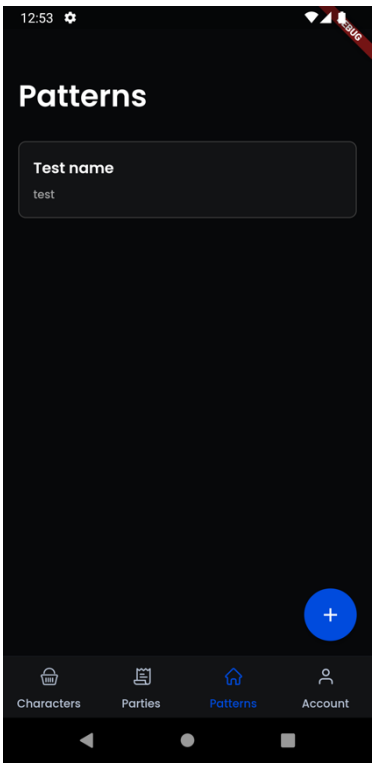

Рисунок 3.7 – Вкладка шаблонів

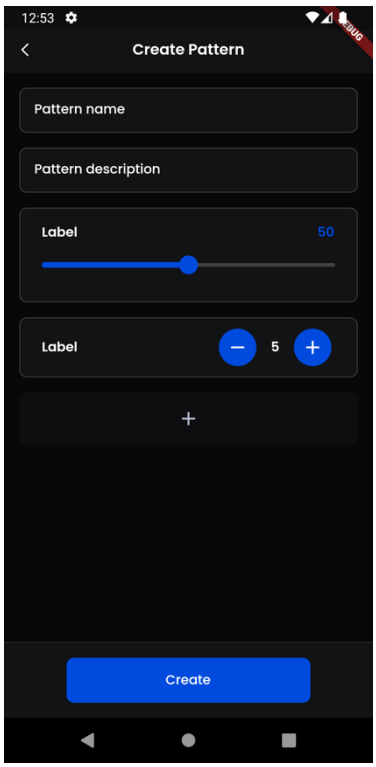

Рисунок 3.8 – Вікно створення шаблона

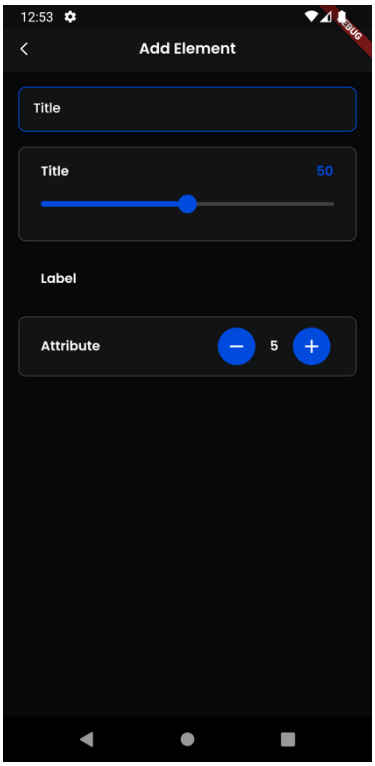

Рисунок 3.9 – Бібліотека елементів для шаблона

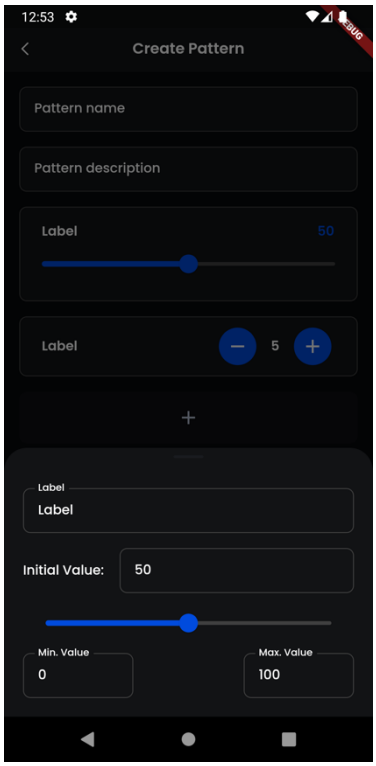

Рисунок 3.10 – Детальні налаштування елементу

Остання вкладка у застосунку – це «Account». Ця вкладка відображає користувачу інформацію про його обліковий запис. На цьому вікні містяться поля, що відображає поля з ім’ям юзера та його електронною адресою, також, тут знаходяться додаткові опції (рис.3.11). Вони дозволяють вийти з акаунту та перейти до списку друзів, але функціонал редагування друзів на разі не реалізований у програмі.

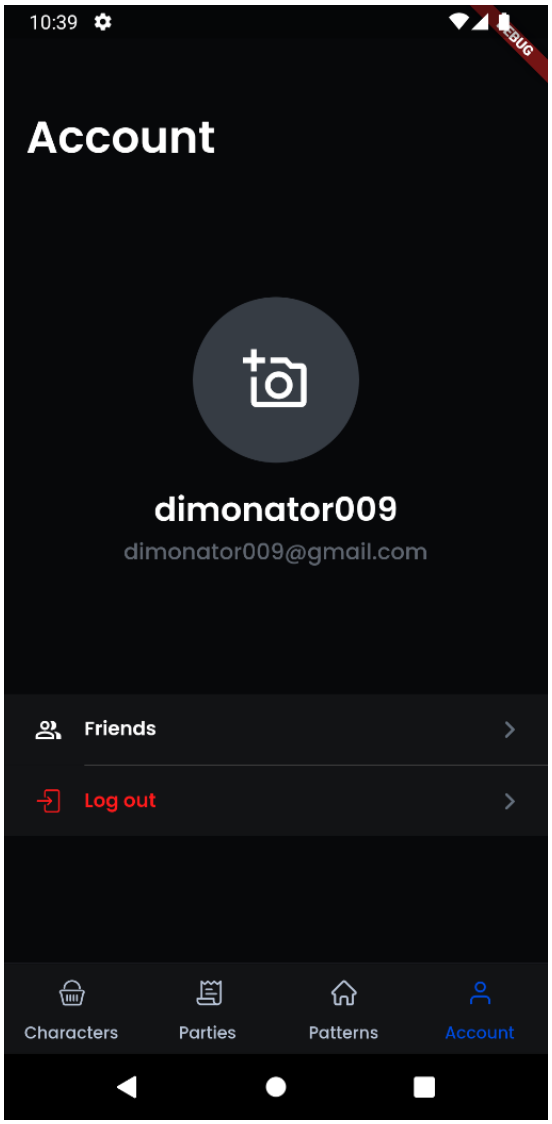

Рисунок 3.11 – Вікно вибору шаблона

3.2 Тестування додатку

Тестування мобільного додатка означає процес перевірки мобільного додатка на його функціональність і зручність використання перед його публічним випуском. Тестування мобільних програм допомагає перевірити, чи відповідає програма очікуваним технічним і бізнес-вимогам.

Якщо мобільні програми не перевірені ретельно, є висока ймовірність того, що користувачі зіткнуться з критичними помилками на своїх пристроях, які можуть призвести до поганої взаємодії з користувачем, особливо для нових користувачів. Перше враження є надзвичайно важливим для успіху будь-якої мобільної програми. Будь-який неочікуваний збій програми або функціональна помилка може призвести до видалення цієї програми. Це також призводить до втрати потенційних клієнтів [16].

Для успішного тестування мобільного застосунку потрібно тестувати його в різних роздільних здатностях екрана, версіях операційних систем і різних пропускних спроможностях мережі. Це допомагає гарантувати бездоганну роботу програми на багатьох конфігураціях пристроїв після публічного випуску релізної версії.

Даний проект було протестовано на оперційних системах iOS та Android різних версій. Для цього використано програмне забезпечення для емуляції даних операційних систем.

Емулятори дозволяють обрати необхідну версію операційної системи, змінити діагональ та роздільну здатність екрану та налаштувати технічні характеристики емульованого девайсу.

Також, тестування проводилось на реальних пристроях, так як емулятор не може забезпечити гарантію роботи на реальних девайсах. В якості пристроїв для тестування були використані iPhone X з версією програмного забезпечення iOS 15.5 та Google Pixel 2 XL для тестування на операційній системі Android 11.

3.3 Перспективи розвитку розробленої програми

Важливо підтримувати розроблений програмний продукт для утримання користувачів та розширення аудиторії. Це все можливо за допомогою запровадження нових функцій та виправлення існуючих помилок, що були знайдені користувачами застосунку.

Розроблена програма має перспективи для розвитку та пост релізної підтримки. Це також можливо завдяки обраному інструменту розробки Flutter. Нові функції одночасно реалізуються на обох операційних системах, що економить час та ресурси та дає можливість імплементації більшої кількості функцій за однаковий проміжок часу.

Перше найважливіше нововведення, яке потрібно додати у застосунок – це можливість редагувати інвентар кожного окремого персонажа. Дана функція дозволить користувачам редагувати предмети ігрового персонажа та легко додавати нові. Додатковою опцією потрібно реалізувати бібліотеку предметів. Вона дозволить зберігати предмети, що доступні для даного шаблону. Зникне потреба у ручному ведені інформації про ігровий предмет кожного разу.

Наступна, але не менш важлива функція – додання груп до застосунку. Користувачі зможуть створювати групи зі своїх ігрових персонажів що належать до одного шаблону.

