Державний торговельно-економічний університет

Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Програмний модуль інформаційної системи підприємства торгівлі вживаними електронними гаджетами»

Студента 4 курсу, 6 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» — Сяська Дмитра Васильовича

_____
підпис студента

Науковий керівник PhD, доцент кафедри інженерії програмного забезпечення та кібербезпеки — Десятко Альона Миколаївна

_____
підпис керівника

Гарант освітньої програми кандидат технічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки — Рзаєва Світлана Леонідівна

_____
підпис гаранта

КИЇВ – 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Сяську Дмитру Васильовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Програмний модуль інформаційної системи підприємства торгівлі вживаними електронними гаджетами»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту - здобути навички та знання із розробки програмних додатків мовою програмування Dart за допомогою комплексу розробки Flutter та дослідити інформаційну систему підприємств торгівлі вживаними електронними гаджетами.

Об'єкт дослідження - управління інформаційною системою підприємства з торгівлі вживаних електронних гаджетів

Предмет дослідження - програмне забезпечення з управління інформаційною системою підприємства з торгівлі вживаних електронних гаджетів.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ТЕХНІЧНОГО ЗАВДАННЯ

1.1. Загальні теоретичні відомості проблем розробки сучасності і способи їх вирішення

1.2. Опис предметної області та мета створення системи

1.3. Технічне завдання

1.4. Висновки до розділу 1

РОЗДІЛ 2. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Вибір середовища розробки, мови програмування та відповідного інструментарію

2.2. Побудова архітектури розроблюваного програмного забезпечення

2.3. Побудова графічного інтерфейсу та макету додатку

2.4. Висновки до розділу 2

РОЗДІЛ 3. РОЗРОБКА КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Розробка Back-End частини додатку

3.2. Розробка Front-End частини додатку

3.3. Висновки до розділу 3

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		За планом	Фактично
1	2	3	4
1.	Вибір теми випускного кваліфікаційного проєкту	21.09.2022	21.09.2022
2.	Розробка та затвердження завдання на проєкту	14.11.2022	14.11.2022
3.	Вступ та перелік літературних джерел	23.12.2022	23.12.2022
4.	Розділ 1. Аналіз предметної області та постановка технічного завдання	27.01.2023	27.01.2023
5.	Розділ 2. Проєктування програмного забезпечення	03.03.2023	03.03.2023
6.	Розділ 3. Розробка коду програмного забезпечення	*14.04.2023*	*14.04.2023*
7.	Висновки та пропозиції	*28.04.2023*	*28.04.2023*
8.	Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)	*17.05.2023*	*17.05.2023*
9.	Підготовка автореферату та презентації доповіді	*26.05.2023*	*26.05.2023*
10.	Попередній захист випускного кваліфікаційного проєкту	*29.05.2023 – 02.06.2023*	
11.	Зовнішнє рецензування випускного кваліфікаційного проєкту	*05.06.2023*	*05.06.2023*
12.	Здача прошитого випускного кваліфікаційного проєкту на кафедру	*05.06.2023*	*05.06.2023*
13.	Публічний захист випускного кваліфікаційного проєкту	*16.06.2023*	*16.06.2023*

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту

Десятко А.М

(прізвище, ініціали, підпис)

9. Гарант освітньої програми Рзаєва С.Л.

(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Сясько Д.В.

(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента _____Сясько Д.В._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Рзаєва С.Л._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена розробці програмного модуля інформаційної системи підприємства торгівлі вживаними електронними гаджетами. Основне завдання роботи - створити програмне забезпечення, що дозволяє ефективне та безпечне управління конфіденційними даними підприємства торгівлі та дослідити інформаційну систему даних підприємств торгівлі вживаними електронними гаджетами.

В результаті порівняльного аналізу визначено головні проблеми, з якими стикаються підприємці даного напрямку. Визначено сучасні інструменти, технології та методи розробки програмного забезпечення та переваги вибраних технологій.

Розробка програмного модуля виконана у середовищі Flutter і мовою програмування Dart, що забезпечує зручне та швидке користування програмним додатком за допомогою поєднання із архітектурним рішенням відділення логіки від інтерфейсу, що забезпечує надійність даних у використанні і швидкість у користуванні..

Готове програмне забезпечення «EGUMobStore» успішно функціонує на платформах Android і доступно будь-якому користувачеві.

Ключові слова: Flutter, Dart, об'єктно-орієнтоване програмування, Bloc, патерни проектування, UML, база даних, SQL, фреймворк, інтерфейс, user experience.

ABSTRACT

In accordance with the purpose of the research, the work is devoted to the development of the software module of the information system of the enterprise trading in used electronic gadgets.

The main task of the work is to create software that allows effective and safe management of confidential data of a trading enterprise and to investigate the data information system of enterprises trading in used electronic gadgets.

As a result of the comparative analysis, the main problems faced by entrepreneurs in this area have been determined. State-of-the-art software development tools, technologies and methods and advantages of selected technologies are identified.

The development of the software module was carried out in the Flutter environment and the Dart programming language, which ensures convenient and fast use of the software application through a combination with the architectural solution of separating the logic from the interface, which ensures data reliability in use and speed in use.

Ready-made software "EGUMobStore" works successfully on Android platforms and is available to any user.

Keywords: Flutter, Dart, object-oriented programming, Bloc, design patterns, UML, database, SQL, framework, interface, user experience.

ЗМІСТ

					ДТЕУ 121 06-24.БР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф		Криворучко О. В		23.12.22	*Програмний модуль інформаційної системи підприємства торгівлі вживаними електронними гаджетами*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Десятко А.М.		23.12.22		3	2	50
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій 4 курс 6 група*		
Розробив		Сясько Д.В		23.12.22	*Зміст*			

ВСТУП

Актуальність. З кожним роком відбувається значний приріст появи нових комерційних підприємств із продажу товарів, в тому числі і вживаних електронних гаджетів. Як правило, у даних підприємств відсутня можливість забезпечити стійку та надійну роботу із зберігання, обліку та аналізу інформаційної системи підприємства із фінансових причин. Саме тому рішення у вигляді мобільного програмного забезпечення стає важливим і надійним варіантом для таких випадків.

Роль програмного забезпечення у нашому житті зростає із кожним днем. На даний момент існує незліченна кількість варіантів використання комп'ютерних технологій майже на кожному людському кроці: на вулицях, у магазинах, у школах та університетах. Людина може користуватися комп'ютером або іншим гаджетом на робочому місці, на відпочинку або ж роблячи будь-які інші потрібні їй справи. Все це програмне забезпечення призначене для полегшення роботи, оптимізації як робочого процесу, так і життя людей. Програмні технології оточують людину на кожному її кроці і продовжує стрімко розвиватися із кожним людським кроком: будуються надзвичайно великі компанії різних типів з розробки програмного забезпечення і різних додатків на будь-які пристрої і досягають свої цілі з розробки успішно. Швидкість розростання та розповсюдження технологій практично неможливо визначити.

Роль окремого програмного додатку може варіюватись у безлічі варіантів: відпочинок, оптимізація певних процесів, в тому числі і робочих,

					ДТЕУ 121 06-24.БР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф		Криворучко О. В		23.12.22	*Програмний модуль інформаційної системи підприємства торгівлі вживаними електронними гаджетами*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Десятко А.М.		23.12.22		В	3	50
Гарант		Рзаєва С.Л.		23.12.22		*Факультет інформаційних технологій 4 курс 6 група*		
Розробив		Сясько Д.В		23.12.22	*Вступ*			

незалежно від завдання роботи, у світі існують додатки для будь-яких дій людини у цьому світі: в будівництві, в торговому бізнесі, тощо.

Переваги швидкого розповсюдження програмного забезпечення і впровадження його у кожен людський крок безумовно незліченні, їх можна перечислювати надто довго: за лічені секунди люди можуть виконати роботу, яку б раніше взагалі людство вважало б неможливою для виконання або займала б десятки років. На даний момент розробка програмного забезпечення, в незалежності від того, на яку платформу воно розробляється, якою мовою іде розробка, за допомогою яких інструментів чи сторонніх ресурсів – це чітко вибудована система великими компаніями та окремі прийняті норми та стандарти для побудови додатків та їх архітектури.

Суспільство може інколи не замислюватися про те, як побудована програма, якою люди кожного дня користуються, але інколи помічає недоліки у роботі певних додатків. Разом із вище переліченими перевагами ПЗ, існує також дуже багато недоліків та проблем у процесі розробки програмних додатків. Кожна проблема, яка на перший погляд може здатися маленькою та незначною може перерости в фактичну катастрофу для компанії чи додатку, якщо вчасно її не виправити.

Саме тому, на сьогодні, перед компаніями, які замовляють або розробляють програмне забезпечення стоїть велике завдання – організувати правильний та скоординований процес розробки додатку і його функціонал. Запорука, за якої бізнес у 21 столітті може існувати довго, в тому числі і прибутково, як основна рушійна сила бізнесу та його розитку – це програмне забезпечення, яке працює та оптимізовує процес роботи компанії і не має проблем, що не призводить до збитків компанії. Можна приблизно уявити збитки у величезній компанії з торгівлі, яка на деяких час матиме не функціонуючий веб-ресурс. Проблеми, які постають перед розробниками кожного дня виміряються сотнями, тому в цьому і є найбільша важливість задачі, не нехтувати навіть найменшими проблемами у розробці і продумувати

кожен свій крок у розробці наперед, адже в більшості випадків, проблема, яка була проігнорована переростає у величезну проблему, яку не завжди вдається виправити, через що додаток або ж навіть компанія вибуває із ринку.

Головне завдання розробників програмного забезпечення у світі зараз - це вибудовувати абсолютно працюючу систему розробки, так і подальшу систему вдосконалення додатку і відсутність поломок у існуючому додатку, щоб забезпечити бізнесу швидку і прибуткову роботу.

Метою дослідження є здобути навички та знання із розробки програмних додатків мовою програмування Dart за допомогою комплексу розробки Flutter та дослідити інформаційну систему підприємств торгівлі вживаними електронними гаджетами.

У відповідності з метою дослідження поставлено завдання:

- розробити програмне забезпечення для мобільних пристроїв для керування інформаційною системою підприємства з торгівлі вживаних електронних гаджетів мовою програмування Dart за допомогою комплексу розробки Flutter;
- дослідити інформаційну систему підприємств торгівлі вживаними електронними гаджетами.

Об'єктом дослідження є управління інформаційною системою підприємства з торгівлі вживаних електронних гаджетів.

Предметом дослідження є програмне забезпечення з управління інформаційною системою підприємства з торгівлі вживаних електронних гаджетів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ТЕХНІЧНОГО ЗАВДАННЯ

1.1 Загальні теоретичні відомості проблем розробки сучасності і способи їх вирішення

Проблеми удосконалення, розвитку, структурності та здатності до модифікації коду існують велику кількість часу, з появою мов програмування, на яких будувались обширні за розміром проекти. Було вигадано та випробувано незліченну кількість методів для знаходження варіанту, який допоможе вирішити всі вище наведені проблеми.

Об’єктно-орієнтоване програмування (ООП) – це одна із найрозповсюдженіших тем у світі розробки програмного забезпечення. Ця парадигма дозволяє розробникам створювати більш складні системи на різних мовах програмування, а саме найбільш розповсюджених, таких як: Java, C#, Ruby, Python, TypeScript, PHP та інші. Існують основні принципи використання об’єктно-орієнтованого підходу розробки, використовуючи класи та об’єкти [1].

Об’єктно-орієнтоване програмування - це модель комп’ютерного програмування, яка організовує дизайн програмного забезпечення навколо даних або об’єктів, а не функцій і логіки. Об’єкт можна визначити, як тіло, яке має спеціальні дані, тобто наділене спеціально визначеними унікальними атрибутами та поведінкою [2].

Схематично логіку цього методу можна зобразити з прикладу із життя – на основі людини, її поведінки, рис та властивостей (рис. 1.1)

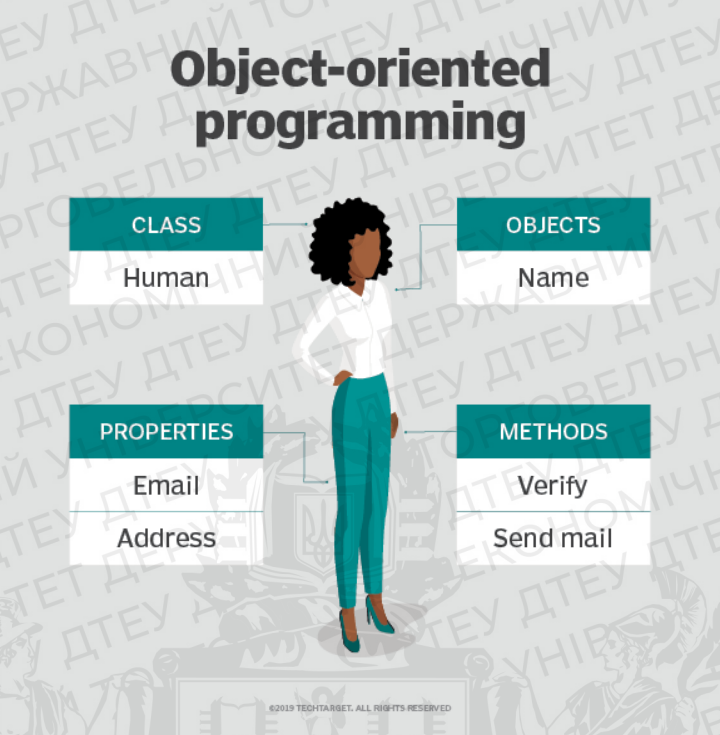


Рисунок 1.1. Приклад об'єктно-орієнтованої структури

Джерело: побудовано на основі джерела [2]

У даного підходу є декілька основних принципів та правил використання, які потрібно дотримуватися для підтримки грамотної та правильної структурності та архітектури програмного забезпечення, що розроблюється. Адже використання самої методики об'єктно-орієнтованого програмування в будь-яких цілях та задачах, котрі цього не потребують, не завжди є правильним рішенням для розробки і найчастіше призводить до великих проблем додатку.

ООП тримає свою увагу на об'єктах, якими розробники планують керувати, але аж ніяк не на функціоналі для управління цими об'єктами. Цей підхід до програмування добре підходить для великих, складних програм, які активно оновлюються або обслуговуються [2].

Існує чотири принципи об'єктно-орієнтованого програмування побудови додатків: це поліформізм, успадковування, наслідування та абстракція [1].

Поліморфізм - це здатність об’єкта набувати різних форм або бути поліморфним. Зазвичай це досягається за допомогою успадкування, коли дочірній клас може перевизначати поведінку членів, успадкованих від батьківського класу [1].

Успадковування означає, що класи можуть повторно використовувати код з інших класів. Між об’єктами можна призначати зв’язки та підкласи, що дозволяє розробникам повторно використовувати загальну логіку, зберігаючи при цьому унікальну ієрархію [2].

Успадковування відкриває шлях до використання поліморфізму, а також дозволяє відокремити та додати деякі риси, властивості чи поведінку до об’єктів окремого типу, які мають ці риси унікальними, наприклад: телефони та аудіо системи є електронними пристроями та мають однакові властивості, такі як розмір, колір або ж ціну, проте з телефону можна виконувати стільникові дзвінки, чого не можна робити з аудіо систем. Це дозволяє тримати структуру проекту в легко модифікованій та керованій формі (рис. 1.2)

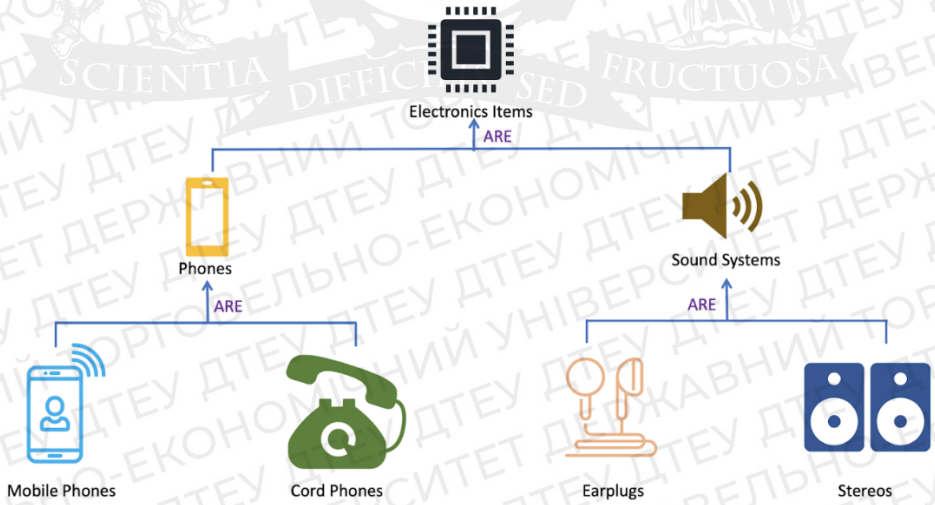


Рисунок 1.2. Приклад використання успадковування

Джерело: побудовано на основі джерела [4]

Використання поліморфізму особливо потребує додаткової уваги розробника, адже при надлишковому і неправильному використанні даного принципу, він може призвести до проблем взаємодії внутрішньої логіки програмного забезпечення між собою. Також і властивість успадковування змушує розробника проводити більш ретельний аналіз даних, що використовуються, щоб запобігти появі проблеми сильної зв'язаності ієрархії, яка при найменших змінах у структурі додатку призводить до поломки.

