Міністерство освіти і науки України
Державний університет «Одеська Політехніка»
Інститут комп’ютерних систем
Кафедра інформаційних технологій

Острянський Дмитро Альбертович
студентка групи АС-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Приватна соціальна мережа для гейміфікація процесу навчання співробітників компаній.
Підтема 1. Клієнтська частина

Спеціальність:
121 – Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Писаренко Катерина Олександрівна,
канд. техн. наук, доцент

Одеса – 2021

ЗМІСТ

Міністерство освіти і науки України
Державний університет «Одеська Політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Спеціалізація: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Любченко В.В.

«___» _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Острянський Дмитро Альбертович, група АС-161

1. Тема роботи:
Приватна соціальна мережа для гейміфікація процесу навчання співробітників компаній.

Підтема:
Клієнтська частина

Керівник роботи:
Писаренко Катерина Олександрівна, канд. техн. наук, доцент

Затверджені наказом ректора від «25» жовтня 2021 р. № 374-в

2. Зміст роботи: завдання на кваліфікаційну роботу, аналіз предметної області та існуючих рішень, опис алгоритму роботи клієнтської частини, специфікація вимог до системи, проектування системи, програмна реалізація системи, тестування.

3. Перелік ілюстративного матеріалу: Згідно зі слайдами презентації

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання «___» _____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз вимог до веб-застосування	1.09.2021-02.09.2021	Виконано
2	План виконання проекту	03.09.2021-07.09.2021	Виконано
3	Аналіз існуючих аналогів	08.09.2021-09.09.2021	Виконано
4	Проектування веб-застосування	10.09.2021-24.09.2021	Виконано
5	Програмна реалізація веб-застосування	24.09.2021- 29.10.2021	Виконано
6	Тестування веб-застосування	01.10.2021-15.11.2021	Виконано
7	Написання пояснювальної записки	15.11.2021-27.11.2021	Виконано

Здобувач вищої освіти _____ *Д. А. Острянський*

Керівник роботи _____ *К. О. Писаренко*

АНОТАЦІЯ

Метою роботи є підвищення кваліфікації співробітників шляхом додавання до процесу навчання формату гри.

Як результат розроблена клієнтської частина для соціальної мережі з такими елементами гри як рівень користувача, очки навичок, проходження тестів.

Розробку програмного забезпечення виконано на базі застосування мови програмування Flutter.

Ключові слова: соціальна мережа, Flutter/Dart, Redux, мобільна розробка.

ANNOTATION

The purpose of the work is to raise the qualification of the students by adding the game format to the teaching process.

As a result, the client parted for the social network with such elements of the game as the level of the user, eyepieces of skills, and passing tests.

The software was developed on the basis of the Flutter programming language.

Keywords: social network, Flutter/Dart, Redux, mobile development.

ВСТУП

У наш час можна констатувати кризу знань, як ідеологічну, так і інструментальну(незрозуміло, як вчити). Сучасні навчальні онлайн-сервіси відмовляються від парадигми єдиної програми як "освітньої труби", яку користувач має пройти від початку до кінця.

В онлайні користувач менш сфокусований і має більшу свободу вибору. Концепція онлайн-навчання має відображати цю свободу. На етапі знайомства з сервісом дуже важливо допомогти користувачеві розібратися, як саме він буде навчатись, і дати йому зрозуміти, що це буде легко, цікаво і весело.

Важлива відмінність гейміфікації від інших методів взаємодії з користувачем – з її допомогою ми можемо організувати процес знайомства таким чином, щоб потенційний користувач не просто дізнався про те, що ми йому говоримо, а відчув це на своєму досвіді. Велком-сценарій проводить користувача з усіх важливих навчальних елементів, шляхом спонукаючи його зробити ряд легких дій: прочитати короткий текст, відповісти на кілька питань і так далі. І, після такого грамотного введення, у користувача не виникає питань щодо того, як він навчатиметься, і з'являється відчуття, що це буде цікаво.

Також роботодавцям складно контролювати процеси, які пов’язані з розвитком своїх співробітників. Працюючи з дому неможливо показати свій розвиток у тій чи іншій сфері роботи.

Метою роботи є підвищення кваліфікації співробітників шляхом додавання до процесу навчання формату гри. Як результат розроблена клієнтської частина для соціальної мережі з такими елементами гри як рівень користувача, очки навичок, проходження тестів.

У першому розділі роботи проводиться аналіз предметної області та існуючих аналогів.

Другий розділ містить опис алгоритму роботи клієнтської частини, а також опис зв’язку серверної та клієнтської частини.

У третьому розділі була сформульована специфікація функціональних та нефункціональних вимог до розробляємої системи.

Четвертий розділ містить проектування клієнтської частини мобільної розробки соціальної мережі: розглянута архітектура системи, продемонстровані програмні класи та діаграми послідовності.

У п’ятому розділі наведена програмна реалізація системи.

Шостий розділ містить тестування розробленої системи.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

В рамках даної роботи поставлено наступне завдання: користувачі можуть додавати в свою анкету навички і ділитися ними з колегами, проходити тести, отримувати за це очки досвіду, піднімати рівні; також один користувач може викликати іншого на поєдинок, переможцем якого буде той, хто дасть більше правильних відповідей.

1.1 Аналіз предметної області

Дане мобільного застосування розробляється для компаній, які зацікавлені в моніторингу і мотивації рівня розвитку своїх співробітників. Для великих компаній, де працює велика кількість людей, в епоху пандемії і віддаленої роботи стало важко перевіряти якість кваліфікації своїх співробітників.

Для того, щоб співробітники швидше освоювали цю соціальну мережу, було прийнято рішення гейміфіціровать деякою мірою використовувати даний ресурсу. За кожен успішно освоєний навичок, співробітник отримує очки досвіду, кількість яких залежить від складності та ступеня освоєння того чи іншого навичка. При певній кількості очок навичок, користувач отримує новий рівень. Чим вище рівень, тим більше очок потрібно буде добути для переходу на наступний.

Два користувача можуть пройти спільно тест-вікторину, за підсумками якого переможе той, хто дасть більше правильних відповідей. Переможець отримує додаткові очки навичок.

Буде доступна рейтинговая система по навичкам. Сортуватися вона буде за якістю освоєння того чи іншого навичка. Робиться це для того, щоб можна було оперативно замінити співробітника.

1.2 Аналіз існуючих аналогів

Було проведено аналіз аналогів існуючих рішень для корпоративних соціальних мереж.

Таблиця 2.1 - Аналіз існуючих аналогів

Ознаки	Назва системи			
	Convo	DaOffice	Yammer	Магістерська робота
Є мобільний додаток	+	-	+	+
обмін повідомленнями	-	-	+	+
можливості керування групами	-	-	+	+
системи мотивації співробітників	-	+	-	+
обмін файлами	+	+	+	-
профілі співробітників	+	-	+	+

Висновок

Базуючись на аналізі предметної області та можливостях аналогів, було вирішено розробити систему, яка буде вирішувати тільки необхідні проблеми, не буде містити зайвих функцій, та простий інтерфейс, а також наступні функції:

- мобільний додаток
- обмін повідомленнями
- можливості керування групами;
- системи мотивації співробітників;
- обмін файлами;
- профілі співробітників.

2 ОПИС АЛГОРИТМУ РОБОТИ КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1 Архітектурний стиль

Для реалізації цього проекту був обран архітектурний стиль REDUX.

Redux[2] — це архітектура, яка спочатку була створена для JavaScript і використовується в додатках, які створені з використанням reactive frameworks. Redux – це спрощена версія архітектури Flux, створена Facebook. Основні речі в Redux є:

- Єдине джерело правди / single source of truth - весь state вашого додатку зберігається тільки в одному місці (називається store).
- стан доступний тільки для читання/state is read-only - для зміни state програми необхідно відправити actions(дія), після чого створиться новий state
- зміни виконуються за допомогою чистих функцій/pure functions — чиста функція (для простоти, це функція без side effects) приймає поточний state додаток та action та повертає новий state додаток

Деякі Redux-реалізації портовані інші платформи, і Flutter не став винятком. У dartpub є пакет redux, який може бути використаний як у Інтернеті, так і на мобільних платформах з впровадженням додаткового пакету flutter_redux. Як ми вже говорили вище, у Redux немає поділу на локальне та глобальне. Стан завжди глобальний - доступний будь-якому віджету і доступно до зміни через екшени в будь-якому місці системи.

Формально у нас існують такі поняття (рисунку 2.1):

State — модель стану, яка може бути як скалярним, і будь-яким іншим складовим типом.

Action - клас-ідентифікатор події, який зберігає в собі payload для передачі потрібних параметрів.

Reducer — обробник екшенів, має доступ до поточного стану та екшену, який чекає на обробку.

Dispatch – метод виклику екшену, який обробляється одним із ред'юсерів.

Store - дерево стану програми, що комбінує в собі всі ред'юсери, які ми визначили в додатку.

StoreConnector – дає можливість дочірньому віджету отримати доступ до store.

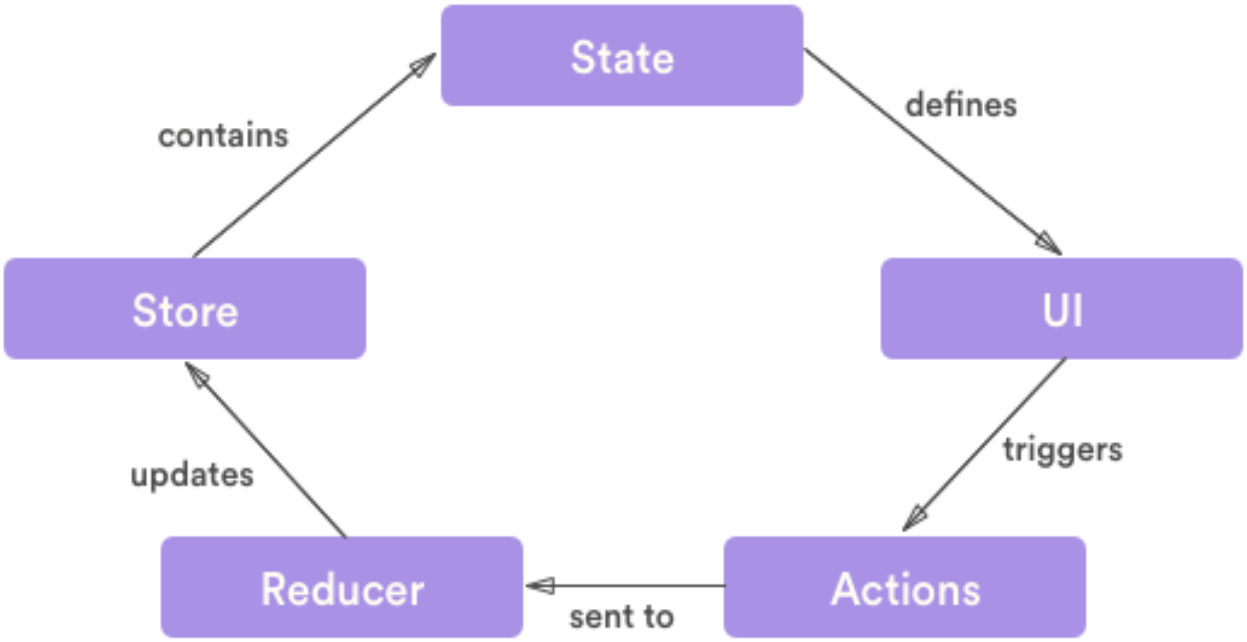

Рисунок 2.1 – Архітектурний стиль Redux

Redux взаємодіє тісно з оновленням та відновлення даних в UI частині, що дуже допомагає в роботі.

2.2 Взаємодія клієнта та сервера

Після того, як був описаний архітектурний стиль, треба описати алгоритм взаємодії клієнтської частини та серверної.

То для даного мобільного-застосунку була обрана трирівнева модель клієнт–сервера. Тому що в контексті систем управління базами даних, то

перший рівень – це клієнт, який дозволяє нам посилати запити з нашого мікроконтролера до нашого сервера. Другий рівень – це двигун СУБД, який інтерпретує запити і реалізує взаємодію між клієнтом і файлової системою, а третій рівень – це сховище даних, приклад показаний на рисунку 2.2

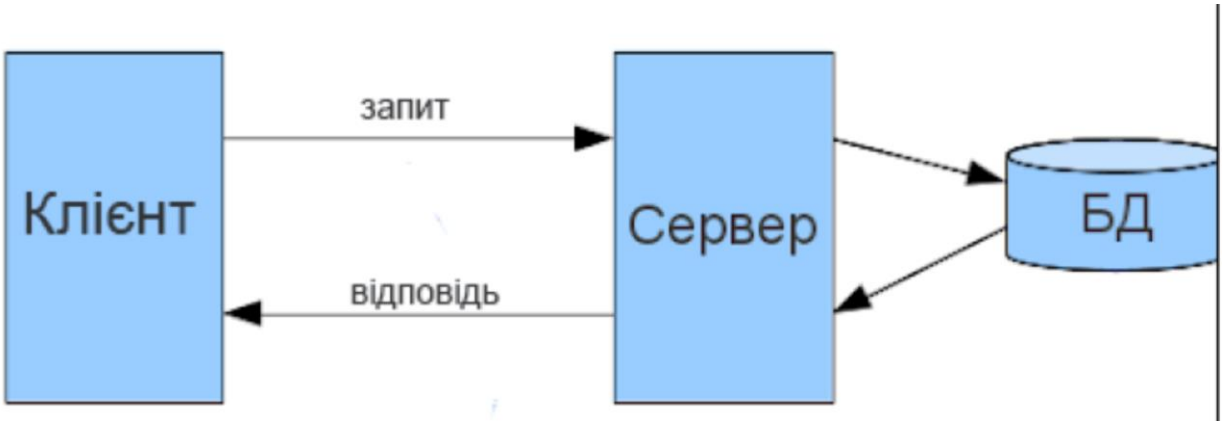

Рисунок 2.2 – Трирівнева модель клієнт–сервера

Розглянемо патерни проектування які будуть використовуватися при розробці додатку.