Ця функціє є гарним доповненням до проекту так як дозволить юзерам переглядати персонажів інших гравців з якими вони грають. Це в свою чергу полегшить комунікацію між гравцями, адже зникає необхідність питати іншого гравця групи про його персонажа, які він має характеристики та предмети. Лідери групи матимуть легкий доступ до інформації інших гравців та зможуть приймати більш точні та ефективні рішення.

Реалізація системи друзів дасть змогу користувачам додавати своїх знайомих до окремого списку, щоб не витрачати кожного разу час на пошук

один одного. Ця функція покращить взаємодію гравців з додатком та один з одним.

Важливим фактором є зручність користування програмний забезпеченням. Наразі, доступна лише одна мова – це англійська, що може викликати складнощі у людей, що не знають цієї мови. Для покращення роботи з застосунком потрібно локалізувати додаток на українську мову.

Паралельно з цими нововведеннями є сенс збирати відгуки про програму від користувачів. Адже вони є цільовою аудиторією й добре розуміють, які недоліки має додаток та які додаткові функції необхідні для зручного користування програмний забезпечення.

ВИСНОВКИ

У ході виконання роботи було зроблено огляд можливих способів розробки програмного забезпечення для мобільних пристроїв. Розглянуто основні переваги та недоліки кожного з варіантів.

Зроблено огляд клієнт-серверної архітектури. Розглянуто різні підходи та дизайни у реалізації цієї системи. Наведено переваги кожного з дизайнів архітектури.

Проаналізовано наявні на ринку додатки, що відповідають тематиці асистента для настільних рольових ігор, а саме «Character Sheet» та «D&D Beyond». Аналіз було виконано за критеріями дизайну, кастомізації та функції що наявні в кожній програмі. На основі зробленого огляду, створено список вимог до майбутнього проекту та виділено пункти необхідні для його реалізації.

Було обрано основні інструменти та методи для вирішення задачі. Розроблено прототип інтерфейсу користувача, виконана практична реалізація застосунку та зроблено тестування.

Також, зважаючи на відсутність великою кількості аналогів на ринку, слід виділити актуальність розробки даного програмного забезпечення з використанням останніх технологій.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Ahmad Sahar and Craig Clayton. iOS 16 Programming for Beginners: Kickstart your iOS app development journey with a hands-on guide to Swift 5.7 and Xcode 14, 7th Edition. — Packt Publishing, 2022. — 686 p. — ISBN 978-1803237046.
2. John Horton. Android Programming for Beginners: Build in-depth, full-featured Android apps starting from zero programming experience, 3rd Edition — Packt Publishing, 2021. — 742 p. — ISBN 978-1800563438.
3. Alexander Benedikt Kuttig. Professional React Native: Expert techniques and solutions for building high-quality, cross-platform, production-ready apps— Packt Publishing, 2022. — 268 p. — ISBN 978-1800563681.
4. Chris Love. Progressive Web Application Development by Example: Develop fast, reliable, and engaging user experiences for the web — Packt Publishing, 2018. — 354 p. — ASIN B0788ZN8Z4.
5. Craig Berg. Cisco Networking Essentials: Complete Guide To Computer Networking For Beginners And Intermediates — Independently, 2020. — 85 p. — ASIN B08B7K5BRM.
6. Arshdeep Bahga. Cloud Computing Solutions Architect: A Hands-On Approach: A Competency-based Textbook for Universities and a Guide for AWS Cloud Certification and Beyond — VPT, 2019. — 826 p. — ISBN 978-0996025591.
7. Benjamin Bähr. Prototyping of User Interfaces for Mobile Applications. — 2017.— ISBN 978-3-319-53210-3.
8. Fabio Staiano. Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop — Packt Publishing, 2022. — 382 p. — ISBN 978-1800564183.
9. Jon Yablonski. Laws of UX: Using Psychology to Design Better Products & Services — O'Reilly, 2020. — 152 p. — ISBN 978-1492055310.

10. Eric Windmill. Flutter in Action. — Manning, 2019. — 368 p. — ISBN 978-1617296147.
11. Walrath, K. and Ladd, S. Dart: Up and Running. — O'Reilly, 2012. — 152 p. — ISBN 9781449330897.
12. Michael Katz, Kevin D Moore, Vincent Ngo and Vincenzo Guzzi. Flutter Apprentice (Third Edition): Learn to Build Cross-Platform Apps. — Razeware, 2022. — 683 p. — ISBN 978-1950325740.
13. Flutter.dev: How big is the Flutter engine? [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/docs/development/packages-and-plugins/developing-packages.
14. Laurence Moroney. The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform. — Apress, 2017. — 288 p. — ISBN 978-1484229422.
15. Quentin Ng. Flutter Practice Patterns: State Management. - JYK Designs, 2022. - 338 p. – ASIN B0B82Q43R6.
16. Daniel Knott. Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business. — Addison-Wesley Professional, 2015. — 256p. – ISBN 978-0134191713.

ДОДАТОК А

Додаток А1. Лістинг коду головного файлу main.dart

```
import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/constants.dart';
import 'package:role_assist/firebase_options.dart';
import 'package:role_assist/ui/views/app_view/app_view.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await EasyLocalization.ensureInitialized();
  await configureDependencies(Environment.prod);
  if (kDebugMode) {
    await
FirebaseCrashlytics.instance.setCrashlyticsCollectionEnabled(false);
  }
  FlutterError.onError = (FlutterErrorDetails details) {
    FlutterError.dumpErrorToConsole(details);
    FirebaseCrashlytics.instance.recordFlutterError(details);
  };

  runApp(EasyLocalization(
      supportedLocales: Constants.supportedLocales,
      path: 'assets/translations',
      fallbackLocale: Constants.fallbackLocale,
      startLocale: Constants.startLocale,
      child: const AppView()));
}
```

Додаток А2. Лістинг коду допоміжних файлів

```
import 'package:get_it/get_it.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/locator.config.dart';

// flutter packages pub run build_runner build
final locator = GetIt.instance;

@InjectableInit(
  initializerName: r'$initGetIt', // default
```

```
  preferRelativeImports: true, // default
  asExtension: false, // default
)
Future<GetIt> configureDependencies(String environment) async {
  //External
  return $initGetIt(locator, environment: environment);
}
import 'package:get_it/get_it.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/locator.config.dart';

// flutter packages pub run build_runner build
final locator = GetIt.instance;

@InjectableInit(
  initializerName: r'$initGetIt', // default
  preferRelativeImports: true, // default
  asExtension: false, // default
)
Future<GetIt> configureDependencies(String environment) async {
  //External
  return $initGetIt(locator, environment: environment);
}
```