Інкапсуляція містить інформацію в об'єкті, відкриваючи лише вибрану інформацію, а абстракція використовується для доступу до об'єкта, розкриваються лише публічні методи високого рівня. Обидва використовують модифікатори доступу [3].

Проблеми, що постійно з'являються як у розробників-одинаків, так і у великих компаніях, які керуються десятками розробників одночасно – існують і досі. Проте за роки активної розробки сформувалися певні закономірності між появою та наслідками цих проблем, не обійшлося також без формування способу вирішення цих проблем. Перед розробкою додатку найкращим варіантом прийнято розроблювати план проекту і визначати основний функціонал програмного забезпечення, яке в майбутньому має використовуватися користувачами. Відштовхуючись від цього функціоналу компанії відразу складають план використання та впровадження у власну систему патернів або шаблонів проектування. Відомі розробники, котрі внесли свій великий внесок у інженерію програмного забезпечення особливо відзначились саме відокремленням цих шаблонів.

Патерн проектування (від англ. Design Pattern) — це загально прийняте рішення найпоширеніших проблем серед розробників ПЗ при будуванні архітектури програмного забезпечення. Шаблон, мається на увазі, як не готовий проект, який можна відразу перенести у власний код. Це опис або деталізована інструкція того, як вирішити проблему [9].

Скільки існує проблем – стільки існує і шаблонів їх вирішення. Патерни класифікують за типами вирішення спеціально визначеного кола завдань, існують такі типи:

- породжуючі, які включають в себе такі патерни, як «Абстрактна фабрика», «Будівельник», «Одинак» або «Singleton» та інші. Використовуються для керування об’єктами у коді;
- структурні, включають в себе «Міст», «Фасад», «Декоратор», «Адаптер» та інші. Загалом використовуються для вирішення проблем розділення коду та формування структури об'єктів;
- поведінкові, включають в себе багато шаблонів, найбільш розповсюджені це «Ітератор», «Медіатор», «Стратегія», «Шаблонний метод», загалом використовуються для правильного керування алгоритмами [10].

Також існують і шаблони архітектури системи, такі як Model View-ViewModel (MVVM), Model-View-Controller (MVC) та Model-View-Presenter (MVP). Вони є найпопулярнішими і всі вони використовуються як фундамент побудови програмного забезпечення і уже від них відштовхнуються розробники, як додавати основні функції додатку. Основна задача даного методу це розбиття структури коду на частини, а також пов’язання цих частин для легкого подальшого удосконалення, підтримки коду та прив’язок даних. MVC – шаблон, який в більшості випадків використовується для побудови WEB-додатків, наприклад: сайтів. MVP – шаблон, який використовується для побудови мобільних додатків на нативних мовах програмування, таких як Swift або Kotlin. MVVM – найчастіше його використовують у побудові комп’ютерних додатків, інколи і будуються на цьому шаблоні і мобільні додатки. Схематично показано принцип взаємодії даної архітектури із користувачем (рис. 1.3) [11].

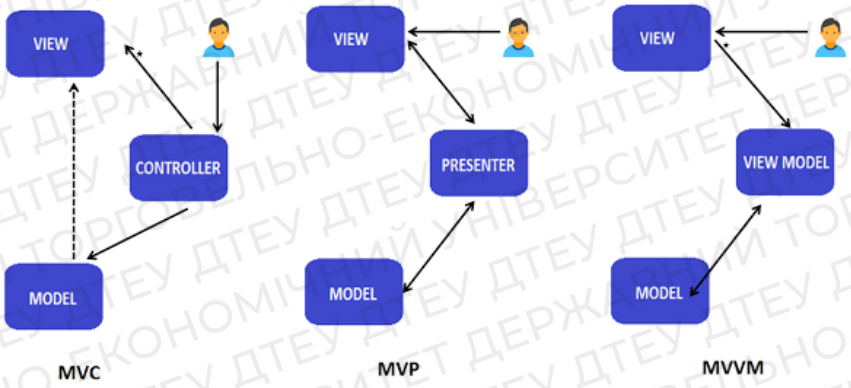


Рисунок 1.3. Схема взаємодії користувача із додатком

Джерело: побудовано на основі джерела [11]

Тому постає завдання із вирішення цих основних проблем найбільш розповсюдженими вище описаними методами.

1.2 Опис предметної області та мета створення системи

Розроблюваний програмний додаток матиме повну назву: Electronic Gadget Used Mobile Store. Скорочене найменування системи буде складатись із перших букв від слів повного найменування, а саме: EGUMobStore.

Нині, у 21 столітті, кількість нових створених малих підприємств продовжує збільшуватися з кожним днем. Напрямки цих підприємства бувають абсолютно різні, проте можна виділити, що поява малих бізнесів з продажу вживаних електронних пристроїв тепер не надто рідке явище, яке можна зустріти майже на кожній вулиці звичайного міста. Ці підприємства мають одну велику проблему, яка їх всі разом поєднує – це відсутність особистого програмного забезпечення, адже розробка програмного додатку виключно під окреме підприємство є надто дорогою можливістю, а малі підприємства такого не можуть дозволити з причин ведення бізнесу. Дана проблема вирішується кожним підприємством самостійно і як правило, використовуються ті методи, що є близькими до власників, навіть і ведення обліку певних даних у ручному форматі, це приводить інколи до втрати

даних, втрати часу або потенційних клієнтів, що аж ніяк не робить керування бізнесом зручним, швидким та прибутковим.

Додаток спрямований на користувачів, котрі мають у своїй власності малий або середній бізнес (наприклад фізичні особи-підприємці) у торгівлі вживаними електронними пристроями та потребують у програмному забезпеченні, яке дозволить легко керувати наявним товаром, що міститься у наявності, деталізувати свої товари, створювати, додавати, редагувати або ж видаляти, тобто керувати списком наявного товару базовими діями, а також містити дані про продані товари, суми прибутків та інші додаткові функції. За планом додаток має бути універсальним за функціями і повинен підходити багатьом підприємствам одночасно, адже він включає у себе базові функції, які потребують власники цих підприємств.

При цьому користування додатком не повинно потребувати вихід у мережу Інтернет, що істотно пришвидшить ведення та розвиток підприємства, а також зберігати цілісність даних користувача. Також, важливим фактором того, що робота по обліку наявних товарів буде проводитися значно швидше, це використання пристрою на мобільному додатку, який завжди є під рукою і вносити зміни у список наявних електронних пристроїв у підприємстві займає лічені хвилини, а інколи навіть і секунди.

Саме тому метою створення проекту є полегшення, прискорення та підвищена захищеність ведення обліку товару підприємства в цілому.

1.3 Технічне завдання

Загальні вимоги до додатку містять певні правила до розробки та побудови коду, а саме: розробка програмного забезпечення повинна проводитися за допомогою новітніх систем, технологій та інструментів, використовуючи системи підтримки, які здатні до постійного оновлення та покращення інструментів розробки та підтримки коду.

Вимоги до структури та функціонування системи: розроблювана система повинна розроблятися використовуючи одну із найпоширеніших структур, шаблонів чи патернів у розробці програмного забезпечення для мобільних пристроїв, задля подальшої легшої, вільної та кращої підтримки даного додатку. Система повинна працювати на пристроях різних версій операційних система та підтримувати відносно застарілі версії, що не піддаються оновленням від офіційних виробників та розробників оновлень для операційних систем, а також додаток повинен мати фундамент для того, щоб підлаштовуватися під оновлення даних операційних систем фізичних пристроїв нових версій.

Вимоги до діагностування системи: розроблюване програмне забезпечення повинно мати основні методи діагностування роботи системи та функціонал для знаходження проблем та швидке вирішення і оновлення додатку, а також підтримувати програмне тестування для легшого впровадження оновлень та максимального недопущення поломок системи при її удосконаленні, в тому числі і автоматичне визначення несправності на основі збоїв системи під час користування додатком потенційними користувачами.

Вимоги до пристосовності системи до змін: програмне забезпечення повинно мати структуровану базову реалізацію та визначену архітектуру побудови системи для підтримки одного із основних принципів розробки програмного забезпечення SOLID, а саме підтримку open–closed принципу або один із решти таких як single responsibility, Liskov substitution, interface segregation і dependency inversion, що дозволяє у майбутньому підтримувати додаток значно швидко, підтримуючи принцип коду, який робить його відкритим для розширення, але закритим до змін, тобто вихідний код певного модуля або підсистеми зачинений і ніхто не має права вносити до нього зміни, що також зменшить ризики поломки системи або окремого функціоналу для користувача при цих змінах.

Вимоги до надійності програмного забезпечення: система повинна працювати успішно та не мати жодних критичних помилок, які можуть спричини втрату даних або їх зміну. Дії користувача не повинні призводити до фатальних помилок додатку, які можуть спричинити його зависання, моментальне зачинення додатку, безперервне або «довічне» завантаження ресурсу, даних або інших компонентів і додатку в цілому.

Вимоги до захисту інформації від несанкціонованого доступу: доступ до даних потенційних користувачів системи повинен бути максимально захищений. Додаток повинен забезпечувати користувача гарантійними точними засобами, які виключають будь-який, навіть найменший та найменш вірогідний варіант витоку даних у мережу або через інші канали витоку даних.

Вимоги до антивірусного захисту: розроблюване програмне забезпечення повинне містити певні інструменти, параметри та методи для захисту додатку від стороннього впливу, а саме: додаток повинен бути унеможливлений з поєднанням сторонніх ресурсів та додатків, без попереднього узгодження розробників даних додатків. При будь-яких діях користувача, додаток повинен мати спеціально визначений функціонал на перевірку всіх файлів чи функцій на виявлення шкідливих алгоритмів та систем стороннього ресурсу третіх осіб, які можуть мати вплив або доступ до файлів розроблюваного додатку.

Вимоги до контролю, зберіганню, оновленню та відновленню даних: при діях користувача, що спрямовані на зміну його даних через СУБД, додаток повинен повторно робити повторний запит через графічний інтерфейс для підтвердження або відміну вибраної дії, а при діях, які мають невідворотний характер або високий ризик втрати великої кількості даних додатково повідомляти користувача також через графічний інтерфейс про наслідки дії.

1.4 Висновки до розділу 1

Розробка програмного забезпечення – справа, яка змінюється найшвидше серед усіх наявних на ринку праць. Одні методи замінюються на інші, одні компанії випускають нові оновлення на середовища програмування і концепт, і підхід до розробки може змінювати максимально швидко, що вимагає сильну пристосовуваність до змін. Саме тому, важливо використовувати саме новітні найкращі практики використання принципів розробки, що підходять під визначений проект та користуються попитом на ринку, адже це забезпечує вдале функціонування нового програмного забезпечення і його підтримку.

Опис даної предметної області, а саме сфера з представників малого бізнесу у торгівлі вживаними електронними гаджетами дозволяє чітко відокремити та зрозуміти пункти, для чого потрібен розроблюваний додаток на мобільні пристрої та які функції повинен він виконувати, щоб забезпечити його попит на ринку та корисність у користуванні, підвищення прибутків для представників підприємств.

Описані вимоги до розроблюваного програмного забезпечення дозволило визначити ключові і чітко сформульовані властивості, якості та функції програмного додатку, що буде розроблено.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір середовища розробки, мови програмування та відповідного інструментарію

Починаючи із 2016 року, кількість завантажених мобільних додатків у світі зросла близько на 60%. Саме це зростання породжує і конкуренцію між собою на ринку певних платформ та мов програмування [5].

Графік зростання кількості завантажених додатків у світі у мільярдах по відношенню до кожного року зображено на наступній діаграмі (рис 2.1).

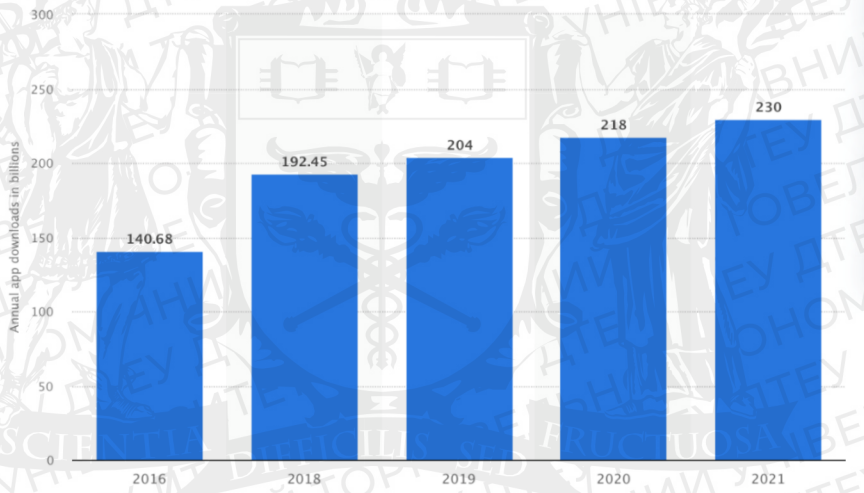


Рисунок 2.1. Діаграма зростання кількості завантажень

Джерело: побудовано на основі джерела [5]

Для розробки мобільних додатків існує не надто багато мов програмування та платформ на яких можна їх будувати, такі як Kotlin, Swift, React-Native, Flutter та інші. Проаналізувавши конкуруючі між собою

					ДТЕУ 121 06-24 БР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф		Криворучко О.В.		03.03.23	*Програмний модуль інформаційної системи підприємства торгівлі вживаними електронними гаджетами*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Десятко А.М.		03.03.23		Р2	16	50
Гарант		Рзаєва С.Л.		03.03.23		*Факультет інформаційних технологій 4 курс 6 група*		
Розробив		Сясько Д.В.		03.03.23	*Проєктування програмного забезпечення*			

платформи, всі плюси та мінуси вище поданих платформ, можна виділити, що найкращим варіантом для виконання поставленого завдання буде вибрано Flutter.

Flutter — це безкоштовний набір середовищ розробки (SDK) для розробки першокласних інтуїтивно зрозумілих графічних інтерфейсів для iOS і Android операційної системи, створений компанією Google. Flutter тісно пов'язаний із такою мовою програмування, яка має назву – Dart.

Dart — це об'єктно-орієнтована мова програмування загального призначення з відкритим вихідним кодом. Він знаходиться в стадії активної розробки, скомпільований до рідного машинного коду для створення мобільних додатків, натхненний іншими мовами програмування, такими як Java, JavaScript, C#, і строго типізований [6].

За даними ресурсу «Stack Overflow», Flutter займає 6 місце серед найпопулярніших технологій, що використовуються у розробці і передує серед конкуруючих платформ розробки мобільних додатків (рис 2.2).