Патерн «Конструктор» – у класичних об'єктно–орієнтованих мовах програмування «конструктор» – це спеціальний метод, який використовується для ініціалізації створеного об'єкту, як тільки для нього виділили пам'ять.«Конструктори» об'єктів використовуються для створення певних типів об'єктів – готує об'єкт до використання і засвоєння аргументів, які «конструктор» може використовувати для визначення значень властивостей елементів і методів, коли об'єкт вперше створюється.

Шаблон Singleton – це шаблон, який обмежує клас для створення його декількох об'єктів. Це не що інше, як спосіб визначення класу. Клас визначається таким чином, що при повному виконанні програми або проекту створюється тільки один екземпляр класу. Він використовується, коли тільки один екземпляр класу необхідний для управління процесом під час виконання. Клас Singleton не повинен мати кілька екземплярів в будь–якому випадку і за всяку ціну. Цей шаблон включає в себе один клас, який

відповідає за створення об'єкта, забезпечуючи при цьому створення тільки одного об'єкта. Цей клас надає спосіб доступу до свого єдиного

Тож ми використовуємо Redux запит від сервера ми будемо обробляти за допомогою ReduxEpic, оскільки він призначений для асинхронної обробки даних, та за допомогою нього ми зможемо реалізувати обробку даних без зупинення системи та в state передавати вже оброблені дані, як приклад ми можемо побачити на рисунку 2.3 де ми беремо з серверу список тестів

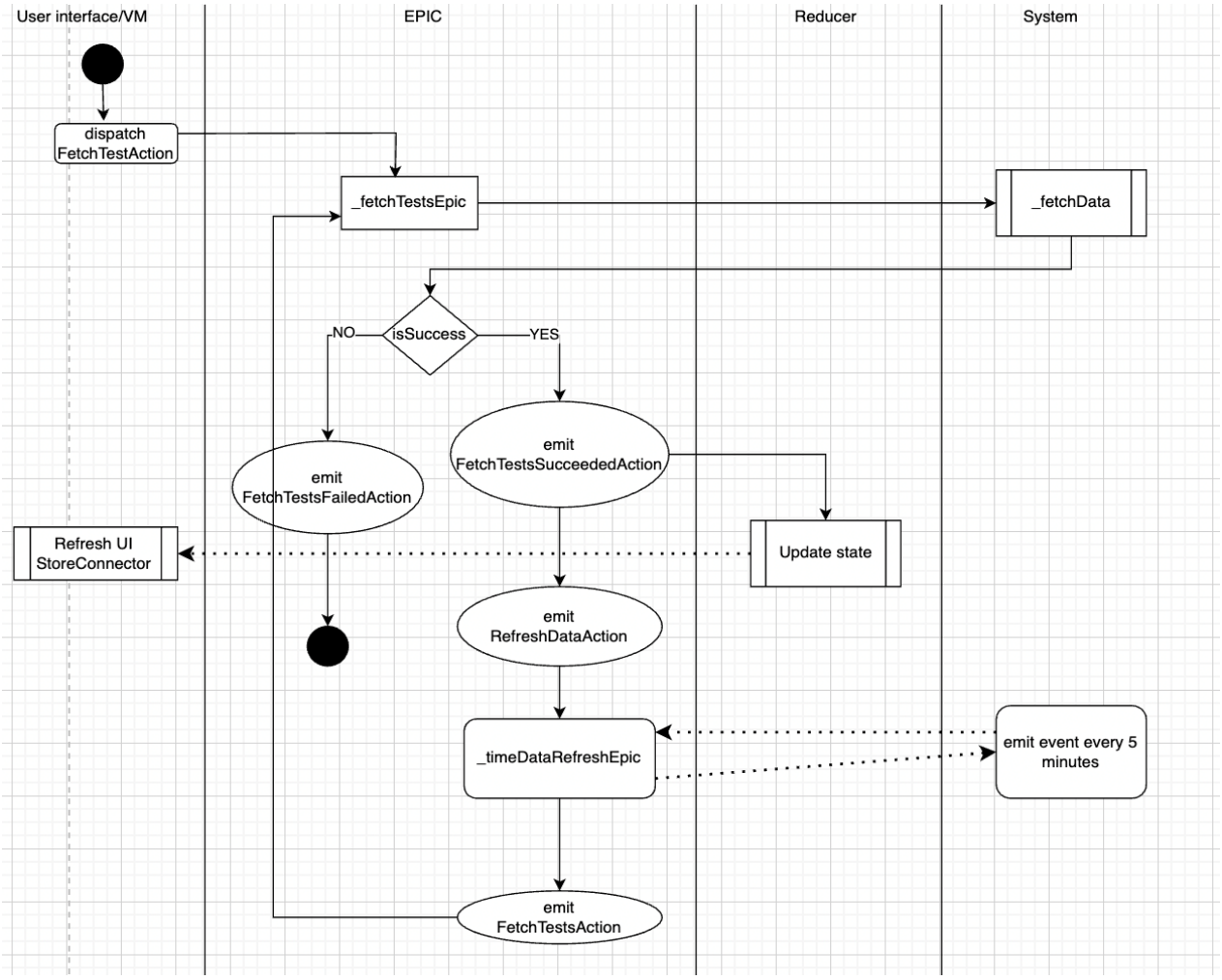

Рисунок 2.3 – Запит на список тестів за допомогую ReduxEpic

2.3 Рівень та очки навичок користувача

Один з засобів гейміфікувати якийсь процес - введення рейтингової системи.

Рейтинг — числовий чи порядковий захід, що вказує на значущість чи важливість певного об'єкта чи явища. Механіка рейтингу пов'язана з механікою очок і часто з механікою рівнів користувача. Рейтинг без очок неможливий – система не зрозуміє порядок у якому користувачі мають з'являтися у рейтингу без рівнів, тому кожен користувач отримує при старті стартову кількість очок що допомагає в системі уникнути помилок коли користувач без очок.

Однак треба пам'ятати, що користувачу легше не приймати участь в гейміфіцірованної системі спочатку, ніж після того як він провів в системі багато часу.

Соціальна мережа буде гейміфіцірувати процес розвитку завдяки рейтингової системи. В кожного користувача буде свій рівень та очки, який він зможе отримати при проходженні тестів. Треба використати формулу підрахунку кількості очок за рівень, яка буде давати низькорівневим користувачам здобувати новий рівень швидше, ніж вискорівневим:

$$\frac{(lvl*(lvl+1))}{2} * const, \quad (2.1)$$

де lvl - рівень користувача

const - коефіцієнт для підвищення кількості очок.

В даній таблиці 2.1 наведені деякі результати розрахунку за формулою, з const = 20.

Таблиця 2.1 - Результати розрахунку за формулою

Рівень	Кількість очок (КО)	Разність ($КО_n$ - $КО_{n-1}$)
2	60	60
3	120	80

4	200	100
5	300	120

Продовження таблиці 2.1

Рівень	Кількість очок (КО)	Разність ($КО_n$ - $КО_{n-1}$)
6	420	140
7	560	160

С результатів розрахунку бачимо, що для досягнення слідуючого рівня, треба отримати більше очок, ніж для попереднього.

Висновок

У данному розділі було представлено архітектурний стиль продукту, що розроблюється. Було обрано архітектурний стиль Redux.

Крім того представлено діаграму запиту на список тестів за допомогую ReduxEpic.

Також описано рівень та очки навичок користувача.

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ

3.1 Діаграма варіантів використання

Один із способів опису вимог до розроблюваної системі [3] - складання діаграми варіантів використання.

Варіанти використання є описом того, що повинна робити система. На рисунку 3.1 представлена діаграма варіантів використання, яка описує основні функції розроблюваного мобільного застосування.

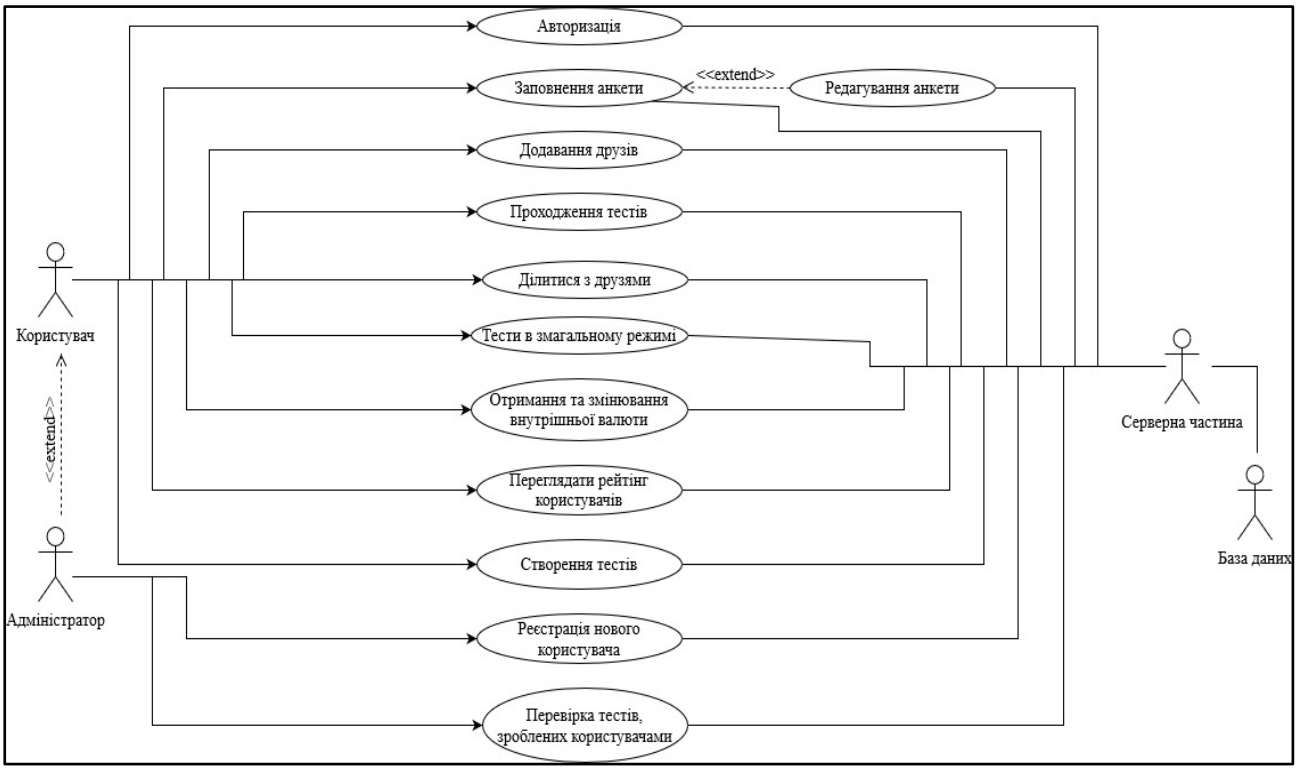

Рисунок 3.1 - Діаграма варіантів використання

Актор з роллю Користувач має наступні можливості:

- Авторизуватись
- Заповнити анкету
- Редагувати анкету
- Додавати друзів
- Проходити тести
- Ділитися результатами

- Проходити тест в змагальному режимі
- Отримувати і обмінювати внутрішню валюту сервісу

Актор з роллю Адміністратор має всі ті ж можливості, але ще і додаткові:

- Реєстрація нового користувача
- Перевірка тестів, зроблених користувачами
- Створення тестів

Актор Серверна частина обробляє запити від користувачів(клієнтська частина), проводить перевірку та валідацію даних, виконує запити до бази даних, повертає результат обробки запиту.

Актор База даних виконує відповідні операції створення, редагування та видалення даних.

3.2 Сценарії варіантів використання

На діаграмі варіантів використання соціальна мережа має 12 варіантів використання. Нижче представлені опис кожного з них.