Додаток А3. Лістинг коду моделей

```
import 'package:json_annotation/json_annotation.dart';
import 'package:role_assist/core/models/pattern_element.dart';

part 'character_model.g.dart';

List<Map> writeElementsDataList(List<ElementData> list) {
  return list.map((e) => e.toJson()).toList();
}

@JsonSerializable()
class CharacterModel {
  final String id;
  final String name;
  final String patternId;
  final String ownerId;

  @JsonKey(toJson: writeElementsDataList)
  final List<ElementData> elementsData;

  CharacterModel(this.id,
      {required this.patternId,
      required this.ownerId,
      this.name = '',
      this.elementsData = const []});

  factory CharacterModel.fromJson(Map<String, dynamic> json) =>
      _$CharacterModelFromJson(json);
```

```
  Map<String, dynamic> toJson() => _$CharacterModelToJson(this);
}

@JsonSerializable()
class ElementData {
  final String id;
  @JsonKey(unknownEnumValue: PatternElementType.label)
  final PatternElementType type;
  dynamic value;
  int compensateValue;

  ElementData(this.id,
      {this.value,
      this.compensateValue = 0,
      this.type = PatternElementType.label});

  factory ElementData.fromJson(Map<String, dynamic> json) =>
      _$ElementDataFromJson(json);

  Map<String, dynamic> toJson() => _$ElementDataToJson(this);
}

class ElementModel {
  final ElementData data;
  final PatternElement element;

  ElementModel(this.data, this.element);
}


import 'package:json_annotation/json_annotation.dart';

part 'pattern_element.g.dart';

enum PatternElementType { text, label, bar, attribute }

abstract class PatternElement {
  final String id;
  final PatternElementType type;
  String label;

  PatternElement({required this.id, required this.type, this.label = ''});

  Map<String, dynamic> toJson();
}

@JsonSerializable()
class TextElement extends PatternElement {
  String? initialValue;
  bool isNumber;

  TextElement(String id,
      {this.initialValue,
      this.isNumber = false,
```

```
      String label = '',
      PatternElementType type = PatternElementType.text})
      : super(id: id, type: type, label: label);

  factory TextElement.fromJson(Map<String, dynamic> json) =>
      _$TextElementFromJson(json);

  @override
  Map<String, dynamic> toJson() => _$TextElementToJson(this);
}

@JsonSerializable()
class LabelElement extends PatternElement {
  LabelElement(String id,
      {String label = '', PatternElementType type =
PatternElementType.label})
      : super(id: id, type: type, label: label);

  factory LabelElement.fromJson(Map<String, dynamic> json) =>
      _$LabelElementFromJson(json);

  @override
  Map<String, dynamic> toJson() => _$LabelElementToJson(this);
}

@JsonSerializable()
class BarElement extends PatternElement {
  int minValue;
  int maxValue;
  int initialValue;
  String? colorHex;

  BarElement(String id,
      {this.minValue = 0,
      this.maxValue = 100,
      this.colorHex,
      this.initialValue = 50,
      String label = '',
      PatternElementType type = PatternElementType.bar})
      : super(id: id, type: type, label: label);

  factory BarElement.fromJson(Map<String, dynamic> json) =>
      _$BarElementFromJson(json);

  @override
  Map<String, dynamic> toJson() => _$BarElementToJson(this);
}

@JsonSerializable()
class AttributeElement extends PatternElement {
  bool isAuto;
  int maxValue;
  int minValue;
  int initialValue;
  String? formula;
```

```
  AttributeElement(String id,
      {this.isAuto = false,
      this.formula,
      this.minValue = 0,
      this.initialValue = 5,
      String label = '',
      PatternElementType type = PatternElementType.attribute,
      this.maxValue = 10})
      : super(id: id, type: type, label: label);

  factory AttributeElement.fromJson(Map<String, dynamic> json) =>
      _$AttributeElementFromJson(json);

  @override
  Map<String, dynamic> toJson() => _$AttributeElementToJson(this);
}


import 'package:json_annotation/json_annotation.dart';
import 'package:role_assist/core/models/pattern_element.dart';

part 'pattern_model.g.dart';

List<Map> writeElementsList(List<PatternElement> list) {
  return list.map((e) => e.toJson()).toList();
}

List<PatternElement> convertElementsList(value) {
  return (value as List<dynamic>?)?.map((e) {
        switch (e['type']) {
          case 'text':
            return TextElement.fromJson(e);
          case 'label':
            return LabelElement.fromJson(e);
          case 'bar':
            return BarElement.fromJson(e);
          case 'attribute':
            return AttributeElement.fromJson(e);
          default:
            return LabelElement.fromJson(e);
        }
      }).toList() ??
      const [];
}

@JsonSerializable()
class PatternModel {
  String id;
  String creatorId;
  String name;
  String description;

  @JsonKey(fromJson: convertElementsList, toJson: writeElementsList)
  List<PatternElement> elements;
```

```
  PatternModel(
      {required this.id,
      required this.creatorId,
      this.description = '',
      this.name = '',
      this.elements = const []});

  factory PatternModel.fromJson(Map<String, dynamic> json) =>
      _$PatternModelFromJson(json);

  Map<String, dynamic> toJson() => _$PatternModelToJson(this);
}


import 'package:json_annotation/json_annotation.dart';

part 'user_model.g.dart';

@JsonSerializable()
class UserModel {
  @JsonKey(includeIfNull: false)
  final String id;
  final String email;
  final String username;

  const UserModel({this.email = '', this.username = '', required this.id});

  factory UserModel.fromJson(Map<String, dynamic> json) =>
      _$UserModelFromJson(json);

  Map<String, dynamic> toJson() => _$UserModelToJson(this);
}
```

Додаток А4. Лістинг коду сервісів

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:injectable/injectable.dart';
import
'package:role_assist/core/services/bottom_sheet/bottom_sheet_service.dart';
import 'package:role_assist/ui/shared/app_colors.dart';

@Injectable(as: AppBottomSheetService)
class AppBottomSheetServiceImpl extends AppBottomSheetService {
  @override
  Future<T?> openBottomSheet<T>(Widget child,
      {bool isScrollControlled = false,
      bool enableDrag = true,
      BorderRadius? borderRadius,
      bool isDismissible = true}) {
    return Get.bottomSheet<T>(
      child,
```

```
      useRootNavigator: true,
      enableDrag: enableDrag,
      isDismissible: isDismissible,
      backgroundColor: AppColors.dark_black,
      shape: RoundedRectangleBorder(
          borderRadius: borderRadius ??
              const BorderRadius.only(
                  topLeft: Radius.circular(24), topRight:
Radius.circular(24))),
      isScrollControlled: isScrollControlled,
    );
  }

  @override
  bool isBottomSheetOpen() {
    return Get.isBottomSheetOpen ?? false;
  }
}


import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/core/models/character_model.dart';
import
'package:role_assist/core/services/character_service/character_service.dart';

@Injectable(as: CharacterService)
class CharacterServiceImpl extends CharacterService {
  final _characters = FirebaseFirestore.instance.collection('characters');

  @override
  Future<CharacterModel?> createCharacter(CharacterModel characterModel)
async {
    String docId = characterModel.id;
    await _characters.doc(docId).set(characterModel.toJson());
    return getCharacter(docId);
  }

  @override
  Future<CharacterModel?> getCharacter(String id) async {
    var snapshot = await _characters.doc(id).get();
    if (snapshot.exists) {
      var data = snapshot.data() ?? {};
      var characterModel = CharacterModel.fromJson(data);
      return characterModel;
    }
    return null;
  }

  @override
  Future<List<CharacterModel>> getCharacters(String ownerId) async {
    var document = await _characters.where('ownerId', isEqualTo:
ownerId).get();
    List<CharacterModel> characters = document.docs.map((doc) {
      return CharacterModel.fromJson(doc.data());
```

```
    }).toList();
    return characters;
  }

}
```

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/core/services/dialog/dialog_service.dart';

@Injectable(as: AppDialogService)
class AppDialogServiceImpl extends AppDialogService {
  @override
  Future<dynamic> openDialog(Widget child, {bool barrierDismissible = true})
{
    return Get.dialog(child, barrierDismissible: barrierDismissible);
  }

  @override
  bool isDialogOpen() {
    return Get.isDialogOpen ?? false;
  }
}
```

```
import 'package:firebase_analytics/firebase_analytics.dart';
import 'package:injectable/injectable.dart';

import 'firebase_analytics_service.dart';

@Injectable(as: FirebaseAnalyticsService)
class FirebaseAnalyticsServiceImpl extends FirebaseAnalyticsService {
  final FirebaseAnalytics _analytics = FirebaseAnalytics.instance;

  FirebaseAnalyticsServiceImpl();

  @override
  FirebaseAnalytics get analytics => _analytics;
}
```

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:injectable/injectable.dart';
import
'package:role_assist/core/services/firebase_auth_service/auth_exception_handl
er.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';

@Injectable(as: FirebaseAuthService)
class FirebaseAuthServiceImpl extends FirebaseAuthService {
  static const _error = ErrorAuthResult('Unexpected error');
```

```
FirebaseAuth get _auth => FirebaseAuth.instance;

@override
User? currentUser() {
  return _auth.currentUser;
}

@override
Future<AuthResult> signInWithEmail(String email, String password) async {
  try {
    await _auth.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
    return const SuccessAuthResult();
  } on FirebaseAuthException catch (e) {
    return ErrorAuthResult(
        AuthExceptionHandler.handleSignInException(e.code));
  } catch (e) {
    return _error;
  }
}

@override
Future<AuthResult> sendResetPasswordEmail(String email) async {
  try {
    await _auth.sendPasswordResetEmail(email: email);
    return const SuccessAuthResult();
  } on FirebaseAuthException catch (e) {
    return ErrorAuthResult(
        AuthExceptionHandler.handleSendResetLinkException(e.code));
  } catch (e) {
    return _error;
  }
}

@override
Future<void> signOut() async {
  try {
    await _auth.signOut();
  } catch (e) {
    rethrow;
  }
}

@override
Future<AuthResult> signUpWithEmail(String email, String password) async {
  try {
    await _auth.createUserWithEmailAndPassword(
      email: email,
      password: password,
    );
    return const SuccessAuthResult();
  } on FirebaseAuthException catch (e) {
    return ErrorAuthResult(
```

```
          AuthExceptionHandler.handleSignUpException(e.code));
    } catch (e) {
      return _error;
    }
  }
}
```

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/core/models/pattern_model.dart';
import
'package:role_assist/core/services/pattern_service/pattern_service.dart';

@Injectable(as: PatternService)
class PatternServiceImpl extends PatternService {
  final _patterns = FirebaseFirestore.instance.collection('patterns');

  @override
  Future<PatternModel?> createPattern(PatternModel patternModel) async {
    String docId = patternModel.id;
    await _patterns.doc(docId).set(patternModel.toJson());
    return getPattern(docId);
  }

  @override
  Future<PatternModel?> getPattern(String id) async {
    var snapshot = await _patterns.doc(id).get();
    if (snapshot.exists) {
      var data = snapshot.data() ?? {};
      var patternModel = PatternModel.fromJson(data);
      return patternModel;
    }
    return null;
  }

  @override
  Future<List<PatternModel>> getPatterns() async {
    var document = await _patterns.get();
    List<PatternModel> patterns = document.docs.map((doc) {
      return PatternModel.fromJson(doc.data());
    }).toList();
    return patterns;
  }
}
```

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';

@LazySingleton(as: AppSnackBarService)
class AppSnackBarServiceImpl extends AppSnackBarService {
```