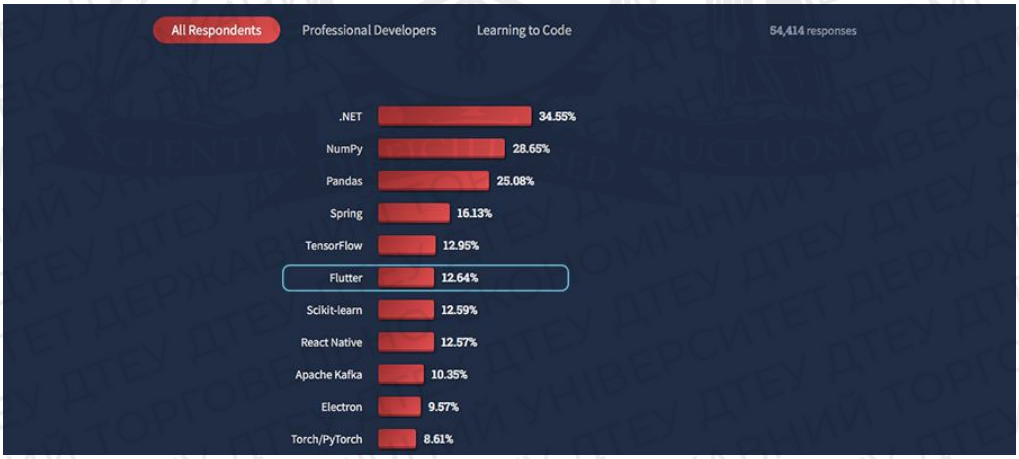


Рисунок 2.2. Графік використання технологій розробниками

Джерело: побудовано на основі джерела [7]

Можна виділити наступні переваги, які надає вибраний комплекс розробки і мова програмування:

- швидкість розробки: Flutter підтримує крос-платформенність, а

саме не потрібно використовувати 2 різні мови програмування та комплекси розробки, підходи та побудову додатку на кожну платформу окремо. Розробка ведеться одночасно на обидві операційні мобільні системи iOS і Android;

- постійна підтримка комплексу від компанії Google, яка є їх розробниками. Flutter та Dart мають постійні та регулярні оновлення, які тільки пришвидшують їх роботу та усувають недоліки;
- доступний індивідуальний анімований інтерфейс будь-якої складності полегшений у розробці, порівнюючи із іншими мовами програмування;
- реалізація малювання інтерфейсу реалізовано на окремому двигуну від Flutter, що передує у швидкості та мінімалізує затрати у ресурсах пристрою [8].

Flutter та мова програмування Dart успішно контактує із інтегрованим середовищем розробки (IDE) Android Studio. Дане середовище буде використовуватися для розробки мобільного додатку, так як воно має усі функції, як основні, так і допоміжні, для керуванням процесу розробки і тестуванням мобільного додатку, а саме: створення, керування даними, тестування додатку на фізичному пристрої або створеному електронному емуляторі Android операційної системи. Середовище Android Studio дозволяє встановлювати додаткові плагіни для Flutter розробки додатків, а також генерацію стартових файлів та конфігурації додатку.

Також у додаток буде інтегрована база даних. Одним із найкращих варіантів додавання БД у додаток побудований на Flutter – це SQLite.

Розробка додатків на Flutter майже ніколи не відбувається без підключення певних потрібних нам пакетів, які містяться на офіційному ресурсі Flutter та Dart. Існує буквально сотні різних допоміжних пакетів для

розробки, які середовище пропонує нам інсталювати перед їх використанням з відповідними посиланнями. Це значно пришвидшує роботу додатку, тому що розробник додає лише ті пакети, які йому потрібні, відкидаючи все зайве, що значно зменшує витрати ресурсів пам’яті та інших локального мобільного пристрою.

Додатково для розробки додатку потрібно виділити деякі додаткові бібліотеки для розробки графічного інтерфейсу додатку та внутрішньої логіки взаємодії, а саме такі: AutoRoute – бібліотека для покращеної взаємодії навігації у додатку, Freezed та BuildRunner – для генерації відповідних файлів, що посилюють цілісність коду. Обов’язковою підключеним пакетом повинен бути FlutterBloc та формування архітектури додатку, яка буде описана далі та швидкий зв’язок між графічним інтерфейсом користувача та логікою розроблюваного додатку. Інтегрування бази даних у середовище Flutter відбуватиметься за допомогою плагіну Sqflite, що є додаток для бази даних SQLite.

2.2 Побудова архітектури розроблюваного програмного забезпечення

Програмне забезпечення розробляється на мобільні платформи, такі як Android, iOS за допомогою комплексу розробки Flutter. Як було згадано вище, для розробки мобільних додатків прийнято за використання архітектури Model-View-Presenter (MVP) або ж Model-View-ViewModel (MVVM), як за основу.

Проте, за рахунок власного двигуна Flutter, він дозволяє розробнику використати нову архітектуру, яка стала найкращою практикою серед усіх Flutter розробників. Дана практика носить назву – Business Logic Component (BLoC).

BLoC - це архітектурний патерн, який базується на окремих компонентах. Компоненти містять лише бізнес-логіку додатку. ЇЇ можна легко

використовувати між різними частинами додатку. Дана архітектура була представлена у 2019 компанією Google, що свідчить про те, що вона є новою і зібрала в собі вирішення усіх проблем старих архітектурних патернів [12].

Принцип роботи заключається в тому, коли користувач взаємодіє з інтерфейсом, він надсилає цю подію у компонент, цей компонент цю дію обробляє і повертає новий стан цього ж компоненту інтерфейсу користувача і тим самим оновлює його. Роботу даного методу можна зобразити схематично (рис 2.3) [13].

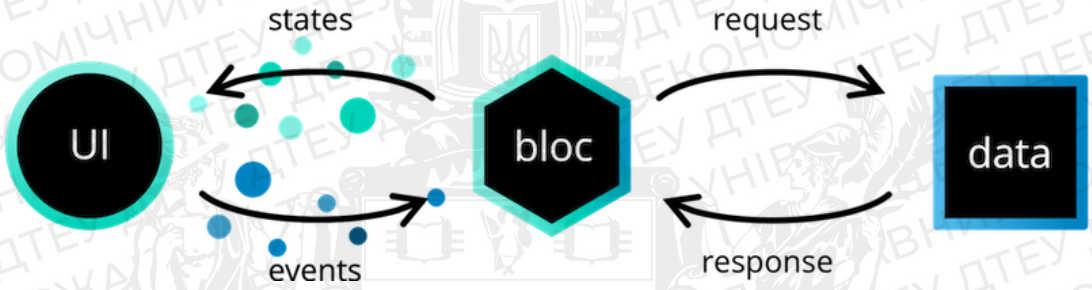


Рисунок 2.3. Принцип роботи патерну BLoC

Джерело: побудовано на основі джерела [13]

Архітектура BLoC розроблена спеціально для комплексу Flutter, що ще раз підкреслює його найбільш досконалим вибором під розробку мобільних додатків на цій платформі, основна логіка роботи цього патерну заключається у відокремленні логіки додатку від інтерфейсу користувача та керування «станами» додатку або певних дій чи логіки. Це надає безліч переваг та плюсів, а саме:

- додаток розробляється значно швидше, завдяки тому, що винесену логіку можна заново використати у різних місцях проекту та підв'язати до неї окремий інтерфейс;
- легкий у тестуванні, можливість додавання Unit тестування;
- можливість удосконалення чи модифікації додатку стає мінімально простою і швидкою;

- швидкість роботи додатку значно швидша, адже за даною архітектурою оновлення відбувається значно швидше;
- підтримка даного патерну відбувається самим Flutter, тому постійні оновлення приводять тільки до пришвидшення та покращення роботи даного методу.

На основі визначеної архітектури можна побудувати наступну діаграму класів та їх зв'язків, характеристик, основні котрі будуть використані для побудови програмного додатку (рис 2.4).

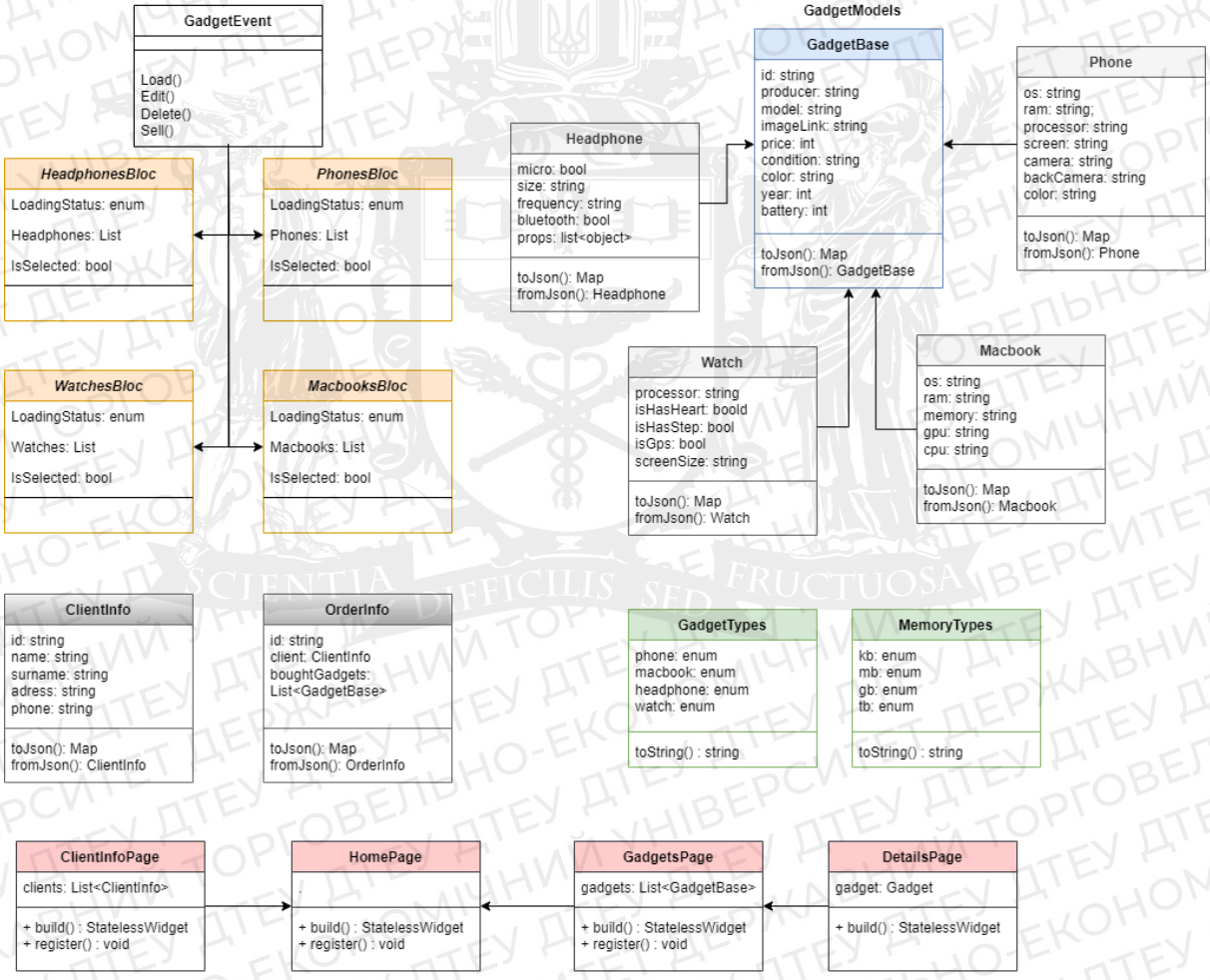


Рисунок 2.4. Діаграма класів розроблюваного програмного додатку

Джерело: побудовано автором в системі VisualParadigm (знімок з екрану)

Також, визначивши основні сценарії взаємодії додатку із локальними компонентами, потенційними користувачами додатку та

клієнтів цих ж користувачів, подано опис поведінки системи під час взаємодії із зовнішнім середовищем на діаграмі (рис. 2.5).

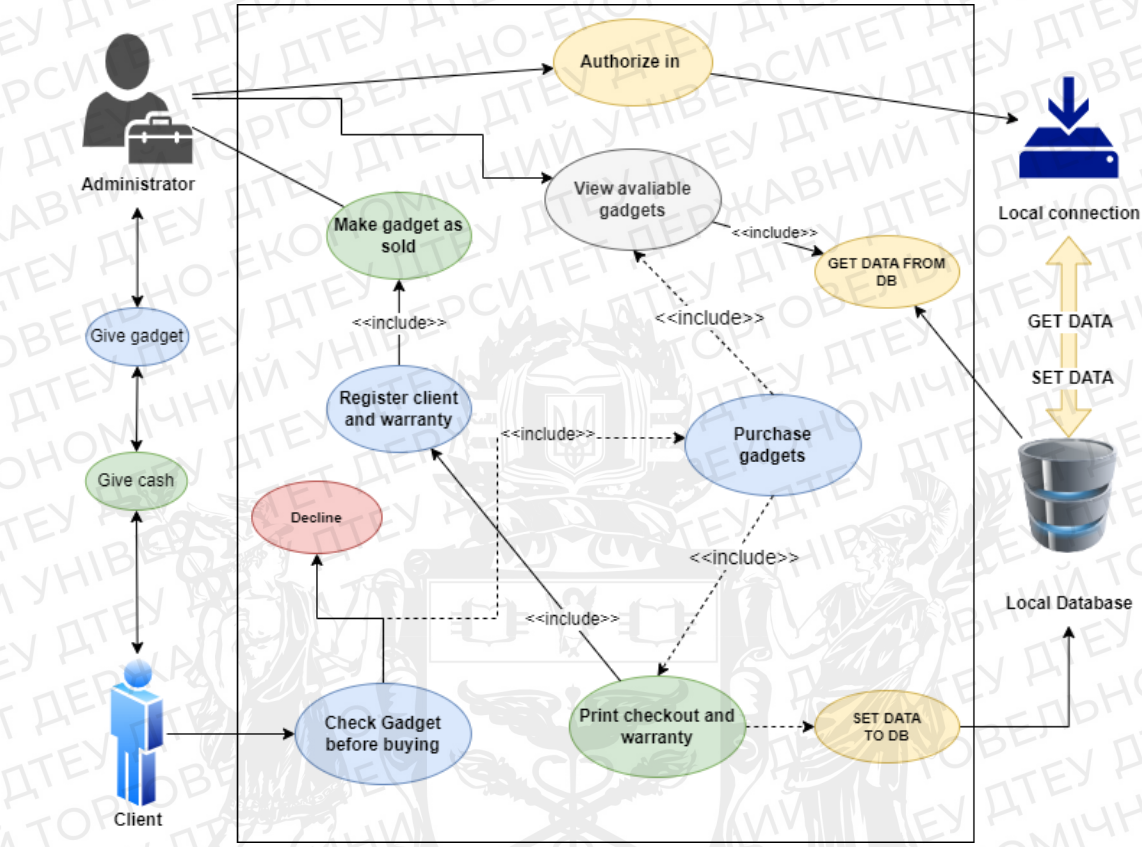


Рисунок 2.5. Модель взаємодії інформаційної системи

Джерело: побудовано автором в системі AppDiagrams (знімок з екрану)

Визначивши головні процеси взаємодії системи із певними чинниками, далі визначений алгоритм, який зображує основний алгоритм роботи розроблюваного програмним додатку. (рис. 2.6).