Опис варіанту “Авторизація”

Опис: користувач відправив запит на авторизацію в соціальній мережі.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач має внести дані(логін та пароль).
2. Натискає на кнопку “Log In”.
3. Сервер отримав запит на авторизацію.
4. Сервер перевіряє дані.
5. Сервер перевіряє дані в базі даних.
6. Сервер повертає користувачу інформацію про успіх.

Альтернативний сценарій:

1а. Не прошла валідація логіну.

2а. Помилка серверной частини.

Опис варіанту “Заповнення анкети”

Опис: Користувач додав нову інформацію до свого профілю.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач переходе на сторінку “Заповнення”
2. Заповнює всі необхідні поля
3. Після успішною валідації, користувач надсилає дані до сервера
4. Сервер отримав запит на додавання певної інформації до свого профілю в соціальної мережі.
5. Користувач отримує повідомлення що все пройшло успішно.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Помилка серверной частини.

Опис варіанту “Редагування анкети”

Опис: Користувач намагається відновити свій профіль.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач переходе на сторінку “Редагування”
2. Після того як користувач змінив якісь дані, з'являється можливість відправити дані на сервер
3. Користувач підтверджує зміну даних
4. Сервер отримав запит на оновлення певної інформації до свого профілю в соціальної мережі.

5. Користувач отримує повідомлення що все пройшло успішно.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Додавання друзів”

Опис: Користувач намагається додати нового друга

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач переходе на профіль іншого користувача

2. Натискає на кнопку “Додати в друзі”

3. Сервер отримав запит на додавання користувачем нового друга.

4. Користувач отримує позитивну відповідь.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Проходження тестів”

Опис: Користувач проходить тест за навичками.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач отримує список тестів

2. Обирає потрібний йому тест, та потрапляє на сторінку тесту

3. Відповідає на всі запитання

4. Підтверджує відповіді та відправляє до серверу

5. Сервер отримав запит на проходження користувачем теста.

6. Сервер повертає користувачу результат проходження тесту.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

3а. json-структура не відповідає серверу.

Опис варіанту “Ділитися з друзями”

Опис: Користувач вирішив поділитися результатами тесту зі свома друзями.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач на сторінці результату тесту
2. Натискає на кнопку “Поділитися з друзями”
3. Сервер отримав запит на додавання користувачем нового запису з результатами тесту.
4. Користувач отримує повідомлення що все пройшло успішно.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Тести в змагальному режимі”

Опис: Два користувача вирішили пройти один тест та вирішити хто краще володіє темою.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач переходить на потрібну сторінку
2. Проходить тест та відправляє запит до серверу
3. Сервер отримав запит на проходження користувачами теста.
4. Користувач отримує повідомлення що все пройшло успішно

5. Після того як другий користувач проходить тест, обидва користувача можуть побачити їх результат

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Отримання та змінювання внутрішньої валюти”

Опис: Користувач отримав новий уровень та вирішив змінити внутрішню валюту на бонус.

Актори: користувач, серверна частина, представник адміністрації команії.

Основний сценарій:

1. Після того як користувач отримав потрібну суму для виводу в бонуси, він переходить на сторінку профіля
2. Натискає на кнопку обмін валюти в бонуси
3. Вводить потрібну суму
4. Сервер отримав запит на зміну внутрішньої валюти на бонус.
5. Користувач отримує повідомлення що все пройшло успішно.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Переглядати рейтинг користувачів”

Опис: Користувач соціальної мережі вирішив переглянути рейтінг володіння навиком.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач переходить на сторінку рейтингу
2. Користувач відправляє запит на таблицю рейтингу до сервера
3. Сервер отримав запит на перегляд рейтингу.
4. Сервер повертає користувачу відсортований список користувачів з очками володіння (очки, які користувачі отримали за проходження тестів).
5. Дані відображаються на сторінці

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

4а. Користувачів, які володіють цім навиком, не було знайдено.

Опис варіанту “Створення тестів”

Опис: Користувач додає тести для певного навика.

Актори: користувач, серверна частина.

Основний сценарій:

1. Користувач вибирає потрібний навик
2. Натискає на кнопку додавання тесту до цього навику
3. Користувач потрапляє до потрібної сторінки та вводить дані для тесту
4. Після узгодження відправляє дані до сервера
5. Сервер отримав запит на створення тесту.
6. Користувач отримує повідомлення що все пройшло успішно.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Реєстрація нового користувача”

Опис: Адміністратор додає нового працівника компанії.

Актори: користувач з роллю адміністратор, серверна частина.

Основний сценарій:

1. Адміністратор переходить до потрібної сторінки
2. Вводить потрібні дані нового користувача
3. Відправляє запит до серверу
4. Сервер отримав запит на реєстрацію працівника.
5. Адміністратор отримує повідомлення що все пройшло успішно.

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

Опис варіанту “Перевірка тестів, зроблених користувачами”

Опис: Користувач з роллю адміністратор схвалив тест, який було створено користувачем.

Актори: користувач з роллю адміністратор, серверна частина.

Основний сценарій:

1. Адміністратор отримує список тестів яких потрібно перевірити
2. Переходить до обраного тесту
3. Після успішної провірки відправляє запит до серверу
4. Сервер отримав запит з повідомленням про схвалення тесту.
5. Адміністратор отримує повідомлення що все пройшло успішно
6. Сервер повідомляє автора тесту про схвалення

Альтернативний сценарій:

1а. Дані не пройшли валідацію. Користувач отримує негативний результат.

2а. Користувач отримує негативний результат з серверной частини.

3.3 Нефункціональні вимоги

Нефункціональні вимоги визначають властивості та обмеження до розроблюваного ПЗ.

Під час формування нефункціональних вимог були визначені наступні сценарії якості розроблювальної системи.

Надійність:

1. Система зберігає 95% введених користувачем даних.

2. При неможливості зберегти дані в БД система зберігає дані в кеш після чого робить додаткові запити.

Зручність використання:

1. Зрозумілі повідомлення при успішному виконанні операції або невірно введеним даним.

Продуктивність:

1. Час додавання нових даних складає не більш 2 сек.

2. Час оновлення даних складає не більше 3 сек.

Безпека:

1. Система робить резервне копіювання після кожного запиту додавання або редагування рядків.

Вимоги до реалізації продукту:

- клієнтська частина повинна бути реалізована мовою Flutter.
- Інтерфейс повинен бути зручним та простим.
- програмне забезпечення повинно працювати на актуальних версіях браузерів або смартфонів.

Висновок

В даному розділі було проведено специфікацію вимог до системи, що розробляється. Була побудована діаграма варіантів використання, де показано основні варіанти використання системи користувачів з різними ролями. Наступний крок – описи прецедентів. На даному етапі був описаний алгоритм дій для кожного варіанту використання: як основний сценарій, так і альтернативний, на випадок, якщо якась дія користувача або стан системи призведе до негативного сценарію. Останнім етапом став опис нефункціональних вимог до системи, в яких були описані вимоги системи для забезпечення надійності, зручності використання, продуктивності та безпеки системи.

4 ПРОЕКТУВАННЯ СИСТЕМИ

Проектування [1] - це етап життєвого циклу розроблення програмних систем, наступний після інженерії вимог. Завданням цього етапу є перетворення побажань замовників системи, які ми подали як моделі вимог, у проектні рішення, що забезпечать здійснення згаданих побажань у формі відповідної системи програмування. Таким чином, під час проектування виконується трансформація простору вимог у простір проектних рішень. При цьому можна виділити процеси, котрі можна вважати відносно незалежними одне від одного і виконувати як послідовно, так і паралельно, окремими командами виконавців. Це такі процеси:

- концептуальне проектування полягає в уточненні розуміння й узгодження деталей вимог;
- архітектурне проектування полягає у визначенні головних структурних особливостей системи, яку будують;
- технічне проектування полягає у відображенні вимог середовища функціонування і розроблення системи та у визначенні всіх конструкцій як композицій компонент

4.1 Архітектура програмного продукту

Для даної системи була обрана концепція чистої архітектури.

Дотримуватись принципів чистої архітектури – значить забезпечити зручність тестування, підтримки та модернізації програми. Розуміння архітектури та state management – це база, необхідна початківцю для успішної командної роботи.

Чиста архітектура[14] - це концепція побудови архітектури систем, запропонована Робертом Мартіном. Концепція передбачає побудову додатку у вигляді набору незалежних шарів на рисунку 4.1, що полегшує тестування, зменшує зв'язність і робить додаток більш простим для розуміння.

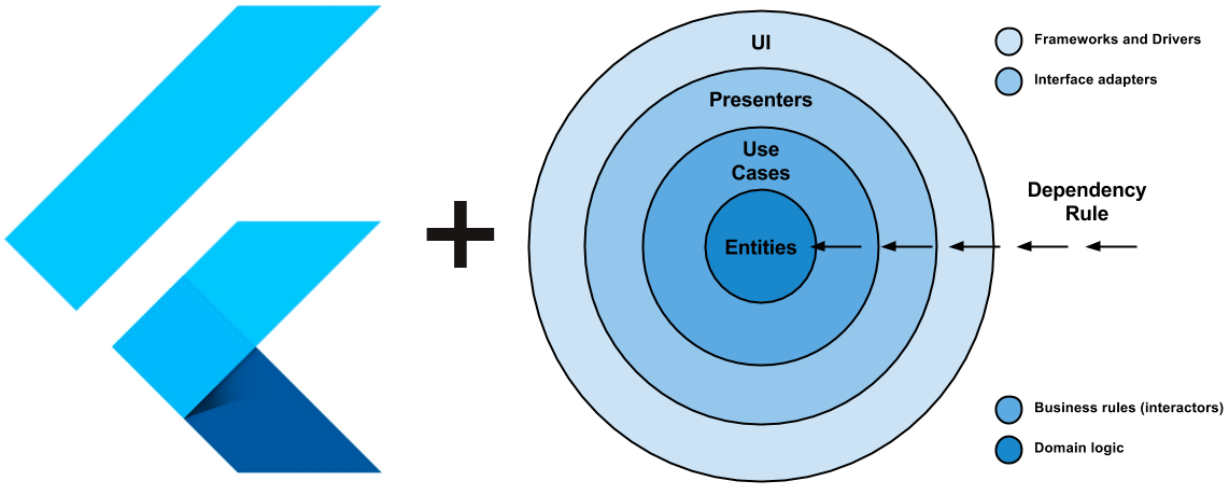

Рисунок 4.1 - Набор незалежних шарів

Зазвичай додаток складається з чотирьох шарів:

- Internal – шар програми, у якому відбувається використання залежностей;
- Presenters – шар, в якому описується візуальна складова вікна та керування його станом;
- Domain – шар бізнес-логіки;
- Data – шар, у якому описується робота з джерелами даних (інтернет-запит чи база даних).

Також самі шари поділяються на елементи:

- data – елемент шару data до роботи з даними. На цьому рівні, наприклад, описуємо роботу із зовнішнім API;
- repository – елемент шару data, який створює та повертає дані з Data-шару у вигляді Entity-об'єкта;
- use case – елемент шару domain, що відповідає за деталізацію, опис дії, яку може зробити користувач системи;
- presenter – елемент шару presentation, цьому рівні описується state management;

- UI – елемент шару presentation, цьому рівні описуються візуальні елементи вікна.

Шар Presenters нічого не знає про те, звідки з'являються дані, які він використовує, шар data не знає, хто і як використовувати дані, що він надає. Це дозволяє легко вносити зміни - наприклад, ми можемо переїхати з REST на GraphQL і не змінити жодного рядка коду в шарі present.

4.2 Діаграма програмних класів

Після формування архітектури системи, слід перейти до формування статичної структури у вигляді класів програми. Цю задачу виконує діаграма програмних класів (рис. 4.2).