```
  @override
  Future<dynamic>? showSnackBar(String message,
      {Widget? mainButton,
      Duration? duration,
      SnackbarStatusCallback? snackBarStatus}) {
    return GetSnackBar(
      message: message,
      duration: duration ?? const Duration(seconds: 2, milliseconds: 500),
      margin: const EdgeInsets.all(20),
      snackStyle: SnackStyle.FLOATING,
      mainButton: mainButton,
      snackbarStatus: snackBarStatus,
      animationDuration: const Duration(milliseconds: 500),
      borderRadius: 12,
    ).show().future;
  }
}
```

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/core/models/user_model.dart';
import 'package:role_assist/core/services/user_service/user_service.dart';

@Injectable(as: UserService)
class UserServiceImpl extends UserService {
  final _users = FirebaseFirestore.instance.collection('users');

  @override
  Future<UserModel?> createUser(UserModel userModel) async {
    var id = userModel.id;
    await _users.doc(id).set(userModel.toJson());
    return getUser(id);
  }

  @override
  Future<UserModel?> getUser(String id) async {
    var snapshot = await _users.doc(id).get();
    if (snapshot.exists) {
      var data = snapshot.data() ?? {};
      data['id'] = snapshot.id;
      var userModel = UserModel.fromJson(data);
      return userModel;
    }
    return null;
  }
}
```

Додаток А5. Лістинг коду стилів

```
import 'dart:math';

import 'package:flutter/material.dart';

class AppColors {
  static const Color main = Color(0xFF004BDD);

  static const Color background = Color(0xFFF8FAFC);
  static const Color dark_background = Color(0xFF07080A);

  static const Color grey = Color(0xFF65707D);
  static const Color grey_dark = Color(0xFFB7C3D6);

  static const Color black = Color(0xFF1C2026);
  static const Color dark_black = Color(0xFF121315);

  static const Color light_grey = Color(0xFFE8E9E9);
  static const Color dark_light_grey = Color(0xFF414141);

  static const Color error = Color(0xFFFF1B1C);
  static const Color green = Color(0xFF12D3A5);
  static const Color yellow = Color(0xFFFFB524);
  static const Color grey_white = Color(0xFFAEB4BF);

  static int _tintValue(int value, double factor) =>
      max(0, min((value + ((255 - value) * factor)).round(), 255));

  static Color _tintColor(Color color, double factor) => Color.fromRGBO(
      _tintValue(color.red, factor),
      _tintValue(color.green, factor),
      _tintValue(color.blue, factor),
      1);

  static int _shadeValue(int value, double factor) =>
      max(0, min(value - (value * factor).round(), 255));

  static Color _shadeColor(Color color, double factor) => Color.fromRGBO(
      _shadeValue(color.red, factor),
      _shadeValue(color.green, factor),
      _shadeValue(color.blue, factor),
      1);

  static MaterialColor generateMaterialColor(Color color) {
    return MaterialColor(color.value, {
      50: _tintColor(color, 0.9),
      100: _tintColor(color, 0.8),
      200: _tintColor(color, 0.6),
      300: _tintColor(color, 0.4),
      400: _tintColor(color, 0.2),
      500: color,
      600: _shadeColor(color, 0.1),
      700: _shadeColor(color, 0.2),
      800: _shadeColor(color, 0.3),
      900: _shadeColor(color, 0.4),
    });
```

```
  }
}


class AppImages {
  static const String _prefix = 'assets/images/';

  static const String loginBackground = '${_prefix}login_background.jpg';
  static const String googleLogo = '${_prefix}google_logo.svg';
}


import 'package:flutter/material.dart';
import 'package:role_assist/ui/shared/app_colors.dart';

class AppThemes {
  static ThemeData mainTheme = ThemeData(
    fontFamily: "Poppins",
    brightness: Brightness.dark,
    primaryColor: AppColors.main,
    colorScheme: const ColorScheme.dark(
        primary: AppColors.main, secondary: AppColors.main),
    buttonTheme: const ButtonThemeData(minWidth: 10),
    dividerColor: AppColors.grey,
    selectedRowColor: AppColors.black,
    bottomAppBarColor: Colors.white,
    scaffoldBackgroundColor: AppColors.dark_background,
    dialogBackgroundColor: AppColors.dark_background,
    primarySwatch: AppColors.generateMaterialColor(AppColors.main),
  );
}
```

Додаток А6. Лістинг вікна акаунта

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/ui/shared/app_colors.dart';
import 'package:role_assist/ui/shared/app_icons.dart';
import 'package:role_assist/ui/views/account_view/account_view_model.dart';
import 'package:role_assist/ui/widgets/account_item.dart';
import 'package:role_assist/ui/widgets/app_header_widget.dart';
import 'package:stacked/stacked.dart';

class AccountView extends ViewModelBuilderWidget<AccountViewModel> {
  const AccountView({super.key});

  @override
  AccountViewModel viewModelBuilder(BuildContext context) {
    return locator<AccountViewModel>();
  }

  @override
  Widget builder(
```

```
      BuildContext context, AccountViewModel viewModel, Widget? child) {
    return SafeArea(
      child: Column(
        children: [
          const AppHeaderWidget(
            title: 'Account',
          ),
          Expanded(
            child: AnimatedSwitcher(
              duration: const Duration(milliseconds: 250),
              child: _buildBody(viewModel),
            ),
          )
        ],
      ),
    );
  }

  Widget _buildBody(AccountViewModel viewModel) {
    if (viewModel.isBusy) {
      return const Center(
          key: ValueKey<String>('Loading'), child:
CircularProgressIndicator());
    }
    return _buildScreen(viewModel);
  }

  Widget _buildScreen(AccountViewModel viewModel) {
    var user = viewModel.user;
    return Column(
      key: const ValueKey<String>('Ready'),
      children: [
        const Spacer(),
        Container(
          decoration: BoxDecoration(
            shape: BoxShape.circle,
            color: AppColors.grey.withOpacity(0.5),
          ),
          width: 120,
          height: 120,
          child: const Icon(
            Icons.add_a_photo_outlined,
            size: 48,
          ),
        ),
        const SizedBox(
          height: 16,
        ),
        Text(
          user.username,
          style: const TextStyle(fontSize: 24, fontWeight: FontWeight.w700),
        ),
        Text(user.email,
            style: TextStyle(
                fontSize: 16,
```

```
                fontWeight: FontWeight.w500,
                color: AppColors.grey_dark.withOpacity(0.5))),
        const Spacer(),
        const AccountItem(title: 'Friends', icon: Icons.people_alt_outlined),
        AccountItem(
            title: 'Log out',
            isBorder: false,
            icon: AppIcons.logout,
            onPressed: viewModel.onSignOut,
            color: AppColors.error),
        const Spacer(),
      ],
    );
  }
}


import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/app.router.dart';
import 'package:role_assist/core/models/user_model.dart';
import 'package:role_assist/core/services/dialog/dialog_service.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';
import 'package:role_assist/core/services/user_service/user_service.dart';
import 'package:role_assist/ui/shared/app_colors.dart';
import 'package:role_assist/ui/widgets/dialogs/app_alert_dialog.dart';
import 'package:role_assist/ui/widgets/dialogs/confirmation_dialog.dart';
import 'package:role_assist/ui/widgets/dialogs/loading_dialog.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';

@injectable
class AccountViewModel extends BaseViewModel {
  final FirebaseAuthService _firebaseAuthService;
  final AppSnackBarService _snackBarService;
  final UserService _userService;
  final AppDialogService _dialogService;
  final NavigationService _navigationService;

  late UserModel user;

  AccountViewModel(this._firebaseAuthService, this._userService,
      this._snackBarService, this._dialogService, this._navigationService) {
    _init();
  }

  Future<void> _init() async {
    setBusy(true);
    try {
      var currentUser = _firebaseAuthService.currentUser();
      if (currentUser == null) {
```

```
        throw Exception();
      }
      var user = await _userService.getUser(currentUser.uid);
      if (user == null) {
        throw Exception();
      }
      this.user = user;
    } catch (e, stacktrace) {
      FirebaseCrashlytics.instance.recordError(e, stacktrace);
      _snackBarService.showSnackBar(tr("an_error_has_occurred"));
    } finally {
      setBusy(false);
    }
  }

  void onSignOut() {
    _dialogService.openDialog(ConfirmationDialog(
      title: tr("are_you_sure"),
      message: tr('log_out_warning'),
      positive: tr("log_out"),
      positiveColor: AppColors.error,
      onNegative: _navigationService.back,
      onPositive: _signOut,
    ));
  }

  void _showLoadingDialog() {
    _dialogService.openDialog(const LoadingDialog(), barrierDismissible:
false);
  }