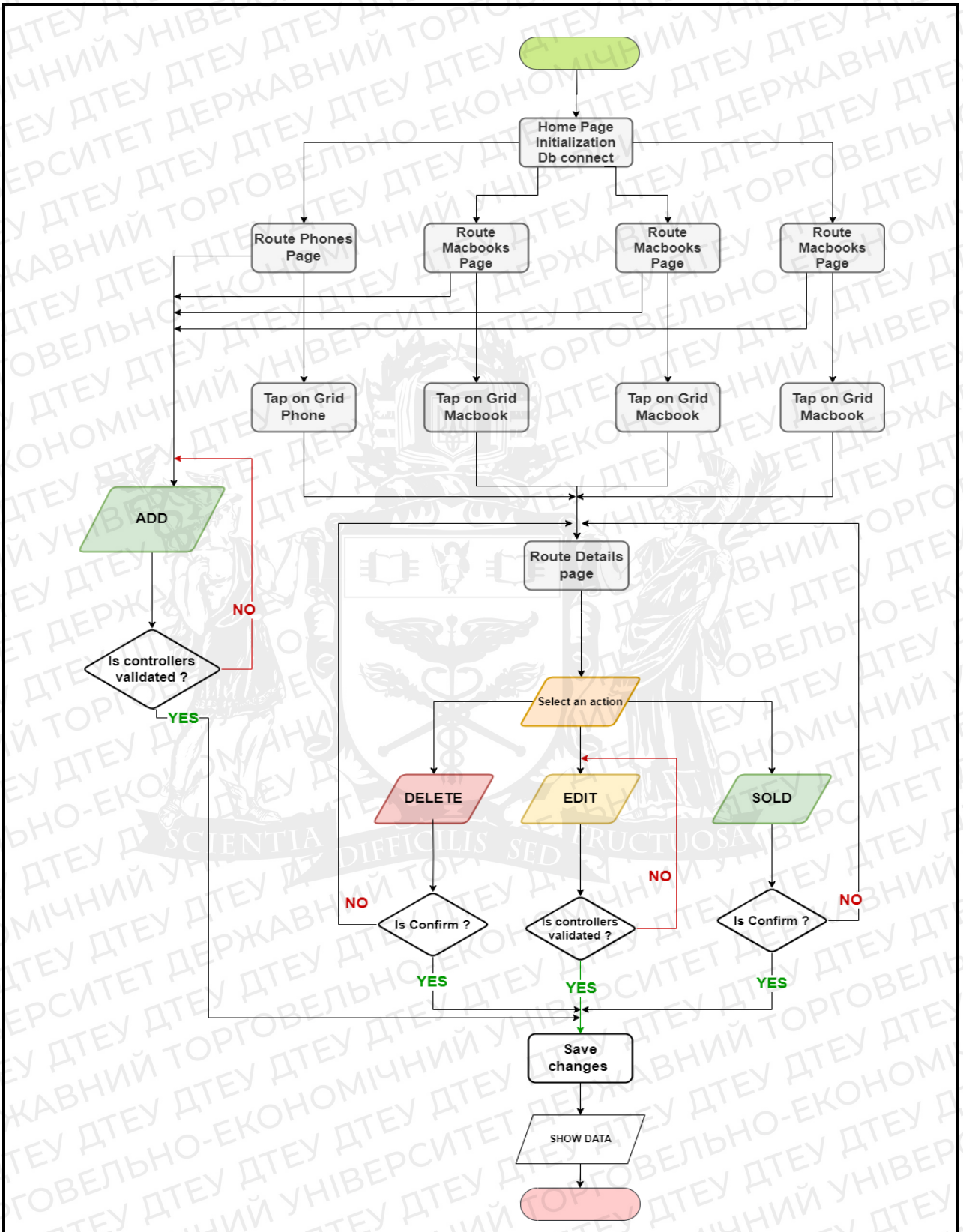


Рисунок 2.6. Алгоритм програмного додатку

Джерело: побудовано автором в системі AppDiagrams (знімок з екрану)

Далі можна розглянути діаграму кооперації, на якій графічно проілюстровано те, як можлива у програмному додатку відбуватись послідовність дій користувача. Також на діаграмі проілюстровано певні типи об’єктів та самі об’єкти зі своїми заповненими даними та полями, що створюються або змінюються при вказаних діях користувача, а також зв’язки між даними, що перебувають у різних об’єктів (рис. 2.7).

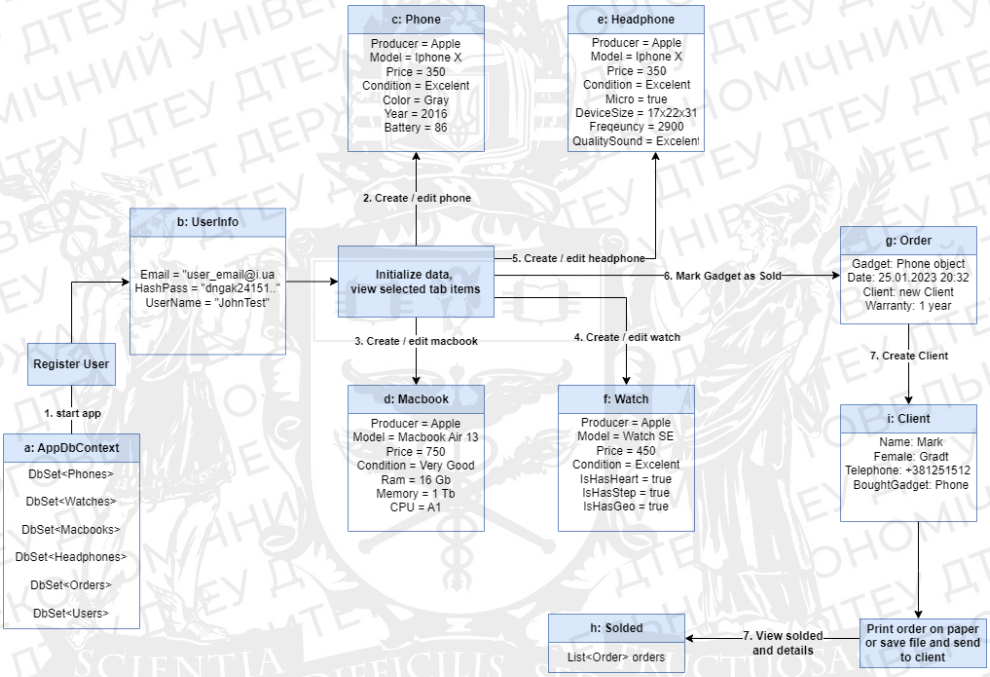


Рисунок 2.7. Діаграма кооперації програмного додатку

Джерело: побудовано автором в системі AppDiagrams (знімок з екрану)

Наступна UML-діаграма розроблена для представлення компонентів, що є важливим інструментом у розробці програмного забезпечення, оскільки вона надає високорівневе уявлення про структуру системи та компонентів, з яких вона складається. Основна мета діаграми компонентів полягає в тому, щоб зобразити компоненти системи та зв'язки між ними, які визначають, як ці компоненти взаємодіють один з одним (рис. 2.8).

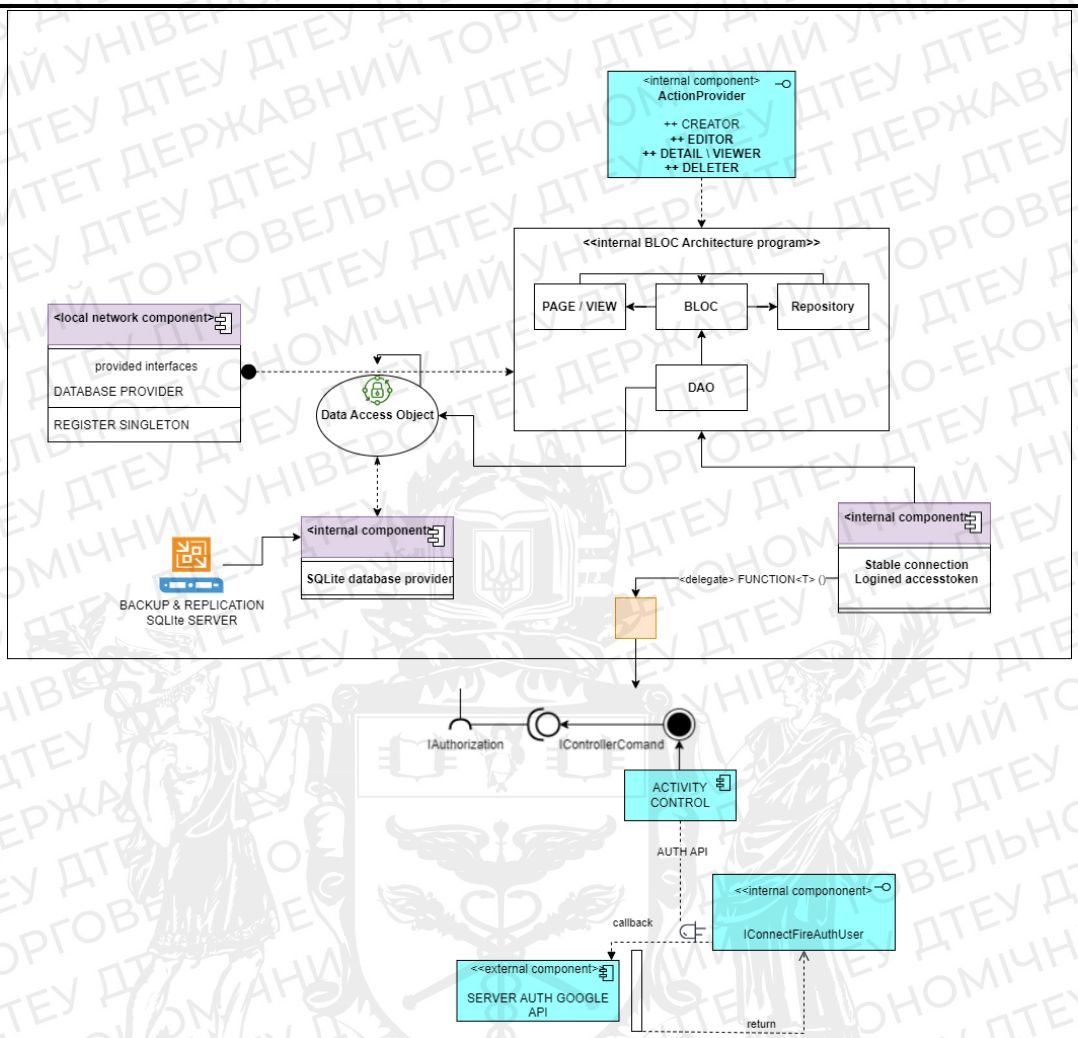


Рисунок 2.8. Діаграма компонентів програмного додатку

Джерело: побудовано автором в системі AppDiagrams (знімок з екрану)

Усі дані, які будуть використовуватися у програмному додатку зберігатимуться і братимуться із бази даних. На основі класів та моделей даних побудовано певні моделі бази даних. На фізичній моделі зображено таблиці та дані, які зберігатимуться у певній із таблиць, а також відповідно до цього їх тип даних для поля та їх зв'язки між собою.

Виділено таблицю GadgetBase, яка має перелічені спільні властивості гаджетів, які наявні. Наприклад, таблиця Phone не дублює дані із таблиці GadgetBase, так як вона успадковує всі потрібні їй поля і залишає лише унікальне їй поле, наприклад: CameraResolution. Це зроблено для того, щоб

зменшити кількість повторюваних даних в базі даних та забезпечити більш ефективне зберігання інформації. Також, це спрощує процес зміни даних, якщо знадобиться внести зміни в загальні поля GadgetBase, які будуть відображені у всіх таблицях, що успадковують від нього.

Важливою складовою у побудові моделі бази даних є побудова зв'язків між таблицями та їх даними, що все разом зображено на фізичній моделі бази даних (рис. 2.9).

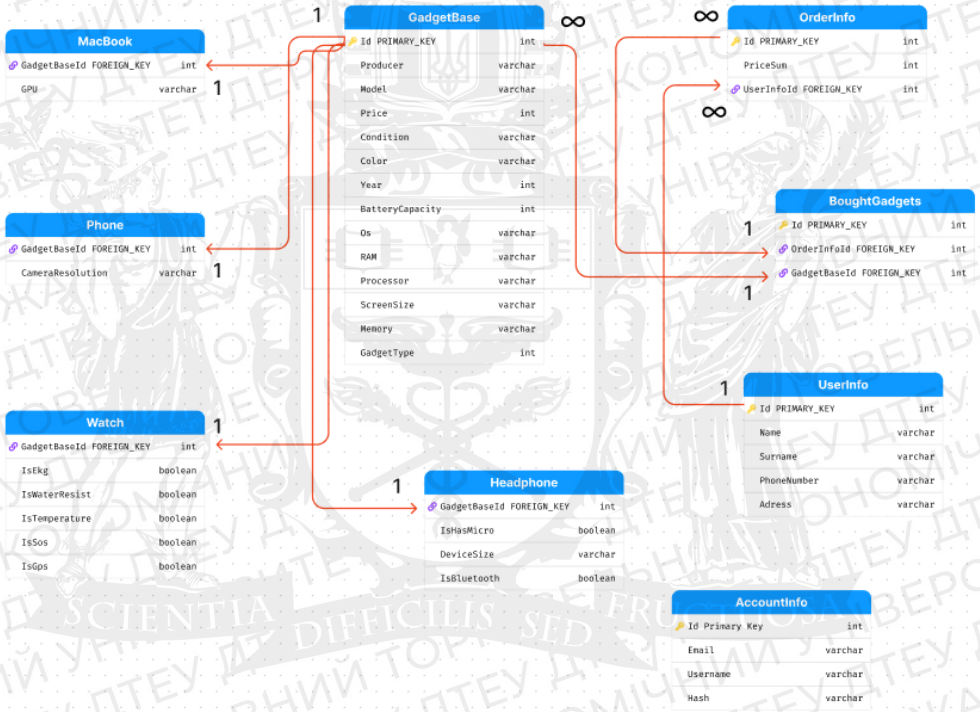


Рисунок 2.9. Фізична модель із зв'язками бази даних

Джерело: побудовано автором в системі Figma (знімок з екрану)

Один гаджет може бути тільки одного з типів (годинник, телефон, ноутбук, навушники), тому використовується зв'язок один-до-одного між таблицею GadgetBase та її підтаблицями. Зв'язок один-до-багатьох між таблицями UserInfo та OrderInfo - кожен користувач може здійснити багато замовлень, але кожне замовлення може бути оформлене тільки одним користувачем.

Зв'язок один-до-багатьох між таблицями OrderInfo та GadgetInfo. Кожне замовлення може містити декілька гаджетів, а один гаджет може бути включений в тільки одне замовлення. Для представлення цього зв'язку і була побудована проміжна таблиця BoughtGadgets, яка містить зовнішні ключі до таблиць OrderInfo та GadgetInfo.

Нормалізація бази даних є один із найважливіших факторів у розробці. Це зможе забезпечити цілісність даних користувача, а також можливістю для легкого розширення даних. Це не призведе до втрати даних чи поломки додатку та бази даних. Подана база даних знаходиться у третій нормальній формі (3НФ), оскільки:

- усі стовпці кожної таблиці містять атомарні значення;
- усі поля таблиць залежать від первинного ключа цієї таблиці;
- усі неключові поля в таблиці залежать від первинного ключа цієї таблиці, а не від інших неключових полів;

Таким чином, база даних у 3НФ дозволяє зменшити дублювання даних та забезпечити ефективність операцій з базою даних та забезпечує високий рівень надійності даних.

2.3 Побудова графічного інтерфейсу та макету додатку

Побудова макету відбуватиметься у системі Figma, яка має безліч відповідних інструментів для попередньої побудови графічного інтерфейсу, з яким повинен працювати потенційний користувач. Попередня розробка дизайну додатку дозволяє на етапі малювання та проектування визначити основні елементи графічної частини додатку та визначити плюси та мінуси певної вигаданої компоновки елементів.

При першому запуску програмному додатку, додаток повинен зустрічати користувача сторінкою із логіном, для вводу даних зареєстрованого

клієнта через електронну пошту та пароль. У випадку, якщо користувач не має зареєстрованого акаунту – також є кнопка для реєстрації акаунту, де і запрошуються відповідні поля, що необхідні (рис. 2.10)

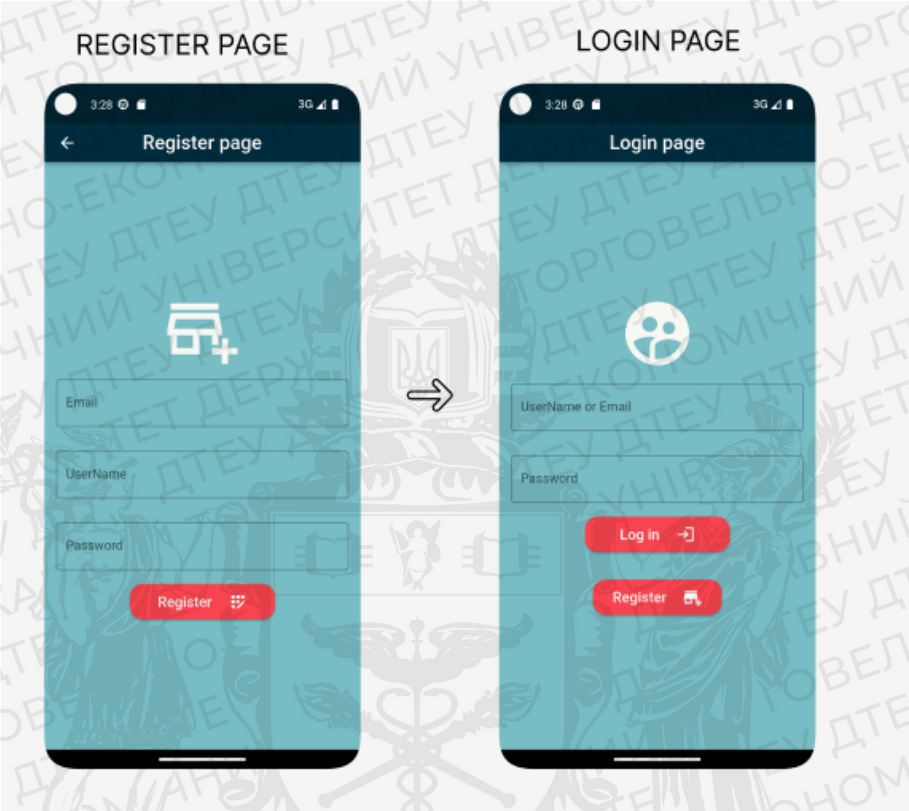


Рисунок 2.10. Сторінки логіну та реєстрації користувача

Джерело: побудовано автором в системі Figma (знімок з екрану)

Користувача також повинна зустрічати домашня сторінка, від якої він може відштовхуватися. На даному макеті показано домашню сторінку додатку із підказкою, а також розроблене меню, для наступної навігації користувача.