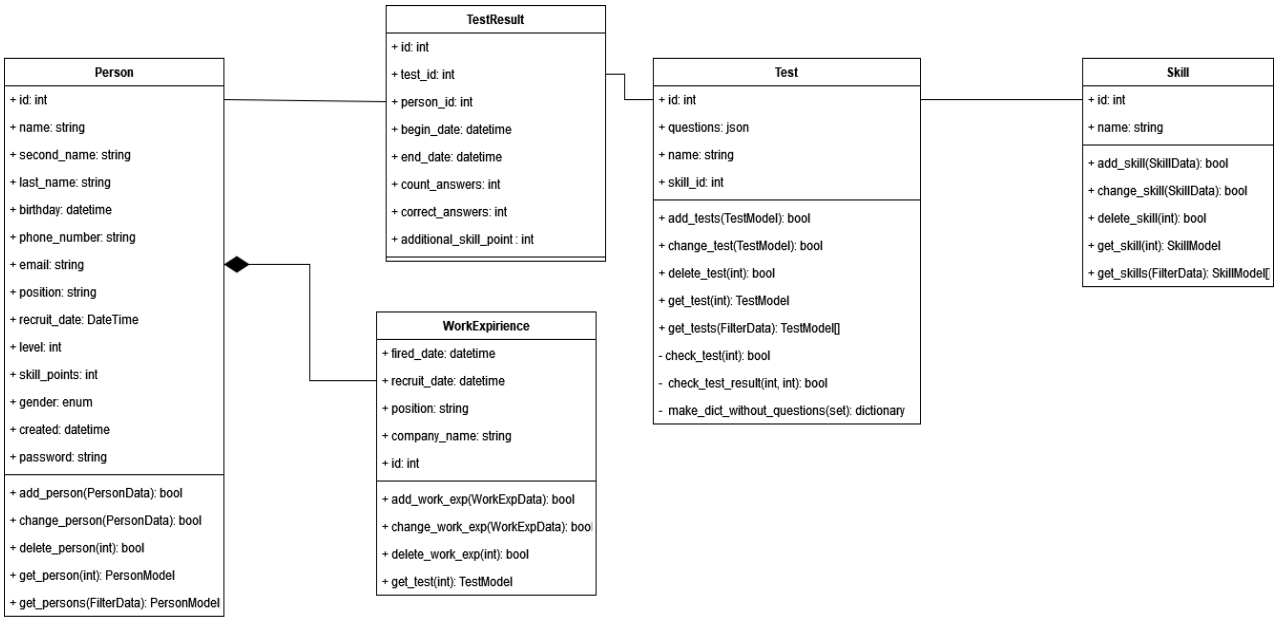

Рисунок 4.2 - Діаграма классів

Клас Person містить інформацію та функції про кандидата на працевлаштування.

Клас Employee містить інформацію та функції для обробки даних про співробітника, який був кандидатом на працевлаштування в компанії.

Клас WorkExperience містить інформацію та функції для обробки інформації про попереднє місце роботи людини.

Клас Test містить інформацію та функції для обробки даних про тести.

Клас Skill містить інформацію та функції для обробки даних про навички.

4.3 Діаграми послідовності

В даному підрозділі було побудовано Діаграми послідовності. Вона застосовується тоді, коли потрібно подивитися на поведінку кількох об'єктів у межах одного прецеденту. Діаграми послідовності використовуються для представлення взаємодії об'єктів.[9], яка була побудована на протязі практики.

Діаграма послідовності відображає поведінку об’єктів протягом часу виклика функції.

На рисунку 4.3 відображена діаграма послідовності прецеденту “Отримання тестів”.

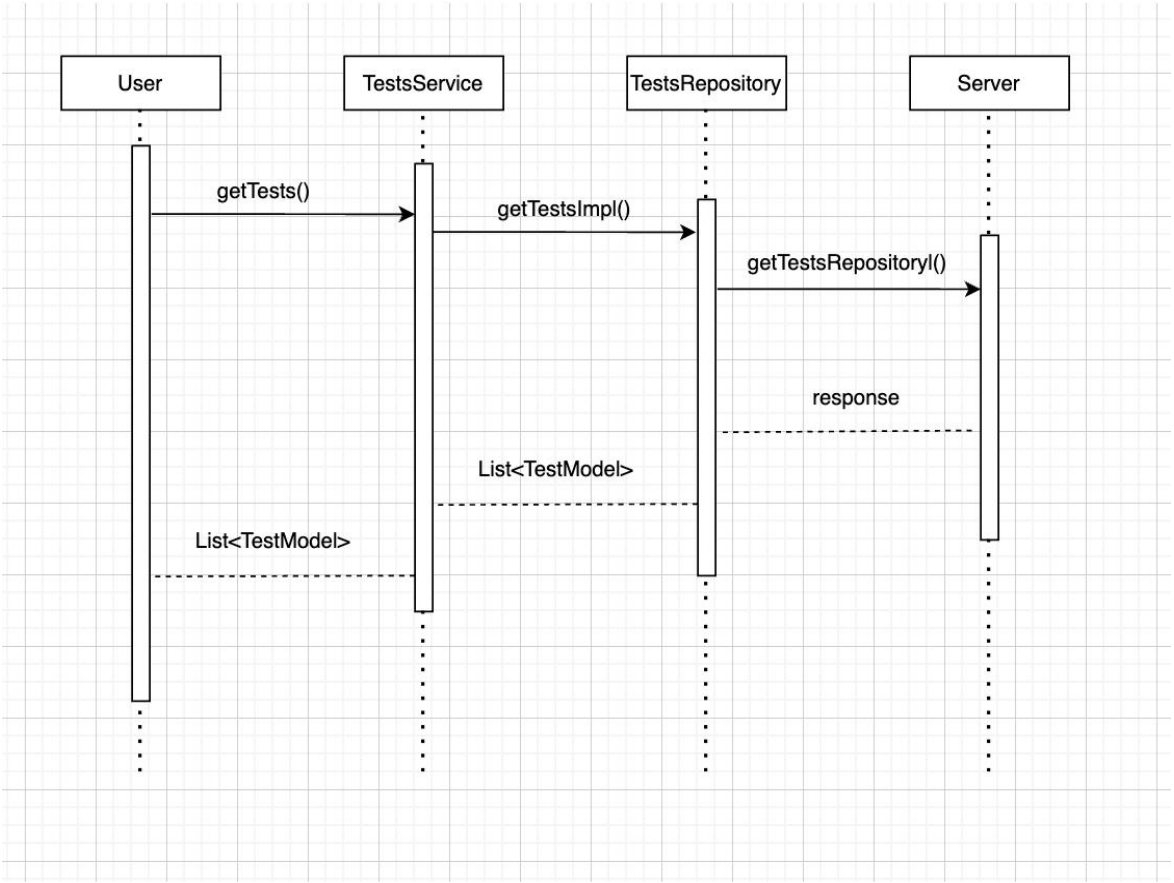

Рисунок 4.3 - Діаграма послідовності прецеденту “Отримання тестів”

Об’єкт User відправляє об’єкту TestService запит на отримання тестів, за допомогою метода getTests(). TesRepository після певної обробки даних, відправляє кінцеві дані до Server після чого сервер відправляє нам список всіх тестів де на кінці User отримує дані в потрібному форматі данних.

Розглянемо діаграму послідовності прецеденту “Додавання нового користувача” на рисунку 4.4.

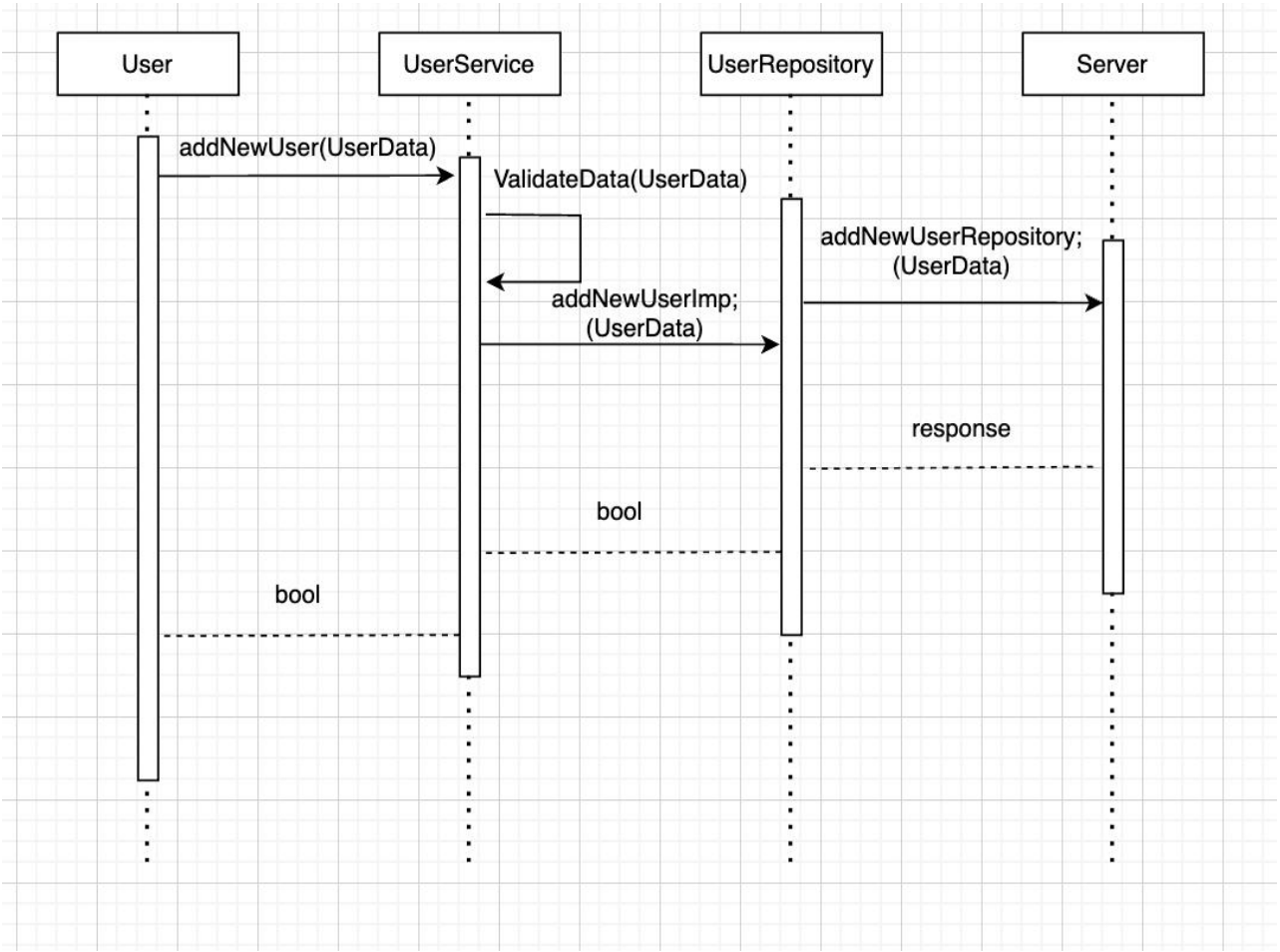

Рисунок 4.4 - Діаграма послідовності прецеденту “Додавання нового користувача”

Об’єкт User відправляє об’єкту UserService запит на додавання нового користувача, за допомогою метода addNewUser(UserData). UserService йде валідація на дані, після повної обробки даних UserRepository відправляє кінцеві дані до Server після чого сервер відправляє нам список результат.

4.4 Проектування інтерфейсу користувача

У цьому розділі зображені макети графічного інтерфейсу, що були створені перед початком розробки програми, а не графічний інтерфейс закінченої програми. На рисунку 4.5 зображено початковий екран системи. Його призначення полягає у тому, щоб користувач увійшов у систему під існуючим аккаунтом або мав можливість звязатись з адміністратором.

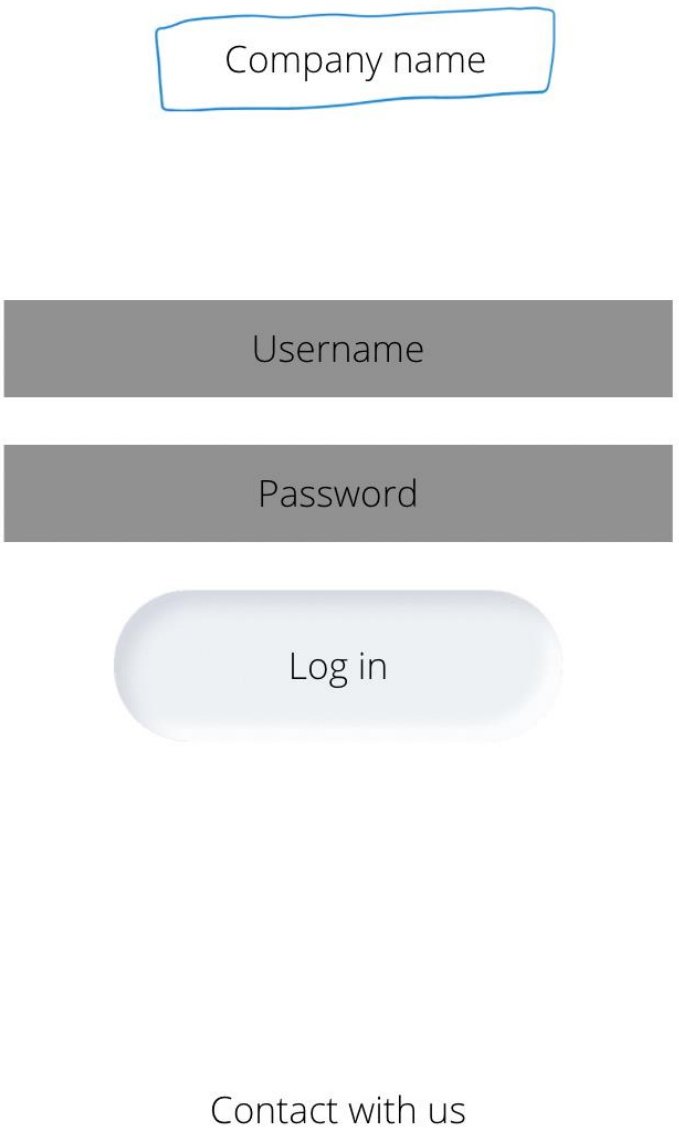

Рисунок 4.5 - Макет початкового екрану

Якщо користувач натискає кнопку «Log in», він попадає на головну сторінку, який зображений на рисунку 4.6. У випадку, коли користувач натискає кнопку «Contact with us», він бачить діалог на якому потрібно ввести свої дані та коментар 4.7.