  Future<void> _signOut() async {
    _navigationService.back();
    _showLoadingDialog();
    setBusy(true);
    try {
      await _firebaseAuthService.signOut();
      _navigationService.pushNamedAndRemoveUntil(Routes.splashView,
          predicate: (route) => false);
    } catch (e) {
      FirebaseCrashlytics.instance.recordError(e, StackTrace.current);
      _dialogService.openDialog(AppAlertDialog(
          title: tr("an_error_has_occurred"),
          onPositive: _navigationService.back));
    } finally {
      _navigationService.back();
      setBusy(false);
    }
  }
}
```

Додаток А7. Лістинг вікна створення персонажа

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/core/models/character_model.dart';
import 'package:role_assist/core/models/pattern_element.dart';
import 'package:role_assist/core/models/pattern_model.dart';
import 'package:role_assist/ui/shared/app_colors.dart';
import
'package:role_assist/ui/views/character_edit_view/character_edit_view_model.d
art';
import 'package:role_assist/ui/widgets/app_text_field.dart';
import 'package:role_assist/ui/widgets/buttons/app_button.dart';
import 'package:role_assist/ui/widgets/custom_app_bar.dart';
import
'package:role_assist/ui/widgets/elements/attribute_element_widget.dart';
import 'package:role_assist/ui/widgets/elements/bar_element_widget.dart';
import 'package:role_assist/ui/widgets/elements/label_element_widget.dart';
import 'package:role_assist/ui/widgets/elements/text_element_widget.dart';
import 'package:stacked/stacked.dart';

class CharacterEditView extends
ViewModelBuilderWidget<CharacterEditViewModel> {
  final PatternModel patternModel;

  const CharacterEditView({super.key, required this.patternModel});

  @override
  CharacterEditViewModel viewModelBuilder(BuildContext context) {
    return locator<CharacterEditViewModel>(param1: patternModel);
  }

  @override
  Widget builder(
      BuildContext context, CharacterEditViewModel viewModel, Widget? child)
{
    return Scaffold(
      body: Column(
        children: [
          CustomAppBar(
              title: 'Create Character',
              onBackPressed: viewModel.onBackPressed),
          const SizedBox(
            height: 16,
          ),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 16),
            child: AppTextField(
              controller: viewModel.nameController,
              error: viewModel.nameError,
              onChanged: viewModel.onNameChanged,
              label: 'Character name',
            ),
          ),
          const SizedBox(
            height: 16,
          ),
```

```
          Expanded(
            child: ListView.separated(
              itemCount: viewModel.elements.length,
              padding: const EdgeInsets.symmetric(horizontal: 16, vertical:
16),
              itemBuilder: (context, index) {
                return _buildPatternElement(
                    viewModel, viewModel.elements[index]);
              },
              separatorBuilder: (BuildContext context, int index) {
                return const SizedBox(
                  height: 16,
                );
              },
            ),
          ),
          Container(
            width: double.infinity,
            padding: EdgeInsets.only(
                left: 16,
                right: 16,
                top: 16,
                bottom: MediaQuery.of(context).padding.bottom + 16),
            alignment: Alignment.center,
            decoration: BoxDecoration(
                color: AppColors.dark_black,
                border: Border(
                    top: BorderSide(
                        color: AppColors.dark_light_grey.withOpacity(0.5)))),
            child: AppButton(
                title: 'Create',
                onPressed: viewModel.onCreateCharacters,
                loading: viewModel.isBusy),
          ),
        ],
      ),
    );
  }

  Widget _buildPatternElement(
      CharacterEditViewModel viewModel, ElementModel elementModel) {
    if (elementModel.element is TextElement) {
      return TextElementWidget(
        label: elementModel.element.label,
        initialValue: elementModel.data.value,
        onChanged: (value) =>
            viewModel.onTextElementChanged(elementModel, value),
      );
    } else if (elementModel.element is LabelElement) {
      return LabelElementWidget(
        label: elementModel.element.label,
      );
    } else if (elementModel.element is BarElement) {
      var element = elementModel.element as BarElement;
      return BarElementWidget(
```

```
        label: element.label,
        minValue: element.minValue,
        maxValue: element.maxValue,
        currentValue: elementModel.data.value,
        onChanged: (value) => viewModel.onElementChanged(elementModel,
value),
      );
    } else if (elementModel.element is AttributeElement) {
      var element = elementModel.element as AttributeElement;
      return AttributeElementWidget(
        label: element.label,
        currentValue: elementModel.data.value,
        maxValue: element.maxValue,
        minValue: element.minValue,
        isAuto: element.isAuto,
        onChange: (value) => viewModel.onElementChanged(elementModel, value),
      );
    }
    return const SizedBox();
  }
}


import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/material.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/core/models/character_model.dart';
import 'package:role_assist/core/models/pattern_element.dart';
import 'package:role_assist/core/models/pattern_model.dart';
import
'package:role_assist/core/services/character_service/character_service.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';
import 'package:uuid/uuid.dart';

@injectable
class CharacterEditViewModel extends BaseViewModel {
  final NavigationService _navigationService;
  final AppSnackBarService _snackBarService;
  final CharacterService _characterService;
  final FirebaseAuthService _firebaseAuthService;
  final PatternModel patternModel;
  final Uuid _uuid = const Uuid();

  final TextEditingController nameController = TextEditingController();

  String? nameError;

  List<ElementModel> elements = [];
```

```
CharacterEditViewModel(
    this._navigationService,
    @factoryParam this.patternModel,
    this._snackBarService,
    this._characterService,
    this._firebaseAuthService) {
  _buildList();
}

void _buildList() {
  for (var patternElement in patternModel.elements) {
    Object? initialValue;
    if (patternElement is TextElement) {
      initialValue = patternElement.initialValue;
    } else if (patternElement is BarElement) {
      initialValue = patternElement.initialValue;
    } else if (patternElement is AttributeElement) {
      initialValue = patternElement.initialValue;
    }

    var data = ElementData(patternElement.id,
        type: patternElement.type, value: initialValue);
    var element = ElementModel(data, patternElement);
    elements.add(element);
  }
}

void onBackPressed() {
  _navigationService.back();
}

void onNameChanged(String value) {
  if (nameError != null && value.isNotEmpty) {
    nameError = null;
    notifyListeners();
  }
}

void onElementChanged(ElementModel elementModel, int value) {
  elementModel.data.value = value;
  notifyListeners();
}

void onTextElementChanged(ElementModel elementModel, String value) {
  elementModel.data.value = value;
  notifyListeners();
}

void onCreateCharacters() {
  String title = nameController.text;
  if (title.isEmpty) {
    nameError = 'Pattern name can not be empty';
    notifyListeners();
  } else {
    _createCharacter();
```

```
    }
  }

  Future<void> _createCharacter() async {
    setBusy(true);
    try {
      var currentUser = _firebaseAuthService.currentUser();
      if (currentUser == null) {
        throw Exception();
      }

      String id = _uuid.v4();
      String name = nameController.text;
      CharacterModel characterModel = CharacterModel(id,
          patternId: patternModel.id,
          ownerId: currentUser.uid,
          name: name,
          elementsData: elements.map((e) => e.data).toList());

      var newCharacterModel =
          await _characterService.createCharacter(characterModel);
      if (newCharacterModel != null) {
        _navigationService.back();
        _snackBarService.showSnackBar('Character Created');
      } else {
        throw Exception();
      }
    } catch (e, stacktrace) {
      FirebaseCrashlytics.instance.recordError(e, stacktrace);
      _snackBarService.showSnackBar(tr("an_error_has_occurred"));
    } finally {
      setBusy(false);
    }
  }
}
```

Додаток А8. Лістинг вікна зі списком персонажів

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import
'package:role_assist/ui/views/characters_view/character_item/character_item.d
art';
import
'package:role_assist/ui/views/characters_view/characters_view_model.dart';
import 'package:role_assist/ui/widgets/app_header_widget.dart';
import 'package:stacked/stacked.dart';

class CharactersView extends ViewModelBuilderWidget<CharactersViewModel> {
  const CharactersView({super.key});

  @override
  CharactersViewModel viewModelBuilder(BuildContext context) {
```

```
    return locator<CharactersViewModel>();
  }

  @override
  Widget builder(
      BuildContext context, CharactersViewModel viewModel, Widget? child) {
    return Scaffold(
      floatingActionButton: viewModel.isBusy
          ? null
          : FloatingActionButton(
              onPressed: viewModel.openEditCharacterView,
              elevation: 4,
              child: const Icon(
                Icons.add,
                size: 24,
                color: Colors.white,
              )),
      body: Column(
        children: [
          const AppHeaderWidget(
            title: 'Characters',
          ),
          const SizedBox(
            height: 16,
          ),
          Expanded(
            child: AnimatedSwitcher(
              duration: const Duration(milliseconds: 250),
              child: _buildBody(viewModel),
            ),
          )
        ],
      ),
    );
  }

  Widget _buildBody(CharactersViewModel viewModel) {
    if (viewModel.isBusy) {
      return const Center(
          key: ValueKey<String>('Loading'), child:
CircularProgressIndicator());
    }
    return _buildScreen(viewModel);
  }

  Widget _buildScreen(CharactersViewModel viewModel) {
    return ListView.separated(
      itemCount: viewModel.characters.length,
      padding: const EdgeInsets.only(left: 16, right: 16, top: 8, bottom:
48),
      separatorBuilder: (context, index) {
        return const SizedBox(
          height: 8,
        );
      },
```

```
      itemBuilder: (context, index) {
        var character = viewModel.characters[index];
        return CharacterItem(characterModel: character, onPressed:
viewModel.onCharacterPressed,);
      },
    );
  }
}


import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/app.router.dart';
import 'package:role_assist/core/models/character_model.dart';
import
'package:role_assist/core/services/character_service/character_service.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';

@injectable
class CharactersViewModel extends BaseViewModel {
  final NavigationService _navigationService;
  final AppSnackBarService _snackBarService;
  final CharacterService _characterService;
  final FirebaseAuthService _firebaseAuthService;

  List<CharacterModel> characters = [];

  CharactersViewModel(this._navigationService, this._snackBarService,
      this._characterService, this._firebaseAuthService) {
    _getCharacters();
  }