При натисканні на один із елементів меню при завантаженні даних повинна відображатись анімація завантаження. (рис. 2.11).

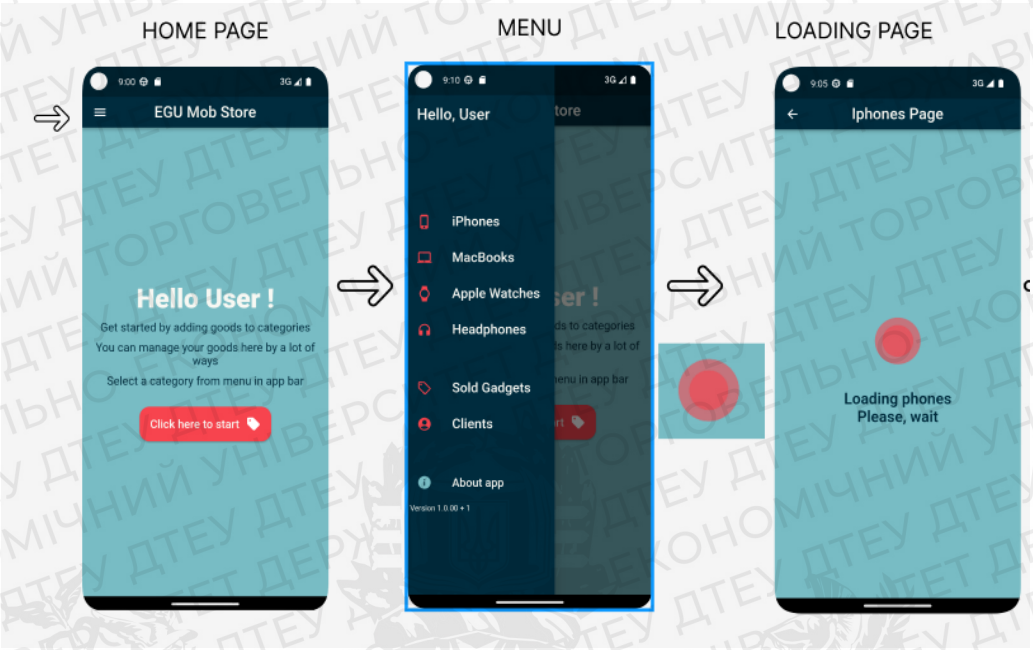


Рисунок 2.11. Стартові елементи інтерфейсу програмного забезпечення

Джерело: побудовано автором в системі Figma (знімок з екрану)

Наступними елементами інтерфейсу, наприклад, якщо користувач вибрав завантаження телефонів із бази даних – список гаджетів буде відображено плитковою формою, що містить в собі модель та виробника, а також ціну та стан гаджету, в даному випадку – телефона. Даний список повинен бути scrollable, а при натисканні на гаджет – відкриватиметься сторінка із деталізованим описом гаджету та панеллю керування даним гаджетом у вигляді кнопок. Кнопка додавання гаджету або редагування буде переносити користувача на сторінку із полями вводу даних, обладнаними меню із виборами та текстовими полями вводу даних (рис. 2.12).

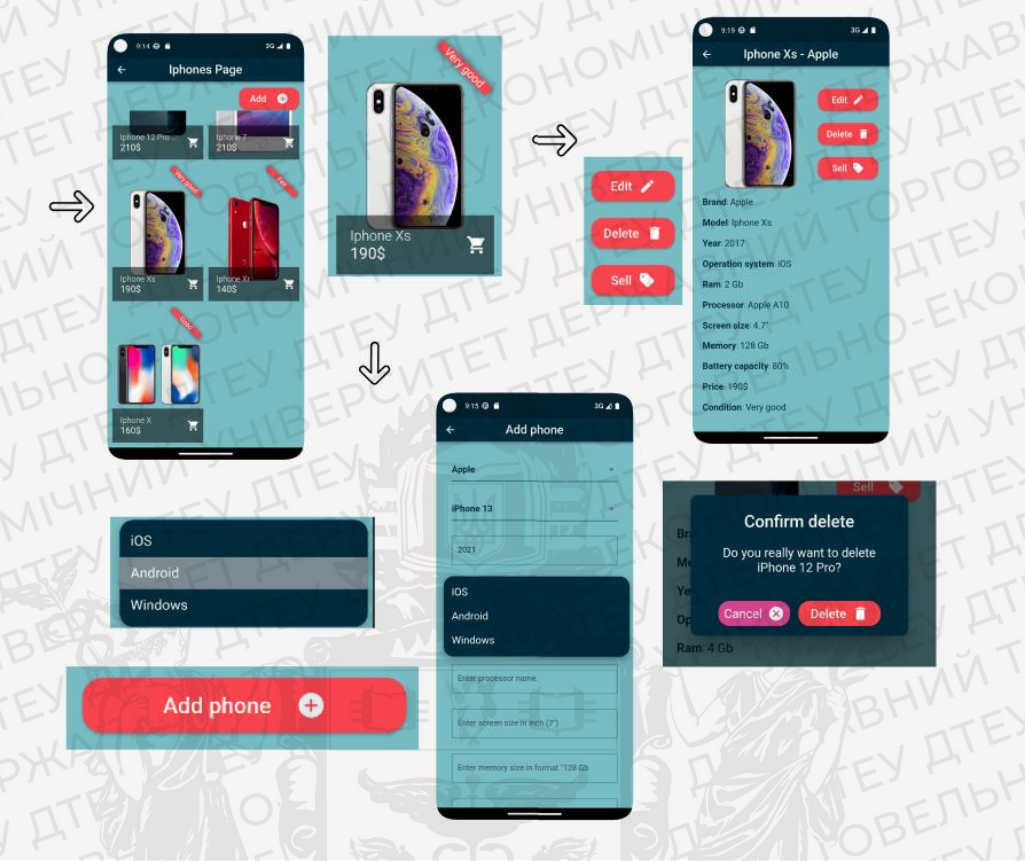


Рисунок 2.12. Основні елементи користування інтерфейсом

Джерело: побудовано автором в системі Figma (знімок з екрану)

Даний вище описаний функціонал додатку буде зберігатися для решти сторінок, які переходять від меню з незначним відходженням та редагуванням графічних елементів, проте важливо підтримувати схожість функціоналу для решти сторінок, що робить додаток User-Friendly – поняття, яке характеризує додаток, як простий у користування і легкий в розумінні на погляд та підтримує стандарти User Experience (UX).

2.4 Висновки до розділу 2

У ході роботи із проєктування програмного забезпечення було визначено та вирішено проблеми із вибором основних принципів розробки, методів та інструментарію, що будуть використовуватися при розробці.

Вибрано один із найкращих інструментаріїв розробки для даного проекту, завдяки його значним перевагам. Це вирішило безліч проблем, які б могли ускладнити подальше оновлення додатку, його удосконалення та покращення або добавлення нового функціоналу.

Побудовані UML-діаграми дають загальне уявлення про роботу додатку у середовищі, його функціонування, а також дають зрозуміти загальні концепти побудови і конкретизовані частини розробки, такі як діаграма класів програмного забезпечення.

Особливу роль у функціонуванні додатку відіграє база даних, логічна і фізична модель якої побудована правильно, із визначенням усіх даних та їх типів, певних зв’язків між таблицями, із нормалізацією, що гарантує вирішення проблем, які б могли викликати труднощі або унеможливлення із ефективного та надійного зберігання та маніпулювання даними.

Розроблений макет додатку у сервісі Figma дозволить візуалізувати та протестувати графічний інтерфейс програмного забезпечення, а також зробить розробку Front-end частини додатку у систематичному способі, що забезпечить успішне втілення проекту.

РОЗДІЛ 3
РОЗРОБКА КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка Back-End частини додатку

Одним із найголовніших завдань при розробці програмного забезпечення, яке націлене на користування людиною для збереження та керування особистими є захистом цих даних та забезпечення у відсутності несанкціонованого доступу до закритої для публічного доступу інформації зі сторони третіх осіб, які не мають певного дозволу на доступ та сторонніх програмних забезпечень та програм.

Саме тому перша сторінка, яка зустрічає користувача, а саме це сторінка із логіном – вводом адреси електронної пошти та паролю, що задаються при реєстрації, повинна бути реалізована за основними вимогами та за допомогою нових методів розробки. Дані користувача будуть зберігатись у локальній базі даних, проте зберігати їх потрібно у спеціально зашифрованій формі, а саме - проводити хешування паролю користувача.

Суть методу хешування полягає в перетворенні паролю в рядок, який має фіксовану довжину – вона є результатом застосування криптографічної функції хешування. Вона є унікальною і найголовніше – незворотною, тобто пароль можна закодувати, але розкодувати – неможливо. Це і забезпечує дані безпекою і робить неможливою «витік» паролю в сторонні ресурси, дозволяє зберігати дані не розкриваючи їх вихідного значення. При логіну користувача пароль, що він вводить також закодовується і порівнюються ці рядки, якщо вони рівні – користувач успішно отримує доступ у додаток.

Хешування паролю відбувається за допомогою SHA-2 – сімейства криптографічних алгоритмів та UTF-8 – стандарту кодування символів (рис. 3.1)

```dart
// ignore_for_file: depend_on_referenced_packages

import 'dart:convert';
import 'package:crypto/crypto.dart';

You, 3 days ago | 1 author (You)
class PasswordHasher {
  static String hashPassword(String password) {
    var bytes = utf8.encode(password);
    var digest = sha256.convert(bytes);
    return digest.toString();
  }
}

```

Рисунок 3.1. Хешування паролю за допомогою криптографічних функцій
Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Перевірка введених даних користувача відбувається за таким чином: спочатку перевіряються форми на правильність введених даних, а саме: пароль повинен містити мінімум 8 символів та 1 цифру. Додатково для підтримки основних принципів User Experience додано спеціальні повідомлення, якщо користувач помилився паролем або ж електронною поштою, яку вказував при реєстрації. Коли користувач натиснув кнопку «Log in» - додаток посилає запит у базу даних та витягує звідти дані, а саме електронну пошту та відповідно хеш паролю, що прив'язаний до адреси і перевіряються на рівність. При правильності даних – користувач перенаправляється на домашню сторінку додатку (рис. 3.2).

```
Future<void> _loginUser() async {
  var userLogerInfo =
      await DatabaseHelper.getCredentialsFromEmail(_emailController.text);
  if (userLogerInfo != null) {
    var hashedInDb = userLogerInfo.hashPassword;
    if (hashedInDb == PasswordHasher.hashPassword(_passwordController.text)) {
      context.router.navigate(const HomeRoute());
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
          CustomSnackBars.getErrorSnackBarWithCustomText('Wrong password'));
    }
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
        CustomSnackBars.getErrorSnackBarWithCustomText(
            'User not found, please register'));
  }
}
```

Рисунок 3.2. Метод перевірки хешів паролів при логіну у додаток

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Розробка моделей даних додатку, а саме об'єктів і їх зв'язки між собою є важливою частиною, яка повинна використовується всюди по додатку, для підтримування основ об'єктно-орієнтованого програмування – задавати всі значення – як об'єкти. Саме тому, для уникнення дублювання коду і вибудовування певних залежностей між моделями було додано базову модель гаджета з основними її полями та властивостями і відповідно ті інші моделі гаджетів, які унаслідуються від нього, перевизначають певні методи, що є основним принципом ООП – поліформізмом.

Одним із основних частин, які потрібно безпомилково і правильно розробити – це навігація по мобільному додатку. Адже у випадку, якщо навігація у таких додатках спрацьовує некоректно, програмне забезпечення стає майже неспроможним для функціонування. Розробка навігації для додатку EGUMobStore відбуватиметься за допомогою додаткового пакету auto_route. Потрібно при запуску додатку створювати об'єкт, за архітектурним патерном «Одинак» або «Singleton», котрий буде відповідати за об'єкт навігації, що забезпечить практично найкращий із можливих захист додатку від поломки при некоректних навігаціях (рис. 3.3).

```dart
abstract class AppLocator {
  static Future<void> setUp() async {
    getIt.registerSingleton<EguRouter>(EguRouter(navigatorKey));
  }
}

```

Рисунок 3.3. Ініціалізація об'єкту навігації у додатку

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Також для кожної сторінки інтерфейсу користувача потрібно прив'язувати посилання навігації з певним атрибутом і запустити команду flutter pub run build_runner watch у терміналі для генерації відповідних файлів для ще більшого захисту від поломок (рис. 3.4).

```dart
@MaterialAutoRouter(
  replaceInRouteName: 'Page|Screen|Form,Route',
  routes: <AutoRoute>[
    AutoRoute(page: LoginPage, initial: true),
    AutoRoute(page: RegisterPage),
    AutoRoute(page: HomePage),
    AutoRoute(page: IphonesPage),
    AutoRoute(page: MacBooksPage),
    AutoRoute(page: HeadphonesPage),
    AutoRoute(page: WatchesPage),
    AutoRoute(page: GadgetDetailsPage),
    AutoRoute(page: AddPhoneForm),
    AutoRoute(page: AddMacbookForm),
    AutoRoute(page: EditPhoneForm),
    AutoRoute(page: AddClientForm),
    AutoRoute(page: ClientsPage),
    AutoRoute(page: ClientDetailsPage),
    AutoRoute(page: EditClientForm)
  ], // <AutoRoute>[]
)
class $EguRouter {}

```

Рисунок 3.4. Прив'язування навігації до відповідних сторінок інтерфейсу

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

База даних – також є однією із основних частин розроблюваного програмного забезпечення і правильна її ініціалізація потребує уваги, адже помилки із базою даних можуть призвести до втрати даних та поломки

додатку або інших його проблем. Для цього створено окремий клас DatabaseHelper для керування методами, а також ініціалізації бази даних. При запуску додатку метод перевіряє чи вже наявна ця база даних і якщо вона відсутня – створює її, що виключить можливість створення декількох об'єктів доступу до бази даних або створення дублікату бази. При зверненні до бази даних застосовуються ті ж методи, що унеможливлюють появу вище вказаних проблем (рис. 3.5).

```
53  static Future<Database> get database async {
54    if (_database != null) return _database!;
55
56    _database = await initDatabase();
57    return _database!;
58  }
59
60  static Future<Database> initDatabase() async {
61    final databasePath = await getDatabasesPath();
62    final path = join(databasePath, 'my_database.db');
63
64    return await openDatabase(path, version: 1, onCreate: _createDatabase);
65  }
```

Рисунок 3.5. Ініціалізація бази даних у додатку

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Основною частиною, на якій базується весь додаток це архітектура програмного забезпечення і як було зазначено у попередніх розділах – це архітектура Bloc. Для її розробки потрібно перш за все створити певну архітектуру та узагальнення тек, для побудови логіки блоку потрібно розробити такі елементи, як state, event та bloc файл а також додатково згенерувати freezed файл для підтримки основного принципу об'єктно-орієнтованого принципу – інкапсуляція і заборонити методами privacy керувати станами додатку із місць, де це не дозволено, а саме – ззовні блоку.