Рисунок 4.6 - Головна

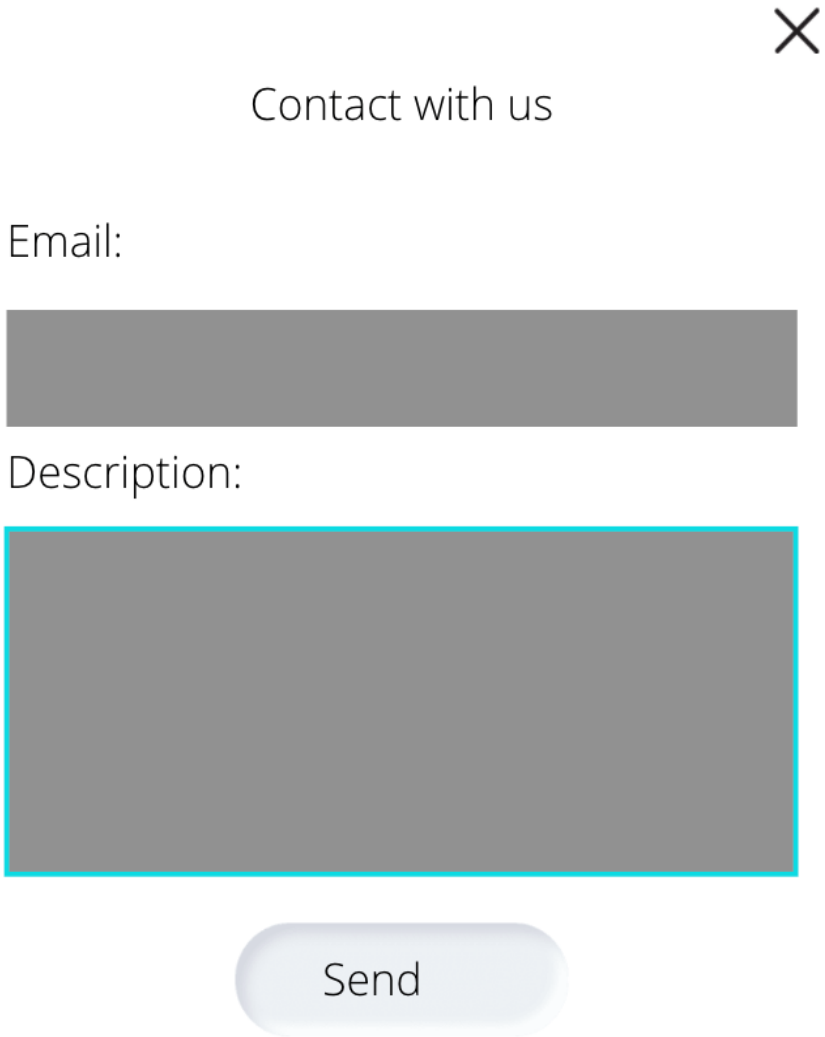

Рисунок 4.7 - Діалог «Contact with us»

Також після того як користувач потрапляє на головну сторінку перший раз, йому спливає вікно онбордингу де йому розповідається про всі особливості застосунку. На головному екрані користувач бачить інформацію про себе, та головні події в компанії, також є можливість переходити на інші сторінки, друга на рисунку 4.8 сторінка це сторінка списку тестів де є можливість знайти або відфільтрувати потрібні тести

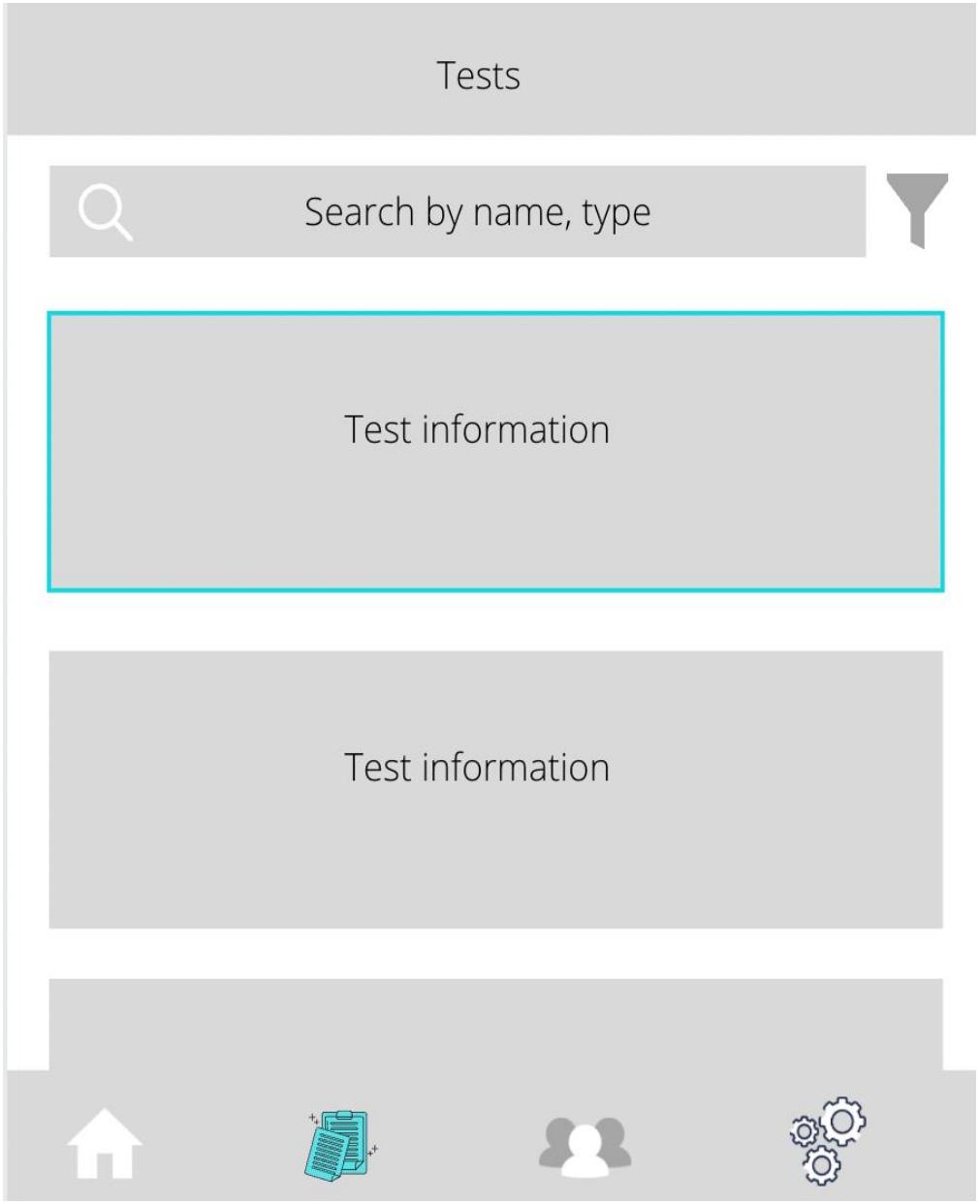

Рисунок 4.8 - Сторінка тестів

Далі ми бачимо сторінку на рисунку 4.9 рейтингу всіх робітників компанії та їх інформацію з можливістю перейти на певного юзера та вже там прочитати більше детальну інформацію про него, та на кожній сторінці ми можемо бачити нижню панель навігації для більш комфортного переходження по додатку.

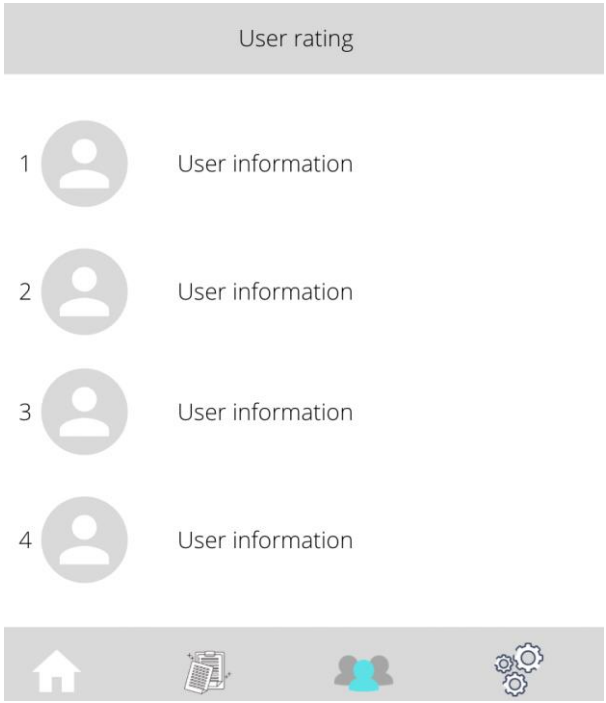

Рисунок 4.9 - Сторінка рейтингу користувачів

На рисунок 4.10 зображена сторінка користувача де буде більше детальна его інформація та історія його проходження

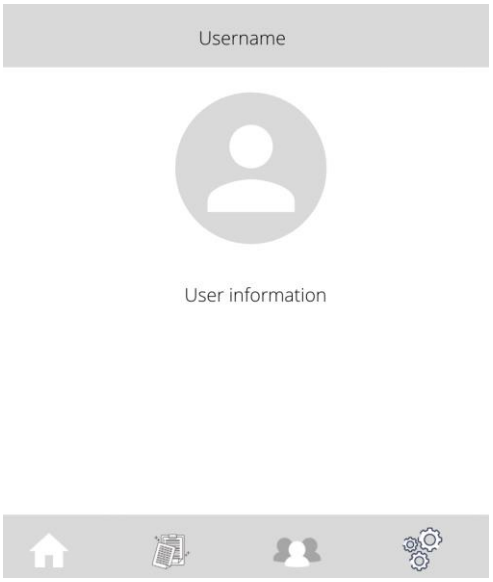

Рисунок 4.10 - Сторінка користувача

Також користувач має можливість перейти на сторінку налаштувань (рисунок 4.11), де він може підібрати потрібні йому налаштування або вийти з аккаунту

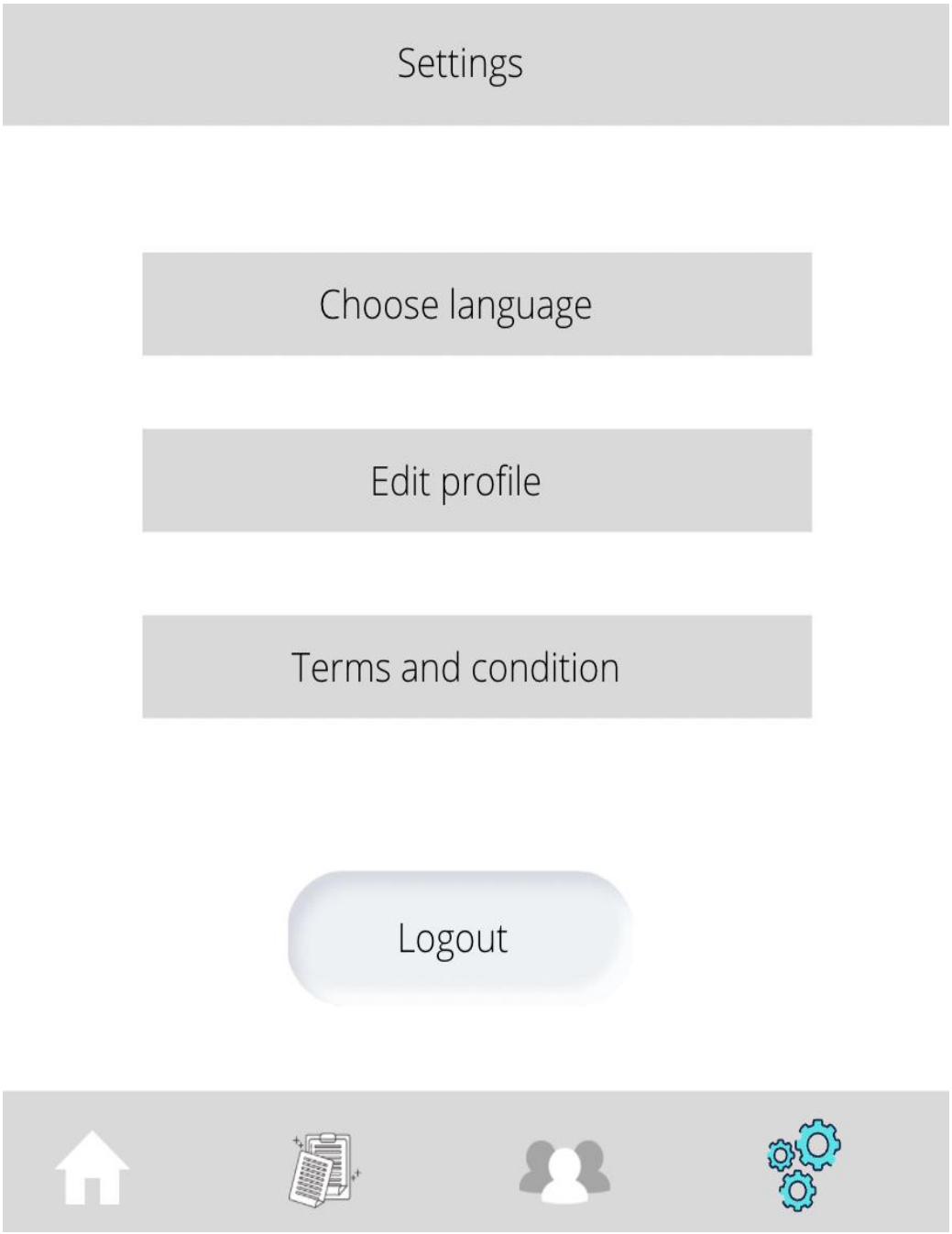

Рисунок 4.11 - Сторінка користувача

На рисунку 4.12 знаходиться сторінка самого тесту, де у кожного тесту буде его назва та питання, з декількома варіантами на вибір