  Future<void> _getCharacters() async {
    setBusy(true);
    try {
      var currentUser = _firebaseAuthService.currentUser();
      if (currentUser == null) {
        throw Exception();
      }
      characters = await _characterService.getCharacters(currentUser.uid);
    } catch (e, stacktrace) {
      FirebaseCrashlytics.instance.recordError(e, stacktrace);
      _snackBarService.showSnackBar(tr("an_error_has_occurred"));
    } finally {
      setBusy(false);
    }
  }

  void openEditCharacterView() {
```

```
    _navigationService
        .navigateToPatternSelectionView()
        .then((value) => _getCharacters());
  }

  void onCharacterPressed(CharacterModel characterModel) {
  }
}
```

Додаток А8. Лістинг вікна створення шаблону

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/core/models/pattern_element.dart';
import 'package:role_assist/ui/shared/app_colors.dart';
import
'package:role_assist/ui/views/edit_pattern_view/edit_pattern_view_model.dart'
;
import 'package:role_assist/ui/widgets/app_text_field.dart';
import 'package:role_assist/ui/widgets/buttons/app_button.dart';
import 'package:role_assist/ui/widgets/custom_app_bar.dart';
import
'package:role_assist/ui/widgets/elements/attribute_element_widget.dart';
import 'package:role_assist/ui/widgets/elements/bar_element_widget.dart';
import 'package:role_assist/ui/widgets/elements/label_element_widget.dart';
import 'package:role_assist/ui/widgets/elements/text_element_widget.dart';
import 'package:stacked/stacked.dart';

class EditPatternView extends ViewModelBuilderWidget<EditPatternViewModel> {
  const EditPatternView({super.key});

  @override
  EditPatternViewModel viewModelBuilder(BuildContext context) {
    return locator<EditPatternViewModel>();
  }

  @override
  Widget builder(
      BuildContext context, EditPatternViewModel viewModel, Widget? child) {
    return Scaffold(
      body: Column(
        children: [
          CustomAppBar(
              title: 'Create Pattern', onBackPressed:
viewModel.onBackPressed),
          Expanded(
            child: ListView.separated(
              itemCount: viewModel.elements.length + 3,
              padding: const EdgeInsets.symmetric(horizontal: 16, vertical:
16),
              itemBuilder: (context, index) {
                if (index == 0) {
                  return AppTextField(
```

```
              controller: viewModel.titleController,
              error: viewModel.titleError,
              onChanged: viewModel.onTitleChanged,
              label: 'Pattern name',
            );
          } else if (index == 1) {
            return AppTextField(
              controller: viewModel.descriptionController,
              error: viewModel.descriptionError,
              onChanged: viewModel.onDescriptionChanged,
              label: 'Pattern description',
            );
          } else if (index == viewModel.elements.length + 2) {
            return TextButton(
                onPressed: viewModel.onAddNewElement,
                style: TextButton.styleFrom(
                    backgroundColor:
                        AppColors.dark_black.withOpacity(0.6),
                    shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(8)),
                    padding: const EdgeInsets.all(16)),
                child: const Icon(
                  Icons.add,
                  color: AppColors.grey_dark,
                ));
          }
          return _buildPatternElement(
              viewModel, viewModel.elements[index - 2]);
        },
        separatorBuilder: (BuildContext context, int index) {
          return const SizedBox(
            height: 16,
          );
        },
      ),
    ),
    Container(
      width: double.infinity,
      padding: EdgeInsets.only(
          left: 16,
          right: 16,
          top: 16,
          bottom: MediaQuery.of(context).padding.bottom + 16),
      alignment: Alignment.center,
      decoration: BoxDecoration(
          color: AppColors.dark_black,
          border: Border(
              top: BorderSide(
                  color: AppColors.dark_light_grey.withOpacity(0.5)))),
      child: AppButton(
          title: 'Create',
          onPressed: viewModel.onCreatePattern,
          loading: viewModel.isBusy),
    ),
  ],
```

```
      ),
    );
  }

  Widget _buildPatternElement(
      EditPatternViewModel viewModel, PatternElement patternElement) {
    if (patternElement is TextElement) {
      return TextElementWidget(
        label: patternElement.label,
        onPressed: () => viewModel.onTextDetails(patternElement),
      );
    } else if (patternElement is LabelElement) {
      return LabelElementWidget(
        label: patternElement.label,
        onPressed: () => viewModel.onLabelDetails(patternElement),
      );
    } else if (patternElement is BarElement) {
      return BarElementWidget(
        label: patternElement.label,
        minValue: patternElement.minValue,
        maxValue: patternElement.maxValue,
        currentValue: patternElement.initialValue,
        onPressed: () => viewModel.onBarDetails(patternElement),
      );
    } else if (patternElement is AttributeElement) {
      return AttributeElementWidget(
        label: patternElement.label,
        currentValue: patternElement.initialValue,
        maxValue: patternElement.maxValue,
        minValue: patternElement.minValue,
        isAuto: patternElement.isAuto,
        onPressed: () => viewModel.onAttributeDetails(patternElement),
      );
    }
    return const SizedBox();
  }
}


import 'package:easy_localization/easy_localization.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/material.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/app.router.dart';
import 'package:role_assist/core/models/pattern_element.dart';
import 'package:role_assist/core/models/pattern_model.dart';
import
'package:role_assist/core/services/bottom_sheet/bottom_sheet_service.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';
import
'package:role_assist/core/services/pattern_service/pattern_service.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';
import
```

```
'package:role_assist/ui/views/attribute_details_view/attribute_details_view.d
art';
import 'package:role_assist/ui/views/bar_details_view/bar_details_view.dart';
import
'package:role_assist/ui/views/label_details_view/label_details_view.dart';
import
'package:role_assist/ui/views/text_details_view/text_details_view.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';
import 'package:uuid/uuid.dart';

@injectable
class EditPatternViewModel extends BaseViewModel {
  final NavigationService _navigationService;
  final AppBottomSheetService _bottomSheetService;
  final FirebaseAuthService _firebaseAuthService;
  final PatternService _patternService;
  final AppSnackBarService _snackBarService;
  final Uuid _uuid = const Uuid();

  final TextEditingController titleController = TextEditingController();
  final TextEditingController descriptionController =
TextEditingController();

  final List<PatternElement> elements = [];

  String? titleError;
  String? descriptionError;

  EditPatternViewModel(this._navigationService, this._bottomSheetService,
      this._firebaseAuthService, this._patternService,
this._snackBarService);

  void onTitleChanged(String value) {
    if (titleError != null && value.isNotEmpty) {
      titleError = null;
      notifyListeners();
    }
  }

  void onDescriptionChanged(String value) {
    if (descriptionError != null && value.isNotEmpty) {
      descriptionError = null;
      notifyListeners();
    }
  }

  void onAddNewElement() {
    _navigationService.navigateToElementsSelectionView().then((value) {
      if (value is PatternElement) {
        elements.add(value);
      }
      notifyListeners();
    });
  }
```

```
void onBarDetails(BarElement barElement) {
  _bottomSheetService
      .openBottomSheet(
          BarDetailsView(
            barElement: barElement,
          ),
          isScrollControlled: true)
      .then((value) {
    notifyListeners();
  });
}

void onTextDetails(TextElement textElement) {
  _bottomSheetService
      .openBottomSheet(
          TextDetailsView(
            textElement: textElement,
          ),
          isScrollControlled: true)
      .then((value) {
    notifyListeners();
  });
}

void onLabelDetails(LabelElement labelElement) {
  _bottomSheetService
      .openBottomSheet(
          LabelDetailsView(
            labelElement: labelElement,
          ),
          isScrollControlled: true)
      .then((value) {
    notifyListeners();
  });
}

void onAttributeDetails(AttributeElement attributeElement) {
  _bottomSheetService
      .openBottomSheet(
          AttributeDetailsView(
            attributeElement: attributeElement,
          ),
          isScrollControlled: true)
      .then((value) {
    notifyListeners();
  });
}

void onCreatePattern() {
  String title = titleController.text;
  if (title.isEmpty) {
    titleError = 'Pattern name can not be empty';
    notifyListeners();
  } else {
```

```
      _createPattern();
    }
  }

  Future<void> _createPattern() async {
    setBusy(true);
    try {
      var id = _uuid.v4();
      var user = _firebaseAuthService.currentUser();
      if (user == null) {
        throw Exception();
      }
      var patternName = titleController.text;
      var descriptionName = descriptionController.text;

      PatternModel patternModel = PatternModel(
          id: id,
          creatorId: user.uid,
          name: patternName,
          description: descriptionName,
          elements: elements);

      var createdModel = await _patternService.createPattern(patternModel);
      if (createdModel != null) {
        _navigationService.back();
        _snackBarService.showSnackBar('Pattern Created');
      } else {
        throw Exception();
      }
    } catch (e, stacktrace) {
      FirebaseCrashlytics.instance.recordError(e, stacktrace);
      _snackBarService.showSnackBar(tr("an_error_has_occurred"));
    } finally {
      setBusy(false);
    }
  }

  void onBackPressed() {
    _navigationService.back();
  }
}
```

Додаток А9. Лістинг коду головного вікна

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/ui/views/home_view/home_view_model.dart';
import 'package:role_assist/ui/widgets/bottom_navigation.dart';
import 'package:stacked/stacked.dart';

class HomeView extends ViewModelBuilderWidget<HomeViewModel> {
```

```
  const HomeView({super.key});