Для завантаження гаджетів, наприклад, телефонів, може бути декілька станів, такі як «завантаження», «завантажено», «помилка» (рис. 3.6).

lib > blocs > phones_bloc > phones_state.dart

```dart
part of 'phones_bloc.dart';


enum PhonesLoading {
  loading,
  loaded,
  error,
}

@freezed
class PhonesState with _$PhonesState {
  const factory PhonesState({
    @Default(PhonesLoading.loading) PhonesLoading loading,
    @Default([]) List<Phone> phones,
  }) = _PhonesState;
}

```

Рисунок 3.6. Клас PhonesState для зберігання та керування станами блоку.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Для керування гаджетами та певними діями із ними користувачем, наприклад для видалення чи редагування і так далі, всі дії повинні проходити через блок для підтримки архітектури додатку, а також для кращої підтримки відловлювання помилок. Саме для цього і створюються event класи, тобто події, які надалі повинні підв'язуватись до візуальної частини додатку і передаватись на блок де і повинна відбуватися вся основна логіка події (рис. 3.7).

lib > blocs > phones_bloc > phones_event.dart > ...

```dart
You, 2 weeks ago | 1 author (You)
part of 'phones_bloc.dart';

@freezed
You, 2 weeks ago | 1 author (You)
class PhonesEvent with _$PhonesEvent {
  const factory PhonesEvent.loadPhones() = LoadPhones;
  const factory PhonesEvent.deletePhone(Phone phone) = DeletePhone;
  const factory PhonesEvent.editPhone(Phone phone) = EditPhone;
  const factory PhonesEvent.addPhone(Phone phone) = AddPhone;
  const factory PhonesEvent.sellPhone(Phone phone) = SellPhone;
}

```

Рисунок 3.7. Клас PhonesEvent для керування подіями блоку.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Приступаючи до побудови основної частини, де виконується логіка усіх подій та управління станами, потрібно створити основний файл bloc, обов'язково застосувати принцип успадковування від flutter_bloc пакету і прив'язати до кожної події (event), що описаний вище певний метод обробки. Для прикладу можна взяти подію завантаження списку гаджетів певного типу, а саме – телефонів. Подія, отримана через графічний інтерфейс користувача із певної кнопки, передається на bloc, де відразу направляється на певний метод, що вказаний у конструкторі класу PhonesBloc. Саме у методах блоку повинен змінюватися state (стан) для коректної роботи додатку. У методі завантаження відбувається початкова зміна стану на «завантаження», яке буде в свою чергу певним чином відображатися на інтерфейсі користувача, завантаження списку гаджетів із бази даних і наступним оновленням стану «завантажено». Під час зміни станів – оновлюється та графічний інтерфейс і відповідно користувач побачить список завантажених гаджетів. У випадку помилки стан змінюється на «помилку» і користувач теж побачить відповідне повідомлення (рис. 3.8).

lib > blocs > phones_bloc > phones_bloc.dart > PhonesBloc > _editPhone

```
16
      You, 7 seconds ago | 1 author (You)
17    class PhonesBloc extends Bloc<PhonesEvent, PhonesState> {
18      PhonesBloc() : super(const PhonesState()) {
19        on<LoadPhones>(_loadPhones);
20        on<DeletePhone>(_deletePhone);
21        on<EditPhone>(_editPhone);
22        on<AddPhone>(_addPhone);
23        on<SellPhone>(_sellPhone);
24      }
25
26      FutureOr<void> _loadPhones(
27          LoadPhones event, Emitter<PhonesState> emit) async {
28        try {
29          emit(state.copyWith(loading: PhonesLoading.loading));
30
31          var allGadgets = await DatabaseHelper.getAllPhones();
32          var list = <Phone>[];
33          for (var element in allGadgets) {
34            list.add(element.copyWith(
35                imageLink: ImageHelper.getPhoneImage(element.model!)));
36          }
37          await Future.delayed(const Duration(milliseconds: 1500));
38          emit(state.copyWith(loading: PhonesLoading.loaded, phones: list));
39        } catch (e) {
40          emit(state.copyWith(loading: PhonesLoading.error));
41        }
42      }
```

Рисунок 3.8. Клас PhonesBloc з основною логікою подій блоку.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

3.2 Розробка Front-End частини додатку та поєднання із back-end

Розробка інтерфейсу користувача, на відміну від логіки додатку, яка пишеться мовою програмування Dart, пишеться на Flutter code – це особлива частина платформи Flutter, яка призначена для побудови фронтової частини додатку. Вона має свій власний рендерний двигун, свої власні форми та «Widget», за якими і будується додаток.

Побудова відбувається з використанням принципу розділення великих віджетів на менші із виносом їх у окремі файли – це основний патерн проектування інтерфейсу Flutter. Окремі сторінки, для їх коректного відображення на екрані пристрою, повинні писатися за допомогою StatelessWidget або StatefulWidget, від якого створений клас повинен унаслідуватися та використати поліформізм, для перевизначення функціоналу методу «build».

Домашня сторінка, а також сторінка із логіном та реєстрацією побудована за допомогою StatefulWidget – саме унаслідування від цього класу дозволяє створити тимчасовий state – стан сторінки, перевизначити методи, які дозволяють перевизначити методи керування станом екрану та додати поля вводу даних, забезпечити перевіркою, одночасним оновленням даних, а також методом dispose забезпечити очищення оперативної пам’яті, що забезпечить швидшу роботу додатку (рис. 3.9).

Сторінки, які представляють список наявних гаджетів, незалежно від їх типу, уже повинні будуватися через StatelessWidget, тимчасовий стан тут не потрібен, а навпаки – у побудові повинен використовуватися state із блоків, які були розроблені вище, у back-end частині додатку.

Рисунок 3.9. Побудова сторінки реєстрації через StatefulWidget.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Для прив'язування логіки додатку до інтерфейсу та моментального оновлення візуальної частини – використовується BlocProvider та BlocBuilder у параметри якого і передається певний йому блок. Під стан блоку – це або завантаження, або помилка, або коли завантаження пройшло успішно – на екран користувача виводяться відповідні дані. Під час завантаження – користувач буде бачити анімацію завантаження та відповідний надпис. При виникненні помилки – користувач буде повідомлений про неї і відповідно показаний список гаджетів (рис. 3.10).

Рисунок 3.10. Побудова сторінки гаджетів через BlocProvider.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Для представлення окремого гаджета - використовується GridBuilder – метод, який будує список із «плиток», які всередині себе мають певну інформацію про гаджет, а саме: назву моделі, ціну та стан пристрою. Для відстежування tap – подія, яка відстежується чи користувач натиснув на плитку, використовується GestureDetector, у який загортається повністю начинка всією плитки і у випадку натиснення – спрацьовує навігація, яка переадресовує користувача на сторінку із деталізованою інформацією про гаджет (рис. 3.11).

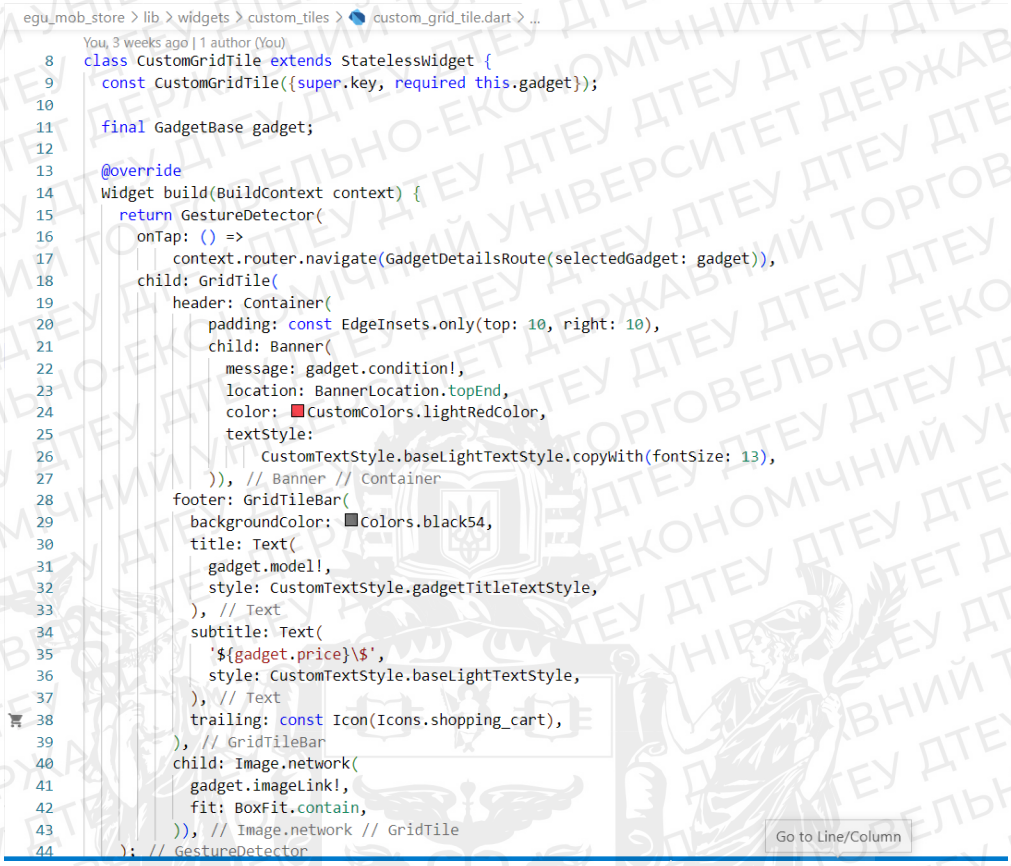
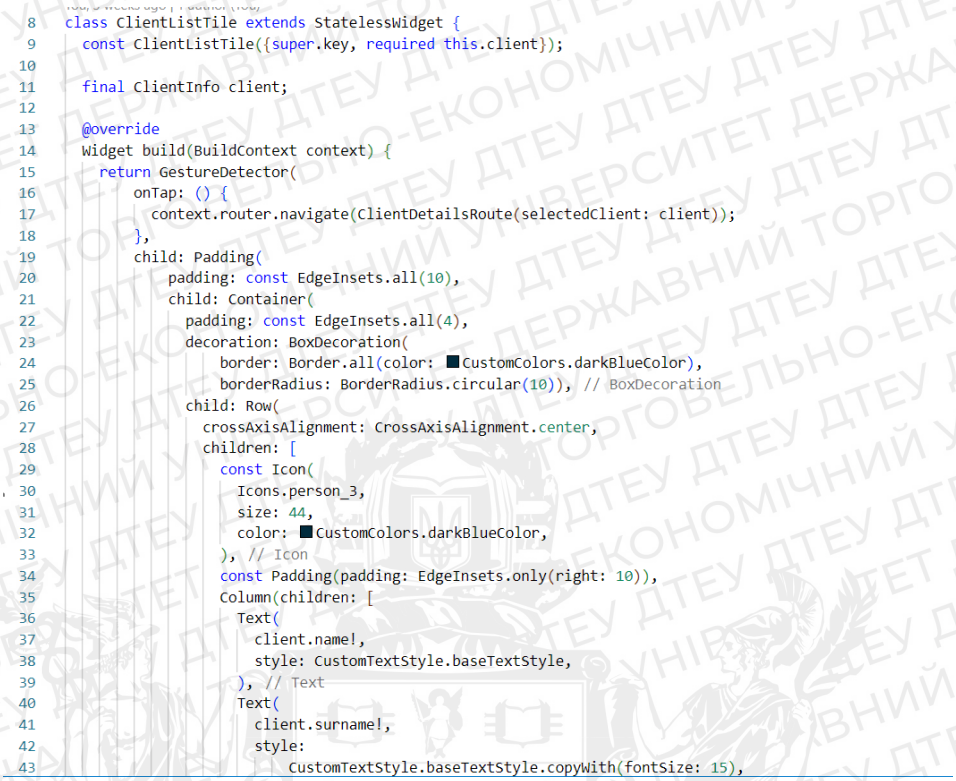


Рисунок 3.11. Побудова списку плиток із інформацією гаджета.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Побудова сторінки із збереженими клієнтами та їх даними відбувається за таким ж чином, як і сторінка з гаджетами – за допомогою bloc state та BlocProvider. Тільки відрізняється візуальною частиною, а саме – використовується ListBuilder, що відрізняється від GridBuilder тим, що будує список, елементи якого розтягуються на весь екран і всі дані поміщуються в один рядок (рис. 3.12).



```dart
class ClientListTile extends StatelessWidget {
  const ClientListTile({super.key, required this.client});

  final ClientInfo client;

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
        onTap: () {
          context.router.navigate(ClientDetailsRoute(selectedClient: client));
        },
        child: Padding(
            padding: const EdgeInsets.all(10),
            child: Container(
              padding: const EdgeInsets.all(4),
              decoration: BoxDecoration(
                  border: Border.all(color: CustomColors.darkBlueColor),
                  borderRadius: BorderRadius.circular(10)), // BoxDecoration
              child: Row(
                crossAxisAlignment: CrossAxisAlignment.center,
                children: [
                  const Icon(
                    Icons.person_3,
                    size: 44,
                    color: CustomColors.darkBlueColor,
                  ), // Icon
                  const Padding(padding: EdgeInsets.only(right: 10)),
                  Column(children: [
                    Text(
                      client.name!,
                      style: CustomTextStyle.baseTextStyle,
                    ), // Text
                    Text(
                      client.surname!,
                      style:
                          CustomTextStyle.baseTextStyle.copyWith(fontSize: 15),
```

Рисунок 3.12. Побудова списку із інформацією клієнта.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

Задля того, щоб подальша розробка додатку була можливо і швидка, а головне – безпроблемна, з підтримкою принципу Clean Architecture та Clean Code, а додаток та всі його сторінки були схожі між собою - базові стилі, кольори додатку, кнопки, які часто використовуються та інші віджети були окремо винесені в структуру проекту та розбито по певним текам. Також, дії, які повинні підтверджувати дію клієнта, наприклад видалення клієнта із списку, додаток повинен ще раз запитати користувача про правильність його дій. Це потрібно для зменшення ризиків випадкових дій користувача, які призведуть до втрати даних. Для цього використовується елемент Alert, який з’являється на екрані, як вікно підтвердження із двома кнопками. На прикладі показано одночасно і структуру стилів, які винесені, в тому числі стилі на текст, кнопки, snackbars, анімацію завантаження і тд, а також реалізацію модального вікна із підтвердженням дії (рис. 3.13).

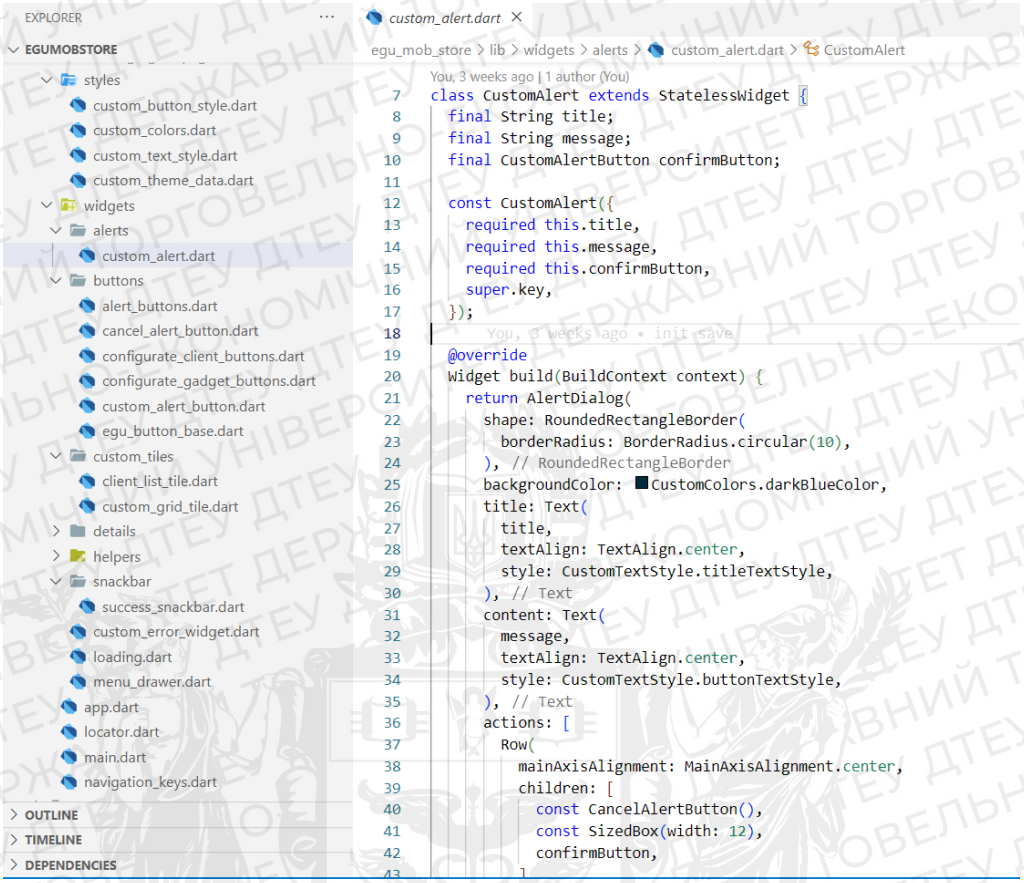


Рисунок 3.13. Побудова модального вікна та узагальнення стилізації.