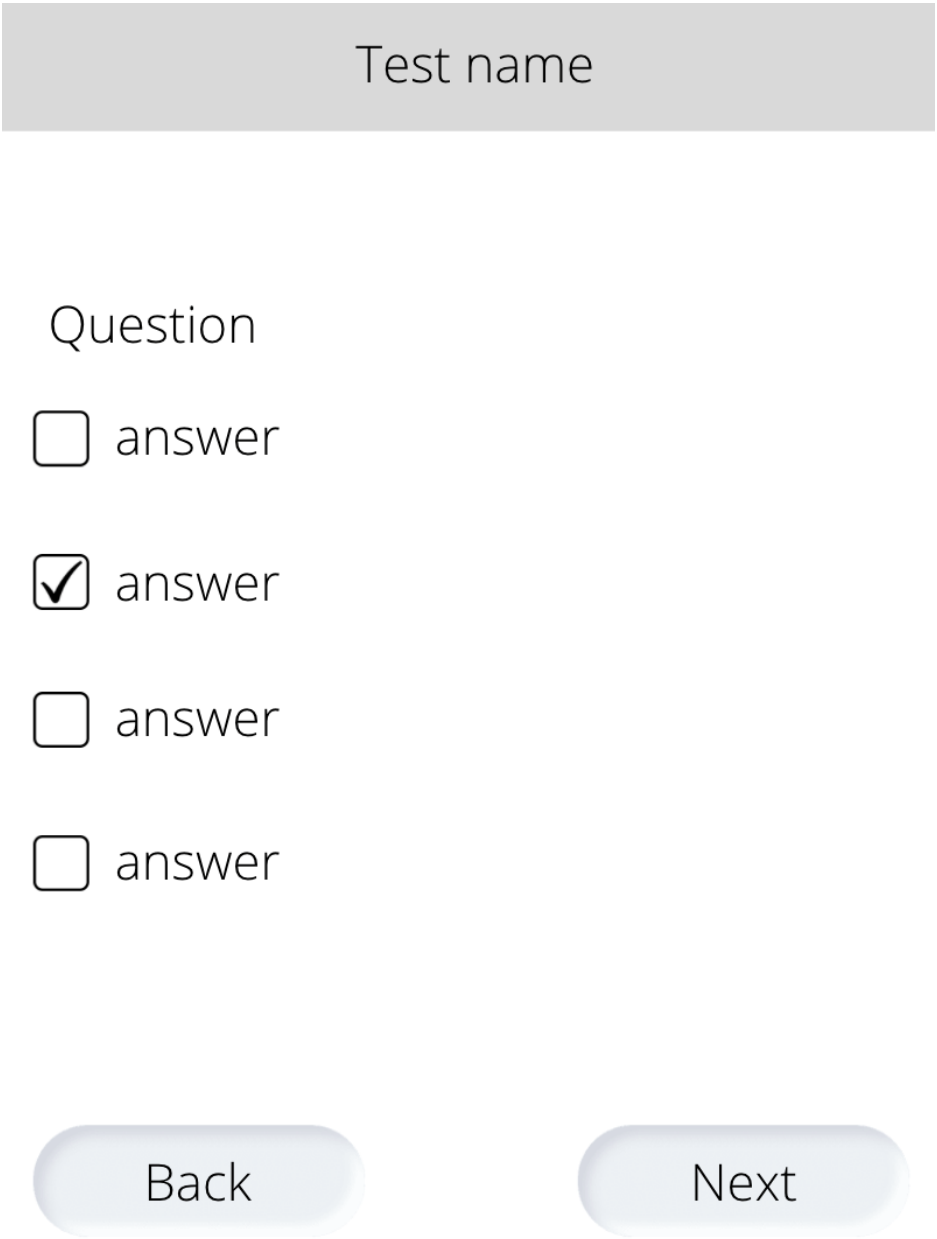

Рисунок 4.12 - Сторінка тесту

Та після успішного проходження тесту користувач попадає на сторінку результату рисунок 4.13 де він може побачити свій результат та перейти знову до головної сторінки тестів

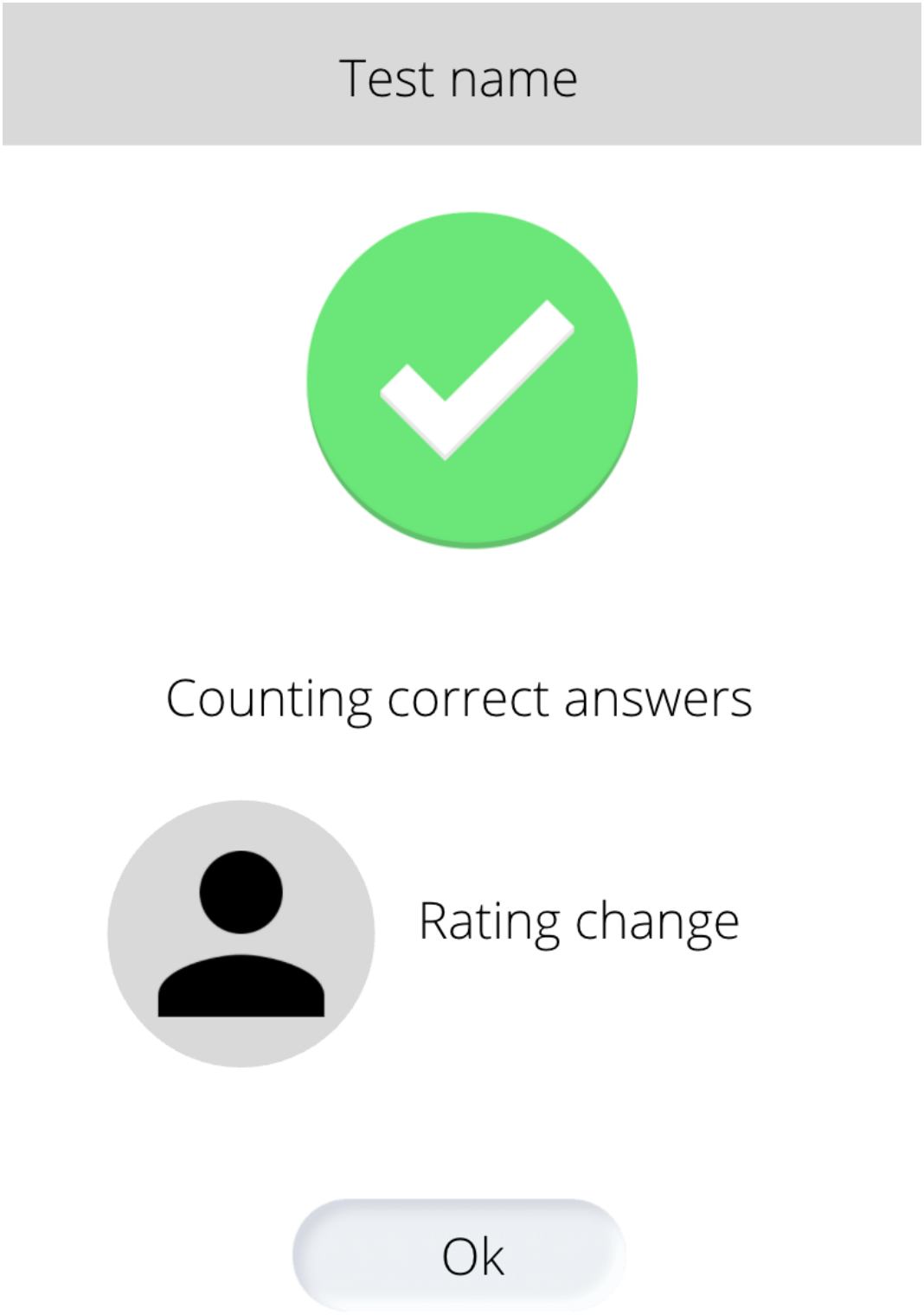

Рисунок 4.13 - Сторінка успішного проходження тесту

Висновок

В даному розділі було спроектовано мобільне-застосування, враховуючи прецеденти, які були створені в попередньому розділі.

По-перше було створено архітектуру мобільного-застосування, яку потрібно зробити. Це буде слоїста архитектура, тому що в каждого рівня своя ступінь відповідальності.

Також спроектована діаграма програмних класів для веб-застосування та створення діаграми послідовностей для двох прецедентів - отримання тестів та додавання нового користувача.

Останнім кроком було створення створення екземпляра інтерфейсу мобільного-застосування.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Для реалізації клієнтської частини мобільного застосування було вирішено використовувати фреймворк Flutter який розроблений на мові програмування Dart.

5.1 Огляд використовуваних технологій

Dart [7] -це оптимізований для клієнтів мову для розробки швидких додатків на будь-якій платформі. Його мета - запропонувати найбільш продуктивний мову програмування для багатоплатформного розробки в поєднанні з гнучкою платформою часу виконання для фреймворків додатків.

Мови визначаються їх технічної оболонкою - вибором, зробленим під час розробки, який визначає можливості і сильні сторони мови. Dart розроблений для технічної оболонки, яка особливо підходить для розробки клієнтів, приділяючи пріоритетну увагу як розробці (гаряча перезавантаження з відстеженням стану менш секунди), так і високоякісному виробничого досвіду для широкого спектра цілей компіляції (веб, мобільні пристрої та настільні комп'ютери).

Dart також становить основу Flutter. Dart надає мову і середу виконання, які використовуються в додатках Flutter, але Dart також підтримує багато основні завдання розробника, такі як форматування, аналіз і тестування коду.

Мова Dart безпечна по типу; вона використовує перевірку статичного типу, щоб гарантувати, що значення змінної завжди відповідає статичному типу змінної. Іноді це називають набором звуків. Хоча типи є обов'язковими, анотації типів не є обов'язковими через виведення типу. Система типізації Dart також є гнучкою, дозволяючи використовувати dynamic в поєднанні з перевірками під час виконання, що може бути корисно під час експериментів або для коду, який повинен бути особливо динамічним.

Dart пропонує надійну нульову безпеку, що означає, що значення не можуть бути нульовими, якщо ви не скажете, що вони можуть бути такими. Володіючи надійної нульовий безпекою, Dart може захистити вас від нульових винятків під час виконання з допомогою статичного аналізу коду. На відміну від багатьох інших мов з нульовим рівнем безпеки, коли Dart визначає, що змінна не допускає значення NULL, ця змінна завжди не допускає значення NULL. Якщо ви перевірите свій запущений код в компіляторі, ви побачите, що неприпустимість значення NULL зберігається під час виконання (отже, безпеку нульового значення).