  @override
  HomeViewModel viewModelBuilder(BuildContext context) {
    return locator<HomeViewModel>();
  }

  @override
  Widget builder(BuildContext context, viewModel, Widget? child) {
    return Scaffold(
      body: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Expanded(child: viewModel.screen()),
          BottomNavigation(
            onSelectTab: viewModel.onSelectTab,
            items: viewModel.buildItems(),
            currentTab: viewModel.currentIndex,
          )
        ],
      ),
    );
  }
}
```

```
import 'package:flutter/material.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/ui/shared/app_icons.dart';
import 'package:role_assist/ui/views/account_view/account_view.dart';
import 'package:role_assist/ui/views/characters_view/characters_view.dart';
import 'package:role_assist/ui/views/parties_view/parties_view.dart';
import 'package:role_assist/ui/views/patterns_view/patterns_view.dart';
import 'package:role_assist/ui/widgets/bottom_navigation.dart';
import 'package:stacked/stacked.dart';

@injectable
class HomeViewModel extends BaseViewModel {
  int _currentIndex = 0;

  int get currentIndex => _currentIndex;

  void onSelectTab(int index) {
    _currentIndex = index;
    notifyListeners();
  }


  Widget screen() {
    switch (_currentIndex) {
      case 0:
        return const CharactersView();
      case 1:
        return const PartiesView();
      case 2:
```

```
        return const PatternsView();
      case 3:
        return const AccountView();
      default:
        return Container();
    }
  }

  List<TabItem> buildItems() {
    return [
      const TabItem(
        index: 0,
        bottomNavigationBarItem: BottomNavigationBarItem(
            label: 'Characters',
            icon: Padding(
              padding: EdgeInsets.only(bottom: 10, top: 5),
              child: Icon(
                AppIcons.shopping,
              ),
            )),
      ),
      const TabItem(
        index: 1,
        bottomNavigationBarItem: BottomNavigationBarItem(
          label: 'Parties',
          icon: Padding(
            padding: EdgeInsets.only(bottom: 10, top: 5),
            child: Icon(AppIcons.invoice),
          ),
        ),
      ),
      const TabItem(
        index: 2,
        bottomNavigationBarItem: BottomNavigationBarItem(
            label: 'Patterns',
            icon: Padding(
              padding: EdgeInsets.only(bottom: 10, top: 5),
              child: Icon(AppIcons.stores),
            )),
      ),
      const TabItem(
        index: 3,
        bottomNavigationBarItem: BottomNavigationBarItem(
            label: 'Account',
            icon: Padding(
              padding: EdgeInsets.only(bottom: 10, top: 5),
              child: Icon(AppIcons.account),
            )),
      )
    ];
  }
}
```

Додаток А10. Лістинг коду вікна входу в обліковий запис

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/ui/views/login_view/login_view_model.dart';
import 'package:role_assist/ui/widgets/app_header_widget.dart';
import 'package:role_assist/ui/widgets/app_text_field.dart';
import 'package:role_assist/ui/widgets/buttons/app_button.dart';
import 'package:stacked/stacked.dart';

class LoginView extends ViewModelBuilderWidget<LoginViewModel> {
  const LoginView({super.key});

  @override
  LoginViewModel viewModelBuilder(BuildContext context) {
    return locator<LoginViewModel>();
  }

  @override
  Widget builder(
      BuildContext context, LoginViewModel viewModel, Widget? child) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
            AppHeaderWidget(title: 'Log In', onBack:
viewModel.onBackPressed),
            const Spacer(),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 24, vertical:
16),
              child: AppTextField(
                label: 'Email',
                error: viewModel.emailError,
                enable: !viewModel.isBusy,
                prefixIcon: Icons.email_outlined,
                onChanged: viewModel.onEmailChanged,
                controller: viewModel.emailController,
                keyboardType: TextInputType.emailAddress,
              ),
            ),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 24),
              child: AppTextField(
                  label: 'Password',
                  enable: !viewModel.isBusy,
                  prefixIcon: Icons.password_outlined,
                  error: viewModel.passwordError,
                  keyboardType: TextInputType.visiblePassword,
                  obscureText: viewModel.obscurePassword,
                  onChanged: viewModel.onPasswordChanged,
                  controller: viewModel.passwordController,
                  suffixIcon: GestureDetector(
                      onTap: viewModel.onObscurePasswordSwitch,
                      child: Icon(viewModel.obscurePassword
                          ? Icons.visibility_outlined
                          : Icons.visibility_off_outlined))),
```

```
            ),
            Align(
              alignment: Alignment.centerLeft,
              child: Padding(
                padding: const EdgeInsets.symmetric(horizontal: 24),
                child: TextButton(
                  onPressed: viewModel.onForgotPassword,
                  style: TextButton.styleFrom(
                      minimumSize: const Size(16, 16),
                      padding: const EdgeInsets.symmetric(
                          horizontal: 4, vertical: 8)),
                  child: const Text(
                    'Forgot password?',
                    style: TextStyle(fontSize: 14, fontWeight:
FontWeight.w500),
                  ),
                ),
              ),
            ),
            const Spacer(),
            AppButton(
                title: 'Login',
                onPressed: viewModel.isBusy ? null :
viewModel.onLoginPressed,
                loading: viewModel.isBusy),
            const SizedBox(
              height: 24,
            )
          ],
        ),
      ),
    );
  }
}


import 'package:easy_localization/easy_localization.dart';
import 'package:email_validator/email_validator.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/material.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/app.router.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';

@injectable
class LoginViewModel extends BaseViewModel {
  final NavigationService _navigationService;
  final FirebaseAuthService _firebaseAuthService;
  final AppSnackBarService _snackBarService;
```

```
final TextEditingController emailController = TextEditingController();
final TextEditingController passwordController = TextEditingController();

bool obscurePassword = true;

String? emailError;
String? passwordError;

LoginViewModel(this._navigationService, this._firebaseAuthService,
    this._snackBarService);

void onBackPressed() {
  _navigationService.back();
}

void onObscurePasswordSwitch() {
  obscurePassword = !obscurePassword;
  notifyListeners();
}

void onPasswordChanged(String value) {
  if (passwordError != null && value.isNotEmpty) {
    passwordError = null;
    notifyListeners();
  }
}

void onForgotPassword() {
  _navigationService.navigateTo(Routes.forgotPasswordView);
}

void onEmailChanged(String value) {
  bool isEmailValid = EmailValidator.validate(value);
  if (emailError != null && isEmailValid) {
    emailError = null;
    notifyListeners();
  }
}

void onLoginPressed() {
  emailError = null;
  passwordError = null;
  String email = emailController.text;
  String password = passwordController.text;
  bool isEmailValid = EmailValidator.validate(email);
  bool isValid = true;
  if (!isEmailValid) {
    emailError = 'Email is invalid';
    isValid = false;
  }
  if (password.isEmpty) {
    passwordError = 'Password can not be empty';
    isValid = false;
  }
  notifyListeners();
```

```
    if (isValid) {
      _login(email, password);
    }
  }

  Future<void> _login(String email, String password) async {
    setBusy(true);
    try {
      var singInResult =
          await _firebaseAuthService.signInWithEmail(email, password);
      if (singInResult is ErrorAuthResult) {
        _snackBarService.showSnackBar(singInResult.error);
        return;
      }
      openHomeView();
    } catch (e, stacktrace) {
      FirebaseCrashlytics.instance.recordError(e, stacktrace);
      _snackBarService.showSnackBar(tr("an_error_has_occurred"));
    } finally {
      setBusy(false);
    }
  }

  void openHomeView() {
    _navigationService.clearStackAndShow(Routes.homeView);
  }
}
```