Джерело: побудовано автором у системі Visual Studio Code (знімок з екрану)

3.3 Висновки до розділу 3

Розробка програмного забезпечення була розбита на дві частини – окремо логіку – back-end та окремо візуальну частину – front-end. Це дозволило використати принцип розділення коду, а також дотриматися архітектури bloc та Clean Architecture – основна мета якого це відокремлення логіки додатку від його інтерфейсу, задля забезпечення надійності додатку та можливості розширення, вдосконалення та швидкого використання без використання надлишкових ресурсів фізичного пристрою.

Додатково вирішено та розглянуто проблему важливості захисту персональних даних користувача від несанкціонованого доступу третіх осіб або стороннього програмного забезпечення, за допомогою хешування паролю з криптографічною функцією.

У розробці back-end частині додатку застосовано основні принципи об’єктно-орієнтованого програмування, потрібні патерни проектування, що підвищує надійність роботи програмного забезпечення , адже унеможливлює появи дій, що могло б викликати зависання додатку або його збій чи втрату даних.

У розробці front-end частини застосовано безліч принципів User Experience – присутня перевірка усіх полів на правильність даних, при очікуванні користувача додано анімацію завантаження, а також повідомлення про помилки, при некоректних діях, тощо. UI частина, написана на Flutter, пов’язана із логікою додатку, що моментально оновлює дані на екрані та виконує основну логіку, прив’язану до дії.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

У результаті виконання даної випускної кваліфікаційної роботи детально визначено та досліджено, проведено аналіз основних проблем, котрі виникають зараз у розробників програмного забезпечення, в тому числі і в незалежності від розроблюваної платформи, а також основні способи їх вирішення, такі як застосування принципів об'єктно-орієнтованого програмування та застосування патернів проектування і загальноприйнятих структур побудови архітектури програмного забезпечення. Правильне та обмірковане застосування даних варіантів вирішення проблем дозволяє з легкістю зробити додаток надійним, забезпечити його легким для розширення та легко підтримувати додаток після розробки.

При дослідженні проблем розробки розроблено основні вимоги до розробленого програмного додатку, від абстрактних понять розробки до конкретного використання певних інструментів.

У ході роботи було ретельно досліджено предметну область із керування інформаційною системою підприємства із торгівлі вживаними електронними гаджетами. Додатково проаналізовано основні проблеми, які виникають у даної предметної області та було представлено спосіб їх загального вирішення у вигляді програмного забезпечення для мобільних пристроїв – «EGUMobStore». Вирішення підкріплені додатково UML-діаграмами, такі як фізична модель бази даних, діаграмою класів, тощо - для додаткового аналізу і пошуку найкращого варіанту вирішення певних проблем предметної області. Для конфіденційних даних користувача додатково представлено фізичну

модель бази даних, її форму нормалізації та обгрунтування, що забезпечує правильне користування даними та їх захищеність і надійність. У ході роботи проаналізовано статистику використання інструментів із розробки, проаналізовано плюси та мінуси вибраних інструментів та встановлених додаткових фреймворків, плагінів та пакетів, основними перевагами яких є надійність, швидкість та крос-платформеність.

Для вирішення проблем із візуальною частиною додатку – розроблено макет інтерфейсу у системі Figma, для того, щоб додаток був зрозумілим і мав не надлишковий інтуїтивний інтерфейс, і для визначення кращої компоновки частин, і вирішення проблем, коли користувач може втратити дані або через неправильне зрозуміння функціоналу, спричинити інші проблеми - застосовано основні принципи UX – user experience, для зручного користування мобільним додатком. Додаток додатково розділений на дві частини – окремо логічна (серверна) частина додатку і окремо візуальна частина.

Додаток розроблений у середовищі Flutter та мовою програмування Dart. Побудований на архітектурі – Bloc - дане рішення вважається світовим стандартом серед мобільних розробників та користується найбільшим попитом. Рішення вирішує всі вище згадані проблеми, що виникають при розробці, а також при використанні користувачами – надійно поєднує інтерфейс додатку та серверну частину, забезпечує швидкісне керування станами додатку. Використані основні принципи об’єктно-орієнтованого програмування та використані деякі із патернів проектування для підвищення надійності додатку, такі як Singleton (Одинак) та інші. Протягом розробки у даній роботі продемонстровано знання та основні навички з роботою із мовою програмування Dart та середовищем Flutter.

Пропозиції до проєкту:

- програмний додаток може бути з легкістю покращений і удосконалений широтою зберігання асортименту, а саме містити

- більше типів і видів електронних гаджетів, може бути доповнений списком виробників, моделей або ж інших потрібних параметрів;
- програмний додаток може бути доповнений можливістю збереження конфіденційних даних користувача на одні із найпоширеніших хмарних сервісів.

Завдання даної роботи виконано: створено програмний додаток для мобільних пристроїв для керування інформаційною системою підприємства із торгівлі вживаних електронних гаджетів мовою програмування Dart за допомогою комплексу розробки Flutter. Розроблений програмний додаток успішно функціонує на телефонах, а використання поданого програмного забезпечення доступно будь-якому користувачеві.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The List of Object-oriented Programming Languages. [Електронний ресурс]. Режим доступу: https://www.orientsoftware.com/blog/list-of-object-oriented-programming-languages (дата звернення 16.01.2023).
2. What is Object-Oriented Programming [Електронний ресурс]. Режим доступу: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP#:~:text=Object%2Doriented%20programming%20(OOP)%20is%20a%20computer%20programming%20model,has%20unique%20attributes%20and%20behavior (дата звернення 16.01.2023).
3. What is object-oriented programming. OOP explained in depth [Електронний ресурс]. Режим доступу: https://www.educative.io/blog/object-oriented-programming (дата звернення 19.01.2023).
4. Four basic principles of Object- Oriented Programming [Електронний ресурс]. Режим доступу: https://lilly021.com/four-basic-principles-of-object-oriented-programming/ (дата звернення 21.01.2023).
5. 5 Best Languages for App Development – Axon [Електронний ресурс]. Режим доступу: https://www.axon.dev/blog/5-best-languages-for-app-development (дата звернення 22.01.2023).
6. Flutter: What is Dart Programming – Javatpoint [Електронний ресурс]. Режим доступу: https://www.javatpoint.com/flutter-dart-programming (дата звернення 22.01.2023).

7. Stack Overflow Developer Survey 2022 [Електронний ресурс]. Режим доступу: https://survey.stackoverflow.co/2022/#most-popular-technologies-misc-tech (дата звернення 21.01.2023

8. Top 8 Flutter Advantages - Relevant Software [Електронний ресурс]. Режим доступу: https://relevant.software/blog/top-8-flutter-advantages-and-why-you-should-try-flutter-on-your-next-project/ (дата звернення 23.01.2023).

9. Design Patterns – SourceMaking [Електронний ресурс]. Режим доступу: https://sourcemaking.com/design_patterns (дата звернення 23.01.2023).

10. Шаблони проектування. Початок. - Travels & Code [Електронний ресурс]. Режим доступу: https://travelscode.com/shabloni-proektuvannya-pochatok/ (дата звернення 23.01.2023).

11. MVC, MVP and MVVM Design Pattern – Medium [Електронний ресурс]. Режим доступу: https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm-design-pattern-6e169567bbad (дата звернення 24.01.2023).

12. How to Implement the BLoC Architecture in Flutter – Mobindustry [Електронний ресурс]. Режим доступу: https://www.mobindustry.net/blog/how-to-implement-the-bloc-architecture-in-flutter-benefits-and-best-practices/ (дата звернення 25.01.2023).

13. bloc | Dart Package [Електронний ресурс]. Режим доступу: https://pub.dev/packages/bloc (дата звернення 25.01.2023).

ДОДАТКИ

ДОДАТОК А

Код класу EGUMobApp

```
import 'package:auto_route/auto_route.dart';
import 'package:egu_mob_store/routes/egu_observer.dart';
import 'package:egu_mob_store/routes/egu_router.gr.dart';
import 'package:egu_mob_store/styles/custom_theme_data.dart';
import 'package:flutter/material.dart';

class EGUMobApp extends StatelessWidget {
 final _appRouter = EguRouter();

 EGUMobApp({super.key});

 @override
 Widget build(BuildContext context) {
  return MaterialApp.router(
   theme: CustomThemeData().defaultTheme,
   debugShowCheckedModeBanner: false,
   title: 'EGUMobStore',
   routerDelegate: AutoRouterDelegate(
    _appRouter,
    navigatorObservers: () => [EguObserver()],
   ),
   routeInformationParser: _appRouter.defaultRouteParser(),
  );
 }
}
```

Код класу HomePage

```
import 'package:egu_mob_store/styles/custom_button_style.dart';
import 'package:egu_mob_store/styles/custom_text_style.dart';
import 'package:egu_mob_store/widgets/helpers/widget_helpers.dart';
import 'package:egu_mob_store/widgets/menu_drawer.dart';
import 'package:flutter/material.dart';

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

  @override
  // ignore: library_private_types_in_public_api
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  final GlobalKey<ScaffoldState> _key = GlobalKey();
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      key: _key,
      appBar: WidgetHelper.getAppBar(context, 'EGU Mob Store'),
      drawer: const MenuDrawer(),
      body: Center(
        child: Column(
       mainAxisAlignment: MainAxisAlignment.center,
       children: [
         Text(
          'Hello User !',
          textAlign: TextAlign.center,
          style: CustomTextStyle.titleTextStyle
              .copyWith(fontWeight: FontWeight.w900, fontSize: 42),
         ),
         const Padding(padding: EdgeInsets.all(5)),
         const Text('Get started by adding goods to categories',
             textAlign: TextAlign.center,
             style: CustomTextStyle.baseTextStyle),
         const Padding(padding: EdgeInsets.all(5)),
         const Text('You can manage your goods here by a lot of ways',
             textAlign: TextAlign.center,
             style: CustomTextStyle.baseTextStyle),
         const Padding(padding: EdgeInsets.all(5)),
         const Text('Select a category from menu in app bar',
             textAlign: TextAlign.center,
             style: CustomTextStyle.baseTextStyle),
         const Padding(padding: EdgeInsets.all(15)),
         ElevatedButton(
          onPressed: () {
```

```
            _key.currentState!.openDrawer(); //<-- SEE HERE
          },
          style: CustomButtonStyle.baseButtonStyle,
          child: Padding(
            padding: const EdgeInsets.all(15),
            child: SizedBox(
              width: 180,
              child: Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: const [
                  Text('Click here to start',
                      textAlign: TextAlign.center,
                      style: CustomTextStyle.buttonTextStyle),
                  Padding(padding: EdgeInsets.only(right: 10)),
                  Icon(Icons.sell)
                ],
              ),
            )),
        )
      ],
    )));
  }
}
```

Код класу GadgetBase

```
part of gadget_models;

abstract class GadgetBase {
  final String? id;
  final String? producer;
  final String? model;
  final String? imageLink;
  final int? price;
  final String? condition;
  final String? color;
  final int? year;
  final int? batteryCapacity;
  final String? os;
  final String? ram;
  final String? processor;
  final String? screenSize;
  final String? memory;
  final String? gadgetType;
  final bool isSelected;

  const GadgetBase({
    this.id,
    this.producer,
    this.model,
    this.imageLink,
    this.price,
    this.condition,
    this.color,
    this.year,
    this.batteryCapacity,
    this.os,
    this.ram,
    this.processor,
    this.screenSize,
    this.memory,
    this.gadgetType,
    this.isSelected = false,
  });

  Map<String, dynamic> toJson();
  factory GadgetBase.fromJson(Map<String, dynamic> json) {
    // implement GadgetBase.fromJson
    throw UnimplementedError();
  }
}
```

Код класу Phone

```
part of gadget_models;

@CopyWith()
class Phone extends GadgetBase {
 final String? cameraResolution;
 // ignore: annotate_overrides, overridden_fields
 final String? gadgetType;
 const Phone(
    {String? id,
    String? producer,
    String? model,
    String? imageLink,
    int? year,
    String? os,
    String? ram,
    String? processor,
    String? screenSize,
    String? memory,
    int? batteryCapacity,
    int? price,
    String? condition,
    bool isSelected = false,
    this.cameraResolution,
    this.gadgetType = GadgetType.phone})
    : super(
        id: id,
        producer: producer,
        model: model,
        imageLink: imageLink,
        year: year,
        os: os,
        ram: ram,
        processor: processor,
        screenSize: screenSize,
        memory: memory,
        batteryCapacity: batteryCapacity,
        price: price,
        condition: condition,
        gadgetType: gadgetType,
        isSelected: isSelected);

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

 @override
 factory Phone.fromJson(Map<String, dynamic> json) => Phone(
     id: json['id'] as String?,
```

```
    producer: json['producer'] as String?,
    model: json['model'] as String?,
    imageLink: json['imageLink'] as String?,
    year: json['year'] as int?,
    os: json['os'] as String?,
    ram: json['ram'] as String?,
    processor: json['processor'] as String?,
    screenSize: json['screenSize'] as String?,
    memory: json['memory'] as String?,
    batteryCapacity: json['batteryCapacity'] as int?,
    price: json['price'] as int?,
    condition: json['condition'] as String?,
    isSelected: json['isSelected'] ?? false,
    cameraResolution: json['cameraResolution'] as String?,
    gadgetType: json['gadgetType'] as String? ?? GadgetType.phone,
  );

Map<String, dynamic> _$PhoneToJson(Phone instance) => <String, dynamic>{
    'id': instance.id,
    'producer': instance.producer,
    'model': instance.model,
    'imageLink': instance.imageLink,
    'price': instance.price,
    'condition': instance.condition,
    'year': instance.year,
    'batteryCapacity': instance.batteryCapacity,
    'os': instance.os,
    'ram': instance.ram,
    'processor': instance.processor,
    'screenSize': instance.screenSize,
    'memory': instance.memory,
    'cameraResolution': instance.cameraResolution,
    'gadgetType': instance.gadgetType,
  };

List<Object?> get props => [
    producer,
    model,
    year,
    os,
    ram,
    processor,
    screenSize,
    memory,
    BatteryHelper.getBattery(batteryCapacity!),
    PriceHelper.getPrice(price!),
    condition,
    cameraResolution,
  ];
```