Мова Dart, включаючає бібліотеки, асинхронні виклики, типи, що допускають і не допускають значення NULL, синтаксис стрілок, генератори, потоки і методи отримання. Dart має багатий набір основних бібліотек, які забезпечують найважливіше для багатьох повсякденних завдань програмування

Dart Бібліотеки:

- Вбудовані типи, колекції та інші основні функції для кожної програми Dart (dart:core)
- Багатіші типи колекцій, такі як черги, зв’язані списки, хеш -карти та двійкові дерева (dart: collection
- Кодери та декодери для перетворення між різними представлення даних, включаючи JSON та UTF-8 (dart:convert)
- Математичні константи та функції та генерація випадкових чисел
- Підтримка файлів, сокетів, HTTP та інших засобів вводу-виводу для не веб-додатків (dart: io)
- Підтримка асинхронного програмування з такими класами, як Future і Stream (dart: async)
- Списки, які ефективно обробляють дані фіксованого розміру (наприклад, 8-байтові цілі числа без знака) та числові типи SIMD (dart: typed_data)

- Інтерфейси іноземних функцій для сумісності з іншим кодом, що представляє інтерфейс у стилі C (dart: ffi)
- Паралельне програмування з використанням ізолятів - незалежних працівників, які схожі на потоки, але не мають спільної пам’яті, спілкуються лише за допомогою повідомлень (dart: isolate)
- Елементи HTML та інші ресурси для веб-додатків, яким потрібно взаємодіяти з браузером та об’єктною моделлю документа (DOM) (dart: html)

Flutter[6] - це мобільний розробник SDK з відкритим вихідним кодом, який можна використовувати для створення оригінальних програм Android та iOS з однієї бази кодів. Flutter існує з 2015 року, коли Google представив його, і залишався на стадії бета -тестування до його офіційного запуску в грудні 2018 року. З тих пір кайф навколо Flutter посилюється. Віджети Центральна ідея флатера є використання віджетів. Завдяки поєднанню різних віджетів розробники можуть створити весь інтерфейс користувача. Кожен з цих віджетів визначає структурний елемент (наприклад, кнопку або меню), стилістичний елемент (шрифт або колірну схему), аспект макета (наприклад, відступ) та багато інших.

Зауважте, що Flutter не використовує віджети OEM(Original equipment manufacturer) що означає «виробник оригінального обладнання», а розробники постачальників із власними готовими віджетами, які виглядають рідними для програм Android або iOS (слідуючи Material Design або Cupertino). Природно, розробники також можуть створювати власні віджети.

Flutter також надає розробникам уявлення про реактивний стиль. Щоб уникнути проблем із продуктивністю, пов'язаних із використанням компільованої мови програмування як мосту JavaScript, Flutter використовує Dart. Він компілює Dart завчасно у власний код для кількох платформ.

Таким чином, Flutter може легко спілкуватися з платформою, не потребуючи мосту JavaScript, який передбачає перемикання контексту між

сферою JavaScript та рідною сферою. Як ви можете собі уявити, компіляція до рідного коду також збільшує час запуску програми.

Сьогодні Flutter - єдиний мобільний SDK, який пропонує реактивні перегляди без необхідності мосту JavaScript. Ось чому так багато розробників мобільних пристроїв випробовують це у своїх проектах.

Ось ще деякі переваги, які Flutter приносить розробці програмного забезпечення для мобільних пристроїв.

Однією з найцікавіших функцій Flutter є мова, якою він користується: Dart. Як і інші системи, які використовують реактивні перегляди, Flutter оновлює дерево перегляду для кожного нового кадру. Для цього він створює багато об'єктів, які можуть жити не більше одного кадру. Dart використовує збір сміття поколінь, який виявився дуже ефективним для систем такого типу.

Крім того, у Dart є компілятор "струшування дерева", який містить лише потрібний вам код у вашому додатку. Навіть якщо вам потрібен лише один - два віджети, ви можете вільно користуватися його великою бібліотекою віджетів.

Нарешті, Dart поставляється зі сховищем програмних пакетів для розширення можливостей програм. Наприклад, він пропонує кілька пакетів, які допомагають отримати доступ до Firebase, щоб розробники могли створювати без серверні програми. Інший пакет дозволяє отримати доступ до сховища даних Redux або полегшує доступ до послуг платформи та обладнання, як -от камери.

Переваги Flutter:

- Це економить ваш час та гроші Flutter-це крос-платформенний інструмент розробки. Це означає, що розробники програмного забезпечення можуть використовувати однакову кодову базу для створення додатків для iOS та Android. Міжплатформна розробка-найкращий метод економії часу та ресурсів протягом усього процесу розробки.Продуктивність
- Flutter пропонує чудову продуктивність з двох причин.

- По -перше, використовується Dart, який компілюється у рідний код.
- По -друге, у Flutter є свої віджети, тому немає необхідності отримувати
- доступ до OEM. В результаті між додатком і платформою стає менше спілкування. Ці дві функції Flutter забезпечують швидкий час запуску додатків і загалом менше проблем із продуктивністю.
- Швидка розробка завдяки гарячому перезавантаженню

Flutter набирає популярність серед розробників мобільних пристроїв через гаряче перезавантаження. Гаряче перезавантаження дозволяє миттєво переглядати зміни, застосовані до коду, на емуляторах, тренажерах та устаткуванні. Змінений код перезавантажується менш ніж за секунду. Весь цей час програма працює, і розробникам не потрібно витрачати час на її перезапуск.

Це спрощує створення інтерфейсів користувача, додавання нових функцій та виправлення помилок. Якщо у програмі виникає помилка, зазвичай її можна виправити, а потім продовжувати користуватися програмою так, ніби її ніколи не було. Навіть якщо ви змушені повністю перезавантажити додаток, ви можете бути впевнені, що його завершено в найкоротші терміни, прискорюючи процес розробки.

Сумісність

Ще однією перевагою Flutter є той факт, що він поставляється з власними віджетами, що призводить до меншої кількості проблем із сумісністю. Розробники побачать менше проблем у різних версіях ОС і можуть витрачати менше часу на тестування програми на старих версіях ОС. Крім того, ви можете бути впевнені, що ваш додаток працюватиме з майбутніми версіями ОС.

Відкрите джерело

Flutter-це технологія з відкритим кодом, оточена активною спільнотою розробників, які надають підтримку, допомагають у великій документації інструменту та розробляють корисні ресурси. І Dart, і Flutter є безкоштовними у використанні.

Flutter - одна з найбільш інноваційних мобільних технологій на ринку зараз. Переваги, які він приносить командам розробників, роблять його перспективним кандидатом на вибір мобільних технологій у найближчому майбутньому.

5.2 Огляд структури проекту

Виходячи з усіх вимог та обмежень, які були виявлені на етапі проектування системи, а також враховуючи архітектуру проекту, структура готового проекту має такий вигляд (рисунок 5.1)

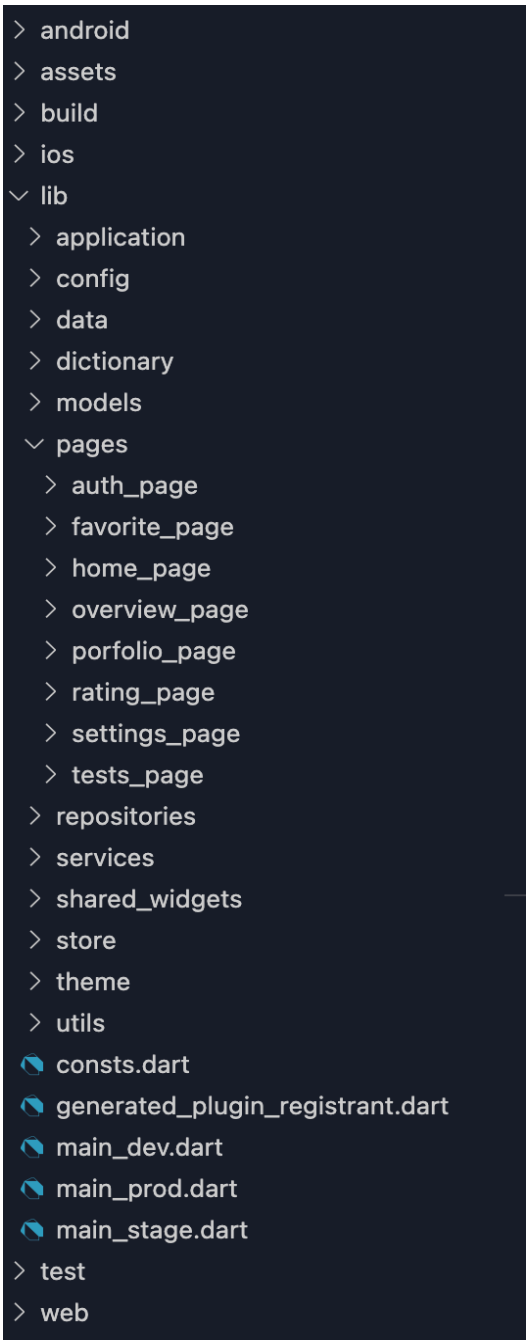

Рисунок 5.1 - Огляд структури проєкту

Де Config містить в собі всю налаштування та роутинг проекту, приклад роутінгу:

```
@MaterialAutoRouter(
 routes: <AutoRoute>[
  AutoRoute(
    page: SplashScreen,
    initial: true,
  ),
  AutoRoute(
    page: HomePage,
    path: 'Home page',
  ),
  AutoRoute(

    page: AuthPage,
    path: AppRoutes.authPage,
  ),
  AutoRoute(
    page: TestsPage,
    path: 'Tests page',
  ),
  AutoRoute(
    page: FavoritePage,
    path: 'Favorite page',
  ),
  AutoRoute(
    page: SettingsPage,
    path: 'Settings page',
  ),
  AutoRoute(
```

```
    page: OverviewPage,
    path: AppRoutes.overviewPage,
  ),
  AutoRoute(
    page: PortfolioPage,
    path: AppRoutes.portfolioPage,
  ),
 ],
)
class $AppRouter {}
```

Data - містить в собі всі запити до серверу шо включає в себе:

- Api - де знаходяться всі точки запитів.
- Dependency - находяться всі обробки перед запитом, такі як обробка успішного запита чи генерацію Headers і т.д.
- Repositories/Services - це вже йде передача даних до запиту та його обробка вже в моделі.

Models - зберігає в собі всю моделі классів.

Dictionary - зберігає текста усіх сторінок, як приклад:

```
class Language {
 final String name;
 final ChatLanguage chatLanguage;
 final TermsLanguage termsLanguage;
 final HomeLanguage popupsLanguage;
 final AuthLanguage authLanguage;
 final TermsLanguage drawerLanguage;
 final PortfolioLanguage portfolioLanguage;
 final OverviewLanguage overviewLanguage;
 final RatingLanguage ratingLanguage;
 final SettingsLanguage settingsLanguage;
```

```
 final BottomBarLanguage bottomBarLanguage;
 const Language({
   required this.name,
   required this.chatLanguage,
   required this.termsLanguage,
   required this.popupsLanguage,
   required this.authLanguage,
   required this.drawerLanguage,
   required this.portfolioLanguage,
   required this.overviewLanguage,
   required this.ratingLanguage,
   required this.settingsLanguage,
   required this.bottomBarLanguage,
 });
}
```

Pages - содержит віджети всіх сторінок проекту.

Repositories - содержит в себе себе всі функції які мають зв'язок напряму с девайсом, як приклад можна побачити реалізацію зберігання токена на память девайса:

```
@lazySingleton
class TokenStorage {
 final IStorage _storage;
 const TokenStorage(this._storage);
 static const String _tokenKey = Persistent.TOKEN;
 Logger get logger => Logger('[$runtimeType]');
 Future<void> saveToken(String tokenData) async {
   logger.info('SAVING TOKEN TO STORAGE ${tokenData.toString()}');
   await _storage.save(_tokenKey, tokenData);
 }
```

```
 Future<String?> getToken() async {
   try {
     return _storage.take<String>(_tokenKey);
   } catch (e) {
     logger.warning('<getToken> => error: $e');
     return null;
   }
 }

 Future<void> clearToken() async {
   try {
     await _storage.delete(_tokenKey);
     logger.info('Token was removed from storage');
   } catch (e) {
     logger.warning('<getToken> => error: $e');
   }
 }
}
```