Додаток А11. Лістинг коду вікна реєстрації

```
import 'package:flutter/material.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/ui/views/sign_up_view/sign_up_view_model.dart';
import 'package:role_assist/ui/widgets/app_header_widget.dart';
import 'package:role_assist/ui/widgets/app_text_field.dart';
import 'package:role_assist/ui/widgets/buttons/app_button.dart';
import 'package:stacked/stacked.dart';

class SignUpView extends ViewModelBuilderWidget<SignUpViewModel> {
  const SignUpView({super.key});

  @override
  SignUpViewModel viewModelBuilder(BuildContext context) {
    return locator<SignUpViewModel>();
  }

  @override
  Widget builder(
      BuildContext context, SignUpViewModel viewModel, Widget? child) {
    return Scaffold(
      body: SafeArea(
        child: Column(
          children: [
```

```
            AppHeaderWidget(title: 'Sign up', onBack:
viewModel.onBackPressed),
            const Spacer(),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 24, vertical:
8),
              child: AppTextField(
                label: 'Username',
                error: viewModel.usernameError,
                enable: !viewModel.isBusy,
                prefixIcon: Icons.account_circle_outlined,
                onChanged: viewModel.onUsernameChanged,
                controller: viewModel.usernameController,
                keyboardType: TextInputType.emailAddress,
              ),
            ),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 24, vertical:
8),
              child: AppTextField(
                label: 'Email',
                error: viewModel.emailError,
                enable: !viewModel.isBusy,
                prefixIcon: Icons.email_outlined,
                onChanged: viewModel.onEmailChanged,
                controller: viewModel.emailController,
                keyboardType: TextInputType.emailAddress,
              ),
            ),
            Padding(
                padding:
                    const EdgeInsets.symmetric(horizontal: 24, vertical: 8),
                child: AppTextField(
                    label: 'Password',
                    enable: !viewModel.isBusy,
                    prefixIcon: Icons.password_outlined,
                    error: viewModel.passwordError,
                    keyboardType: TextInputType.visiblePassword,
                    obscureText: viewModel.obscurePassword,
                    onChanged: viewModel.onPasswordChanged,
                    controller: viewModel.passwordController,
                    suffixIcon: GestureDetector(
                        onTap: viewModel.onObscurePasswordSwitch,
                        child: Icon(viewModel.obscurePassword
                            ? Icons.visibility_outlined
                            : Icons.visibility_off_outlined)))),
            Padding(
              padding: const EdgeInsets.symmetric(horizontal: 24, vertical:
8),
              child: AppTextField(
                  label: 'Confirm Password',
                  enable: !viewModel.isBusy,
                  obscureText: true,
                  prefixIcon: Icons.password_outlined,
                  error: viewModel.confirmPasswordError,
```

```
                  keyboardType: TextInputType.visiblePassword,
                  onChanged: viewModel.onConfirmPasswordChanged,
                  controller: viewModel.confirmPasswordController),
            ),
            const Spacer(),
            AppButton(
                title: 'Signup',
                onPressed: viewModel.isBusy ? null :
viewModel.onSignupPressed,
                loading: viewModel.isBusy),
            const SizedBox(
              height: 24,
            )
          ],
        ),
      ),
    );
  }
}


import 'package:easy_localization/easy_localization.dart';
import 'package:email_validator/email_validator.dart';
import 'package:firebase_crashlytics/firebase_crashlytics.dart';
import 'package:flutter/material.dart';
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/app.router.dart';
import 'package:role_assist/core/models/user_model.dart';
import
'package:role_assist/core/services/firebase_auth_service/firebase_auth_servic
e.dart';
import 'package:role_assist/core/services/snack_bar/snack_bar_service.dart';
import 'package:role_assist/core/services/user_service/user_service.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';

@injectable
class SignUpViewModel extends BaseViewModel {
  final NavigationService _navigationService;
  final FirebaseAuthService _firebaseAuthService;
  final AppSnackBarService _snackBarService;
  final UserService _userService;

  final TextEditingController usernameController = TextEditingController();
  final TextEditingController emailController = TextEditingController();
  final TextEditingController passwordController = TextEditingController();
  final TextEditingController confirmPasswordController =
      TextEditingController();

  bool obscurePassword = true;

  String? usernameError;
  String? emailError;
  String? passwordError;
  String? confirmPasswordError;
```

```
SignUpViewModel(this._navigationService, this._firebaseAuthService,
    this._snackBarService, this._userService);

void onBackPressed() {
  _navigationService.back();
}

void onObscurePasswordSwitch() {
  obscurePassword = !obscurePassword;
  notifyListeners();
}

void onUsernameChanged(String value) {
  if (usernameError != null && value.isNotEmpty) {
    usernameError = null;
    notifyListeners();
  }
}

void onPasswordChanged(String value) {
  if (passwordError != null && value.isNotEmpty) {
    passwordError = null;
    notifyListeners();
  }
}

void onConfirmPasswordChanged(String value) {
  if (confirmPasswordError != null && value.isNotEmpty) {
    confirmPasswordError = null;
    notifyListeners();
  }
}

void onEmailChanged(String value) {
  bool isEmailValid = EmailValidator.validate(value);
  if (emailError != null && isEmailValid) {
    emailError = null;
    notifyListeners();
  }
}

void onSignupPressed() {
  emailError = null;
  passwordError = null;
  confirmPasswordError = null;
  usernameError = null;
  String username = usernameController.text;
  String email = emailController.text;
  String password = passwordController.text;
  String confirmPassword = confirmPasswordController.text;
  bool isEmailValid = EmailValidator.validate(email);
  bool isValid = true;
  if (!isEmailValid) {
    emailError = 'Email is invalid';
```

```
      isValid = false;
    }
    if (username.isEmpty) {
      usernameError = 'Username can not be empty';
      isValid = false;
    }
    if (password.isEmpty) {
      passwordError = 'Password can not be empty';
      isValid = false;
    }
    if (confirmPassword != password) {
      confirmPasswordError = 'Passwords do not match';
      isValid = false;
    }
    notifyListeners();
    if (isValid) {
      _signup(email, password);
    }
  }

  Future<void> _signup(String email, String password) async {
    setBusy(true);
    try {
      var singUpResult =
          await _firebaseAuthService.signUpWithEmail(email, password);
      if (singUpResult is ErrorAuthResult) {
        _snackBarService.showSnackBar(singUpResult.error);
        return;
      }
      var singInResult =
          await _firebaseAuthService.signInWithEmail(email, password);
      if (singInResult is ErrorAuthResult) {
        _snackBarService.showSnackBar(singInResult.error);
        return;
      }
      var currentUser = _firebaseAuthService.currentUser();
      if (currentUser == null) {
        _snackBarService.showSnackBar(tr("an_error_has_occurred"));
        return;
      }
      String username = usernameController.text;
      var userModel = await _userService.createUser(UserModel(id:
currentUser.uid,email: email, username: username));
      if (userModel == null) {
        _snackBarService.showSnackBar(tr("an_error_has_occurred"));
        return;
      }
      openHomeView();
    } catch (e, stacktrace) {
      FirebaseCrashlytics.instance.recordError(e, stacktrace);
      _snackBarService.showSnackBar(tr("an_error_has_occurred"));
    } finally {
      setBusy(false);
    }
  }
```

```
  void openHomeView() {
    _navigationService.clearStackAndShow(Routes.homeView);
  }
}
```

Додаток А12. Лістинг коду вікна привітання

```
import 'package:flutter/material.dart';
import 'package:flutter_svg/flutter_svg.dart';
import 'package:role_assist/app/locator.dart';
import 'package:role_assist/ui/shared/app_images.dart';
import 'package:role_assist/ui/views/welcome_view/welcome_view_model.dart';
import 'package:role_assist/ui/widgets/buttons/outline_button.dart';
import 'package:role_assist/ui/widgets/or_line_widget.dart';
import 'package:stacked/stacked.dart';

class WelcomeView extends ViewModelBuilderWidget<WelcomeViewModel> {
  const WelcomeView({super.key});

  @override
  WelcomeViewModel viewModelBuilder(BuildContext context) {
    return locator<WelcomeViewModel>();
  }

  @override
  Widget builder(
      BuildContext context, WelcomeViewModel viewModel, Widget? child) {
    return Scaffold(
      body: Stack(
        fit: StackFit.expand,
        children: [
          Positioned.fill(
              child: Image.asset(
            AppImages.loginBackground,
            fit: BoxFit.cover,
          )),
          Container(
            color: Colors.black45,
            child: Column(
              children: [
                buildTitle(),
                const Expanded(flex: 3, child: SizedBox()),
                _buildButtons(viewModel),
              ],
            ),
          ),
        ],
      ),
    );
  }

  Expanded buildTitle() {
```

```
    return Expanded(
      flex: 2,
      child: Container(
        alignment: Alignment.center,
        decoration: const BoxDecoration(
            gradient: LinearGradient(
                colors: [Colors.transparent, Colors.black45, Colors.black87],
                begin: Alignment.bottomCenter,
                end: Alignment.topCenter)),
        child: const Text(
          'Welcome',
          style: TextStyle(
              fontSize: 40, fontWeight: FontWeight.w700, color:
Colors.white),
        ),
      ),
    );
  }

  Expanded _buildButtons(WelcomeViewModel viewModel) {
    return Expanded(
      flex: 3,
      child: Container(
        width: double.infinity,
        padding: const EdgeInsets.only(top: 24),
        decoration: const BoxDecoration(
            gradient: LinearGradient(
                colors: [Colors.transparent, Colors.black45, Colors.black87],
                begin: Alignment.topCenter,
                end: Alignment.bottomCenter)),
        child: Column(
          children: [
            OutlineButton(
                title: 'Log In',
                width: 260,
                onPressed: viewModel.openLoginView),
            const SizedBox(
              height: 8,
            ),
            OutlineButton(
                title: 'Sign Up',
                width: 260,
                onPressed: viewModel.openSignupView),
            const SizedBox(
              height: 8,
            ),
            const SizedBox(width: 260, child: OrLineWidget()),
            const SizedBox(
              height: 8,
            ),
            OutlineButton(
                title: 'Sign Up with Google',
                width: 260,
                leading: SvgPicture.asset(AppImages.googleLogo)),
          ],
```

```
        ),
      ),
    );
  }
}
```

```
import 'package:injectable/injectable.dart';
import 'package:role_assist/app/app.router.dart';
import 'package:stacked/stacked.dart';
import 'package:stacked_services/stacked_services.dart';

@injectable
class WelcomeViewModel extends BaseViewModel {
  final NavigationService _navigationService;

  WelcomeViewModel(this._navigationService);

  void openLoginView() {
    _navigationService.navigateTo(Routes.loginView);
  }

  void openSignupView() {
    _navigationService.navigateTo(Routes.signUpView);
  }

}
```