Код класу DatabaseHelper

```
// ignore_for_file: depend_on_referenced_packages

import 'package:egu_mob_store/models/enums/gadget_type.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

import '../gadget_models.dart';

class DatabaseHelper {
  static Database? _database;

  static const String gadgetTable = 'gadgets';
  static const String clientsTable = 'clients';
  static const String orderTable = 'orders';
  static const String credentialsTable = 'credentials';
  static const String soldGadgetsTable = 'soldGadgets';

  static const String columnId = 'id';
  static const String columnProducer = 'producer';
  static const String columnModel = 'model';
  static const String columnImageLink = 'imageLink';
  static const String columnPrice = 'price';
  static const String columnCondition = 'condition';
  static const String columnColor = 'color';
  static const String columnYear = 'year';
  static const String columnBatteryCapacity = 'batteryCapacity';
  static const String columnOS = 'os';
  static const String columnRAM = 'ram';
  static const String columnProcessor = 'processor';
  static const String columnScreenSize = 'screenSize';
  static const String columnMemory = 'memory';
  static const String columnCameraResolution = 'cameraResolution';
  static const String columnGPU = 'gpu';
  static const String columnIsEkg = 'isEkg';
  static const String columnIsWaterResist = 'isWaterResist';
  static const String columnIsTemperature = 'isTemperature';
  static const String columnIsSos = 'isSos';
  static const String columnIsGps = 'isGps';
  static const String columnIsHasMicro = 'isHasMicro';
  static const String columnSize = 'size';
  static const String columnIsBluetooth = 'isBluetooth';
  static const String columnName = 'name';
  static const String columnSurname = 'surname';
  static const String columnPhoneNumber = 'phoneNumber';
  static const String columnAdress = 'adress';
  static const String columnPriceSum = 'priceSum';
  static const String columnBoughtGadgets = 'boughtGadgets';
  static const String columnClientInfo = 'userInfo';
```

```
  static const String gadgetType = 'gadgetType';
  static const String columnCredentialEmail = 'email';
  static const String columnCredentialUsername = 'username';
  static const String columnCredentialHash = 'hashPassword';
  static const String columnDate = 'date';

  static Future<Database> get database async {
    if (_database != null) return _database!;

    _database = await initDatabase();
    return _database!;
  }

  static Future<Database> initDatabase() async {
    final databasePath = await getDatabasesPath();
    final path = join(databasePath, 'my_database.db');

    return await openDatabase(path, version: 1, onCreate: _createDatabase);
  }

  static Future<void> _createDatabase(Database db, int version) async {
    await db.execute('''
      CREATE TABLE $gadgetTable (
        $columnId TEXT PRIMARY KEY,
        $columnProducer TEXT,
        $columnModel TEXT,
        $columnImageLink TEXT,
        $columnPrice INTEGER,
        $columnCondition TEXT,
        $columnColor TEXT,
        $columnYear INTEGER,
        $columnBatteryCapacity INTEGER,
        $columnOS TEXT,
        $columnRAM TEXT,
        $columnProcessor TEXT,
        $columnScreenSize TEXT,
        $columnMemory TEXT,
        $columnCameraResolution TEXT,
        $columnGPU TEXT,
        $columnIsEkg INTEGER,
$columnIsWaterResist INTEGER,
    $columnIsTemperature INTEGER,
    $columnIsSos INTEGER,
    $columnIsGps INTEGER,
    $columnIsHasMicro TEXT,
    $columnSize TEXT,
    $columnIsBluetooth TEXT,
    $gadgetType TEXT
```

```
    )
  ''');
    await db.execute('''

    await db.execute('''
  CREATE TABLE $clientsTable (
    $columnId TEXT PRIMARY KEY,
    $columnName TEXT,
    $columnSurname TEXT,
    $columnPhoneNumber TEXT,
    $columnAdress TEXT
  )
  ''');

    await db.execute('''
  CREATE TABLE $orderTable (
    $columnId TEXT PRIMARY KEY,
    $columnPriceSum INTEGER,
    $columnBoughtGadgets TEXT,
    $columnClientInfo TEXT,
    $columnDate Text
  )
  ''');
    await db.execute('''
  CREATE TABLE $credentialsTable (
    $columnId TEXT PRIMARY KEY,
    $columnCredentialEmail TEXT,
    $columnCredentialUsername TEXT,
    $columnCredentialHash TEXT
  )
  ''');
  }

  static Future<int> insertGadget(GadgetBase gadget) async {
    final db = await database;
    return await db.insert(gadgetTable, gadget.toJson());
  }

  static Future<int> insertSoldGadget(GadgetBase gadget) async {
    final db = await database;
    return await db.insert(soldGadgetsTable, gadget.toJson());
  }

  static Future<List<Phone>> getAllPhones() async {
    final db = await database;
    final gadgetsMapList = await db.query(gadgetTable);
    return List.generate(
        gadgetsMapList.where((e) => e['gadgetType'] == GadgetType.phone).length,
```

```
      (i) {
    return Phone.fromJson(gadgetsMapList[i]);
  });
}

static Future<List<MacBook>> getAllMacbooks() async {
  final db = await database;
  final gadgetsMapList = await db.query(gadgetTable);
  return List.generate(
      gadgetsMapList
          .where((e) => e['gadgetType'] == GadgetType.macbook)
          .length, (i) {
    return MacBook.fromJson(gadgetsMapList[i]);
  });
}

static Future<List<Headphone>> getAllHeadphones() async {
  final db = await database;
  final gadgetsMapList = await db.query(gadgetTable);
  return List.generate(
      gadgetsMapList
          .where((e) => e['gadgetType'] == GadgetType.headphone)
          .length, (i) {
    return Headphone.fromJson(gadgetsMapList[i]);
  });
}

static Future<List<Watch>> getAllWatches() async {
  final db = await database;
  final gadgetsMapList = await db.query(gadgetTable);
  return List.generate(
      gadgetsMapList.where((e) => e['gadgetType'] == GadgetType.watch).length,
      (i) {
    return Watch.fromJson(gadgetsMapList[i]);
  });
}

static Future<GadgetBase?> getGadget(String id, bool isSold) async {
  final db = await database;
  final gadgetsMapList = await db.query(
      isSold ? soldGadgetsTable : gadgetTable,
      where: '$columnId = ?',
      whereArgs: [id]);
  if (gadgetsMapList.isNotEmpty) {
    switch (gadgetsMapList.first['gadgetType']) {
      case GadgetType.phone:
        return Phone.fromJson(gadgetsMapList.first);
      case GadgetType.macbook:
```

```
        return MacBook.fromJson(gadgetsMapList.first);
      case GadgetType.headphone:
        return Headphone.fromJson(gadgetsMapList.first);
      case GadgetType.watch:
        return Watch.fromJson(gadgetsMapList.first);
    }
  }
  return null;
}

static Future<int> updateGadget(GadgetBase gadget) async {
  final db = await database;
  return await db.update(gadgetTable, gadget.toJson(),
      where: '$columnId = ?', whereArgs: [gadget.id]);
}

static Future<int> deleteGadget(String id) async {
  final db = await database;
  return await db
      .delete(gadgetTable, where: '$columnId = ?', whereArgs: [id]);
}

static Future<int> insertClient(ClientInfo user) async {
  final db = await database;
  return await db.insert(clientsTable, user.toJson());
}

static Future<List<ClientInfo>> getAllClients() async {
  final db = await database;
  final usersMapList = await db.query(clientsTable);
  return List.generate(usersMapList.length, (i) {
    return ClientInfo.fromJson(usersMapList[i]);
  });
}

static Future<ClientInfo?> getClientById(String id) async {
  final db = await database;
  final usersMapList =
      await db.query(clientsTable, where: '$columnId = ?', whereArgs: [id]);
  if (usersMapList.isNotEmpty) {
    return ClientInfo.fromJson(usersMapList.first);
  }
  return null;
}

static Future<int> updateClient(ClientInfo user) async {
  final db = await database;
  return await db.update(clientsTable, user.toJson(),
```

```
      where: '$columnId = ?', whereArgs: [user.id]);
}

static Future<int> deleteClient(String id) async {
  final db = await database;
  return await db
      .delete(clientsTable, where: '$columnId = ?', whereArgs: [id]);
}

static Future<int> insertOrder(OrderInfo order) async {
  final db = await database;
  return await db.insert(orderTable, order.toJson());
}

static Future<List<OrderInfo>> getAllOrders() async {
  final db = await database;
  final ordersMapList = await db.query(orderTable);
  return List.generate(ordersMapList.length, (i) {
    return OrderInfo.fromJson(ordersMapList[i]);
  });
}

static Future<OrderInfo?> getOrder(String id) async {
  final db = await database;
  final ordersMapList =
      await db.query(orderTable, where: '$columnId = ?', whereArgs: [id]);
  if (ordersMapList.isNotEmpty) {
    return OrderInfo.fromJson(ordersMapList.first);
  }
  return null;
}

static Future<int> updateOrder(OrderInfo order) async {
  final db = await database;
  return await db.update(orderTable, order.toJson(),
      where: '$columnId = ?', whereArgs: [order.id]);
}

static Future<int> deleteOrder(String id) async {
  final db = await database;
  return await db.delete(orderTable, where: '$columnId = ?', whereArgs: [id]);
}

static Future<int> insertCredentials(LoginInfo login) async {
  final db = await database;
  return await db.insert(credentialsTable, login.toJson());
}
```

```
static Future<LoginInfo?> getCredentialsFromEmail(String email) async {
  final db = await database;
  final usersMapList = await db.query(credentialsTable,
      where: '$columnCredentialEmail = ?', whereArgs: [email]);
  if (usersMapList.isNotEmpty) {
    return LoginInfo.fromJson(usersMapList.first);
  }
  return null;
}
```

Код класу PhonesBloc

```
import 'dart:async';

import 'package:egu_mob_store/models/database/database_helper.dart';
import 'package:egu_mob_store/models/gadget_models.dart';
import 'package:egu_mob_store/models/helpers/id_helper.dart';
import 'package:egu_mob_store/models/helpers/image_helper.dart';
import 'package:egu_mob_store/models/helpers/price_helper.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

// ignore: depend_on_referenced_packages
import 'package:freezed_annotation/freezed_annotation.dart';

part 'phones_event.dart';

part 'phones_state.dart';

part 'phones_bloc.freezed.dart';

class PhonesBloc extends Bloc<PhonesEvent, PhonesState> {
  PhonesBloc() : super(const PhonesState()) {
    on<LoadPhones>(_loadPhones);
    on<DeletePhone>(_deletePhone);
    on<EditPhone>(_editPhone);
    on<AddPhone>(_addPhone);
    on<SellPhones>(_sellPhones);
    on<SelectPhone>(_selectPhone);
    on<SearchPhone>(_searchPhone);
  }

  FutureOr<void> _loadPhones(
      LoadPhones event, Emitter<PhonesState> emit) async {
    try {
      emit(state.copyWith(loading: PhonesLoading.loading));

      var allGadgets = await DatabaseHelper.getAllPhones();
      var list = <Phone>[];
      for (var element in allGadgets) {
        list.add(element.copyWith(
            imageLink: ImageHelper.getPhoneImage(element.model!)));
      }
      await Future.delayed(const Duration(milliseconds: 1500));
      emit(state.copyWith(loading: PhonesLoading.loaded, phones: list));
    } catch (e) {
      emit(state.copyWith(loading: PhonesLoading.error));
    }
  }
```

```
FutureOr<void> _deletePhone(
  DeletePhone event, Emitter<PhonesState> emit) async {
 try {
  await DatabaseHelper.deleteGadget(event.phone.id!);
 } catch (e) {
  emit(state.copyWith(loading: PhonesLoading.error));
 }
}

FutureOr<void> _editPhone(EditPhone event, Emitter<PhonesState> emit) async {
 await DatabaseHelper.updateGadget(event.phone);
}

FutureOr<void> _addPhone(AddPhone event, Emitter<PhonesState> emit) async {
 await DatabaseHelper.insertGadget(event.phone);
}

FutureOr<void> _sellPhones(
  SellPhones event, Emitter<PhonesState> emit) async {
 await DatabaseHelper.insertOrder(OrderInfo(
    id: IdHelper.getRandomId(),
    priceSum: PriceHelper.getSumPrice(event.phones),
    boughtGadgets: event.phones.map((phone) => phone.id!).toList(),
    userInfo: event.client.id,
    date: DateTime.now().toString().substring(0, 10)));
 for (var phone in event.phones) {
  await DatabaseHelper.deleteGadget(phone.id!);
 }
 for (var phone in event.phones) {
  await DatabaseHelper.insertSoldGadget(phone);
 }
}

FutureOr<void> _selectPhone(SelectPhone event, Emitter<PhonesState> emit) {
 final List<Phone> updatedPhones = [];

 for (final phone in state.phones) {
  if (phone.id == event.phone.id) {
   if (phone.isSelected == true) {
    updatedPhones.add(phone.copyWith(isSelected: false));
   } else {
    updatedPhones.add(phone.copyWith(isSelected: true));
   }
  }
 emit(state.copyWith(phones: updatedPhones));
```

Код класу Loading

```
import 'package:egu_mob_store/styles/custom_colors.dart';
import 'package:egu_mob_store/styles/custom_text_style.dart';
import 'package:flutter/material.dart';
import 'package:flutter_spinkit/flutter_spinkit.dart';

class Loading extends StatelessWidget {
  const Loading({super.key, required this.text});

  final String text;
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          const SpinKitDoubleBounce(
            duration: Duration(milliseconds: 2400),
            color: CustomColors.lightRedColor,
            size: 122,
          ),
          const Padding(padding: EdgeInsets.only(bottom: 15)),
          Text(
            'Loading $text\nPlease, wait',
            textAlign: TextAlign.center,
            style: CustomTextStyle.titleTextStyle.copyWith(
                color: CustomColors.darkBlueColor, fontWeight: FontWeight.w600),
          )
        ],
      ),
    );
  }
}
```

Код класу RegisterPage

```
// ignore_for_file: use_build_context_synchronously

import 'package:auto_route/auto_route.dart';
import 'package:egu_mob_store/models/database/database_helper.dart';
import 'package:egu_mob_store/models/gadget_models.dart';
import 'package:egu_mob_store/models/helpers/id_helper.dart';
import 'package:egu_mob_store/models/helpers/password_hasher.dart';
import 'package:egu_mob_store/routes/egu_router.gr.dart';
import 'package:egu_mob_store/styles/custom_colors.dart';
import 'package:egu_mob_store/styles/custom_text_style.dart';
import 'package:egu_mob_store/widgets/helpers/widget_helpers.dart';
import 'package:egu_mob_store/widgets/snackbar/success_snackbar.dart';
import 'package:flutter/material.dart';

import '../../widgets/buttons/egu_button_base.dart';

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

  @override
  // ignore: library_private_types_in_public_api
  _RegisterPageState createState() => _RegisterPageState();
}

class _RegisterPageState extends State<RegisterPage> {
  final _formKey = GlobalKey<FormState>();
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();

  final TextEditingController _userNameController = TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();

    _userNameController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: WidgetHelper.getAppBar(context, 'Register page'),
        body: Center(
          child: Form(
```

```
key: _formKey,
child: Column(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
  const Icon(
   Icons.add_business_outlined,
   size: 95,
   color: CustomColors.lightMainColor,
  ),
  Padding(
   padding: const EdgeInsets.all(15),
   child: TextFormField(
    controller: _emailController,
    validator: (value) {
     if (value == null || value.isEmpty) {
      return 'Please enter an email';
     }
     if (!RegExp(r'^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$')
        .hasMatch(value)) {
      return 'Please enter a valid email';
     }
     return null;
    },
    decoration: InputDecoration(
     labelStyle: CustomTextStyle.baseTextStyle,
     hintText: 'Email',
     border: const OutlineInputBorder(),
     focusedBorder: OutlineInputBorder(
      borderSide: const BorderSide(
          color: CustomColors.lightRedColor, width: 2.0),
      borderRadius: BorderRadius.circular(10.0),
     ),
    ),
   ),
  ),
  Padding(
   padding: const EdgeInsets.all(15),
   child: TextFormField(
    controller: _userNameController,
    validator: (value) {
     if (value == null || value.isEmpty) {
      return 'Please enter an username';
     }
     return null;
    },
    decoration: InputDecoration(
     labelStyle: CustomTextStyle.baseTextStyle,
     hintText: 'UserName',
```

```
      border: const OutlineInputBorder(),
      focusedBorder: OutlineInputBorder(
       borderSide: const BorderSide(
           color: CustomColors.lightRedColor, width: 2.0),
       borderRadius: BorderRadius.circular(10.0),
      ),
     ),
    ),
   ),
   Padding(
    padding: const EdgeInsets.all(15),
    child: TextFormField(
     controller: _passwordController,
     validator: (value) {
      if (value == null || value.isEmpty) {
       return 'Please enter a password';
      }
      if (value.length < 8) {
       return 'Password must be at least 8 characters long';
      }
      if (!RegExp(r'^(?=.*?[0-9])').hasMatch(value)) {
       return 'Password must contain at least one digit';
      }
      return null;
     },
     obscureText: true,
     decoration: InputDecoration(
      labelStyle: CustomTextStyle.baseTextStyle,
      hintText: 'Password',
      border: const OutlineInputBorder(),
      focusedBorder: OutlineInputBorder(
       borderSide: const BorderSide(
           color: CustomColors.lightRedColor, width: 2.0),
       borderRadius: BorderRadius.circular(10.0),
      ),
     ),
    ),
   ),
   EguButtonBase(
    onPressed: () async {
     if (_formKey.currentState!.validate()) {
      await _registerUser();
     }
    },
    icon: Icons.app_registration,
    textData: 'Register',
    width: 160,
   ),
```

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

Future<void> _registerUser() async {
  var loggedInfo =
      await DatabaseHelper.getCredentialsFromEmail(_emailController.text);
  if (loggedInfo != null) {
    ScaffoldMessenger.of(context).showSnackBar(
      CustomSnackBars.getErrorSnackBarWithCustomText(
          'This email is already registered'));
    return;
  }
  var loginInfo = LoginInfo(
      id: IdHelper.getRandomId(),
      email: _emailController.text,
      username: _userNameController.text,
      hashPassword: PasswordHasher.hashPassword(_passwordController.text));

  await DatabaseHelper.insertCredentials(loginInfo);
  context.router.navigate(const HomeRoute());
  ScaffoldMessenger.of(context).showSnackBar(
      CustomSnackBars.getSuccessSnackBarCustomText(
          'Successfully registered'));
```