Theme - містить в собі всі стилі, кольори, картинки, тіні та градієнти які використовуються в проекті, як приклад реалізація кольорів та текстів:

```
class AppColors {
 static const Color transparent = Color(0x00000000);
 static const Color aliceBlue = Color(0xFFF2F6FE);
 static const Color aliceBlue2 = Color(0xFFf2f6fe);
 static const Color topaz = Color(0xFF1ACCAE);
 static const Color pinkyLight = Color(0xFFE0A8EA);
 static const Color mainPurple = Color(0xFF6B39B6);
```

```
 static const Color mainPink = Color(0xFFA44DB3);
 static const Color thunder = Color(0xFF303030);
 static const Color darkSlateBlue = Color(0xFF1d3557);
 static const Color chathamsBlue = Color(0xFF234b70);
 static const Color greenWhite = Color(0xFFe8e8e8);
 static const Color lipstick = Color(0xFFE63946);
 static const Color jasmine = Color(0xFFFFDC7E);
 static const Color lightBlue = Color(0xFF62aef5);
 static const Color veryLightPink = Color(0xFFfef2f2);
 static const Color shadowRed = Color(0xFFed8882);
 static const Color black = Color(0xFF1D1626);
 static const Color white = Color(0xFFFFFFFF);
 static const Color whiteTwo = Color(0xFFd8d8d8);
 static const Color whiteThree = Color(0xFFf0f0f0);
 static const Color background = Color(0xFFF2F2F2);
 static const Color grey = Color(0xFF8E8B93);
 static const Color pinkishGrey = Color(0xFFD0D0D0);
 static const Color greyTwo = Color(0xFFD2D0D4);
 static const Color darkGrey = Color(0xFF3C3C43);
 static const Color lightGrey = Color(0xFF828282);
 static const Color pinkSwan = Color(0xFFb2b2b2);
 static const Color borderColor = Color(0xFFf0f0f0);
 static const Color success = Color(0xFF55CF70);
 static const Color error = Color(0xFFF44336);
 static const Color stars = Color(0xFFF0CA90);
}
class AppTextStyles {
 static const TextStyle s27fw700 = TextStyle(
   fontFamily: 'RagSans',
   color: AppColors.aliceBlue,
```

```
  fontSize: 27.0,
  height: 1.0,
  fontWeight: FontWeight.w500,
);
static const TextStyle s12fw500 = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.darkSlateBlue,
  fontSize: 12.0,
  fontWeight: FontWeight.w500,
);

static const TextStyle s15fw300 = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.darkSlateBlue,
  fontSize: 15.0,
  height: 22.0 / 15.0,
  fontWeight: FontWeight.w300,
);

static const TextStyle s11fw300 = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.darkSlateBlue,
  fontSize: 11.0,
  fontWeight: FontWeight.w300,
);

static const TextStyle s22fw500cWhite = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.white,
  fontSize: 22.0,
```

```
  fontWeight: FontWeight.w500,
);

static const TextStyle s11fw300Lipstick = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.lipstick,
  fontSize: 11.0,
  fontWeight: FontWeight.w300,
);

static const TextStyle s11fw300Topaz = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.topaz,
  fontSize: 11.0,
  fontWeight: FontWeight.w300,
);

static const TextStyle s11fw300Grey = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.grey,
  fontSize: 11.0,
  fontWeight: FontWeight.w300,
);

static const TextStyle s15fw300PinkishGrey = TextStyle(
  fontFamily: 'RagSans',
  color: AppColors.pinkishGrey,
  fontSize: 15.0,
  height: 20.0 / 15.0,
  fontWeight: FontWeight.w300,
```

```
 );

 static const TextStyle s15fw300cWhite = TextStyle(
   fontFamily: 'RagSans',
   color: AppColors.white,
   fontSize: 15.0,
   fontWeight: FontWeight.w300,
 );
}
```

Store - має реалізацію архітектурного стилю Redux і зберігає в собі головний state:

```
/// Class [AppState], is the main [state] application.
/// It keeps 3, smaller states.
/// Namely, [dialogState], [storageState], [loaderState].
/// First [dialogState], this variable stores the state of dialogs, it is used to call various dialogs.
/// Second [storageState], the primary state, stores all information from all states.
/// The third [loaderState] is required to loading.
class AppState {
 final DialogState dialogState;
 final LoaderState loaderState;
 final LanguageState languageState;
 final HomeState homeState;
 final AuthState authState;
 final TestsState testsState;
 final SettingState settingState;
 final RatingState ratingState;
 final ProfileState profileState;
```

```
AppState({
  required this.dialogState,
  required this.loaderState,
  required this.languageState,
  required this.homeState,
  required this.authState,
  required this.testsState,
  required this.settingState,
  required this.ratingState,
  required this.profileState,
});

///All states are initialized in the [initial] function.
factory AppState.initial() {
  return AppState(
    languageState: LanguageState.initial(),
    dialogState: DialogState.initial(),
    loaderState: LoaderState.initial(),
    homeState: LoaderState.initial(),
    authState: LoaderState.initial(),
    testsState: LoaderState.initial(),
    settingState: LoaderState.initial(),
    profileState: LoaderState.initial(),
    ratingState: LoaderState.initial(),
  );
}

///The [getReducer] function is the main Reducer in which you call Reducer, all other states.
static AppState getReducer(AppState state, dynamic action) {
```

```
  if (action is ClearStateAction) {
    return AppState.initial();
  }
  return AppState(
    dialogState: dialogReducer(state.dialogState, action),
    loaderState: loaderReducer(state.loaderState, action),
    languageState: languageReducer(state.languageState, action),
     homeState: homeeReducer(state.languageState, action),
    authState: authReducer(state.languageState, action),
    testsState: testsReducer(state.languageState, action),
    settingState: settingReducer(state.languageState, action),
    profileState: profileReducer(state.languageState, action),
    ratingState: ratingReducer(state.languageState, action),
  );
 }

 ///In [getAppEpic], call the main epic.
 static final getAppEpic = combineEpics<AppState>([
  initializationEpic,
  languageEpic,
  homeEpic,
  authEpic,
  testsEpic,
  settingEpic,
  profileEpic,
  ratingEpic,
 ]);
}
```

6. ТЕСТУВАННЯ

Під час розробки було виконано 3 методи тестування, за допомогою пакета Flutter test, який містить у собі:

- WidgetTester, який дозволяє створювати віджети та взаємодіяти з ними у тестовому середовищі;
- Функція testWidgets(), яка автоматично створює новий WidgetTester для кожного тестового прикладу та використовується замість звичайної функції test();
- Класи Finder дозволяють шукати віджети у тестовому середовищі;
- Константи Matcher для конкретних віджетів допомагають перевірити, чи виявляє Finder віджет або кілька віджетів у тестовому середовищі.

Було виконано 3 види тестів:

1. Модульне тестування – це тип тестування програмного забезпечення, у якому тестуються окремі модулі чи компоненти програмного забезпечення. Його мета полягає в тому, щоб перевірити, що кожна одиниця програмного коду працює належним чином. Цей вид тестування виконується розробниками на етапі кодування програми. Модульні тести ізолюють частину коду та перевіряють його працездатність. Одиницею для вимірювання може бути окрема функція, метод, процедура, модуль чи об'єкт.
2. Виджет тесты - тестируют один виджет и направлены на то, чтобы убедиться что интерфейс выглядит и взаимодействует с пользователем так, как было запланировано. Тестирование виджета включает несколько классов и требует тестовой среды, которая предоставляет соответствующий контекст жизненного цикла виджета.
3. Інтеграційне тестування – це тип тестування, коли програмні модулі об'єднуються логічно і тестуються як група. Як правило, програмний продукт складається з кількох програмних модулів, написаних різними

програмістами. Метою нашого тестування є виявлення багів при взаємодії між цими програмними модулями та в першу чергу спрямований на перевірку обміну даними між цими самими модулями. Саме тому воно також називається "I&T" (інтеграція та тестування), "тестування рядків" і іноді "тестування потоків".

6.1 Тестування сервісів

Для тестування сервісів було обрано модульне тестування, як приклад на рисунку 6.1 ви можете бачити тестування сервісу на валідацію пошти, так як логін юзера це пошта.

```
Run | Debug
test('Email validation:', () async {
  final String? emailValidation = ValidationService.emailValidation('test@gmail.com');

  print('VALIDATION: $emailValidation');

  expect(emailValidation == null, true);
});
```

Рисунок 6.1 - Тестування сервісу верифікації пошти

На рисунку 6.2 представлено результат тестування.

```
SET UP DEPENDED CONFIG
COMPLETION DEPENDED CONFIG
Email validation
VALIDATION: null
✓ Service test get Email validation:
Exited
```

Рисунок 6.2 - Результат тесту

6.2 Тестування на пошук

Для тесту пошуку був виконаний Widget тест на рисунку 6.3, так як нам потрібно знайти елемент на сторінці та внести в нього текст, та натиснути на кнопку пошуку.

```dart
testWidgets('Search test ', (WidgetTester tester) async {
  await tester.pumpWidget(HomePage());

  final textfield = find.byType(TextFormField);
  final button = find.byType(InkWell);

  await tester.enterText(textfield, 'test');
  await tester.tap(button);

  print('Test is Done');
});
```

Рисунок 6.3 - Тест на пошук

На рисунку 6.4 представлено результат тестування.

```
SET UP DEPENDED CONFIG
COMPLETION DEPENDED CONFIG
Search test
Test is Done
✓ Service test get Search:
Exited
```

Рисунок 6.4 - Результат тесту

6.3 Тестування авторизації

Для тестування авторизації був обраний інтеграційний тест на рисунку 6.5, в якому буи реалізован сценарій авторизації користувача, де при запуску тесту вводились дані в поля, була перевірка на валідацію та в кінці натискається кнопка переходу до головної сторінки, так як нам потрібно побачити як система себе веде зі взаємодією сервісів та іншими віджетами

```dart
testWidgets('Authorization test', (WidgetTester tester) async {
  auth_page.AuthPage();

  await tester.pumpWidget(
    MaterialApp.router(
      routerDelegate: router.delegate(),
      routeInformationParser: router.defaultRouteParser(),
      builder: (context, child) {
        return StoreProvider<AppState>(
          store: store,
          child: AuthPage(),
        ); // StoreProvider
      },
    ), // MaterialApp.router
  );

  final authFiled = find.byKey(Key(AuthPageKeys.authTextField));
  await tester.pumpAndSettle();

  await Future.delayed(Duration(seconds: 1), () {});

  await tester.enterText(authFiled, 'email@gmail.com');
  await tester.pumpAndSettle();

  await Future.delayed(Duration(seconds: 1), () {});

  await tester.tap(find.byKey(Key(AuthPageKeys.authButton)));
  await tester.pumpAndSettle();

  await tester.tap(find.byKey(Key(AuthPageKeys.authButton)));
  await tester.pumpAndSettle();

  await Future.delayed(Duration(seconds: 12), () {});

  await tester.tap(find.byKey(Key(AuthPageKeys.authButton)));
  await tester.pumpAndSettle();

});
```

Рисунок 6.5 - Тестування авторизації

ВИСНОВОК

Кваліфікаційна робота була присвячена розробці програмного забезпечення для гейміфікація процесу навчання співробітників компаній, яка має назву «Соціальна мережа для гейміфікація процесу навчання співробітників компаній». Метою кваліфікаційної роботи є розробка простого та зрозумілого інтерфейсу для всіх груп людей, також щоб користувач мав можливість гейміфікувати процес вивчення нового матеріалу чи теми пов'язаної з роботою. Цю мету дипломної роботи можно вважати досягненою так як при користуванні системою більш ніж 20 особами їх опитування показало, що 97% користувачів сподобався інтерфейс, та близько 93% сподобалось, методика навчання за допомогою гейміфікації.

Робота почалася зі специфікації вимог до програмного забезпечення. З самого початку був проведений аналіз предметної області та пошук і вивчення існуючих програмних рішень. Далі було визначено основний функціонал системи у вигляді діаграми варіантів використання та їх опис.

Планування було створена діаграма програмних класів, діаграми послідовності, прототипування зовнішнього вигляду системи та її реалізація.

Останнім етапом розробки стало її тестування, де були складені тестові випадки для одного варіантів використання.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dobrynin, Ye. V., Boltenkov, V. O. & Maksymov, M. V. “Information Technology for Automated Assessment of the Artillery Barrels Wear Based on SVM Classifier”. *Applied Aspects of Information Technology. Publ. Nauka i Tekhnika*. Odessa: Ukraine. 2020; Vol. 3 No. 3: 117–132. DOI: https://doi.org/10.15276/aait.03.2020.1
2. Redux URL: https://pub.dev/documentation/redux/latest/(дата зверненя 15.10.2021)
3. Sivokobylenko V. F., Nikiforov A. P. & Zhuravlov I. V. “ Detecting development scenarios of dynamic events in electric power networks smart-grid. Part 1 “Method”. *Applied Aspects of Information Technology. Publ. Nauka i Tekhnika*. Odessa: Ukraine. 2021; Vol. 4 No. 3: 219– 234. DOI: https://doi.org/10.15276/aait.03.2021.1
4. Kulakov, K., Zavyalova, Y. Navigation infrastructure for people with disabilities. 20th Conference of open Innovations Association (FRUCT), St. Petersburg, April 03-07, 2017. St. Petersburg, Russia, 2017. P. 78-84.
5. В. А. Крісілов, К. О. Писаренко, і В. Н. Хуї, «Метод динамічного формування контенту в умовах обмежених ресурсів», вид. 2019, вип. 2, вип. 2, с. 89-104. URL: https://aait.opu.ua (дата звернення 18.10.2021).
6. Фреймворк Flutter - [Електронний ресурс] – Режим доступу: - https://flutter.dev/docs
7. Мова програмування Dart- [Електронний ресурс] – Режим доступу: https://dart.dev/overview
8. Kowtko, M. Using assistive technologies to improve lives of older adults and people with disabilities. 2012 IEEE Long Island Systems, Applications and Technology Conference (LISAT), Farmingdale, May 04, 2012. Farmingdale, NY, USA, 2012. P. 11-35.
9. Abul, K. Internet accessible remote experimentation: setting the right course of action. International Journal of Online Engineering. 2010. Vol. 6. No 3. P. 4-12.

10. Brian Liang, Mobile Design and Development O'Reilly Media, Inc, USA,18 Sep 2009,Sebastopol, United States,
11. Zigurd Mednieks Programming Android, O'Reilly Media, Inc, USA, 06 Nov 2012
12. Robert C. Martin Series Clean Architecture : A Craftsman's Guide to Software Structure and Design, Pearson Education, 20 Sep 2017