МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-26

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.26.15.000.КРБ

СКЛЯРОВА ЛАДОНА СПАРТАКОВИЧА

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**
Освітня програма: **«Комп'ютерна інженерія»**
Група: **2БКС-26**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему:_____

«Автоматизація бізнес-процесів за допомогою мобільного додатка на Flutter»

_____

Проектний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах)

Виконавець _____ (Скляров Л.С)
Керівник проекту_____ (Іванова Л.В.)

Консультанти:
з охорони праці_____ (Чорновол Н.І.)
з дотримання вимог ЄСКД _____ (Петрашова В.І.)
старший консультант _____ (Скорнякова О.В.)

До захисту допущений
Завідувачка кафедри _____ (Іванова Л.В.)
Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 202____ р. Протокол ДКК №_____
Оцінка ЕК _____

Секретар ДКК _____

АНОТАЦІЯ

Тема дипломного проекту «Автоматизація бізнес-процесів за допомогою мобільного додатка на Flutter».

У цій роботі описані принципи розробки сучасного мобільного додатку та процес підбору інструментів та технологій для розробки.

У аналітичній частині приведене технічне завдання до проекту, а також аналіз мов розробки зі сфери кросплатформених рішень. В технологічній частині приведені інструменти для розробки мобільних додатків та обгрунтування вибіру конкретного рішення. У програмному розділі описана розробка UI/UX-інтерфейсу, налаштування бази даних Firebase, та побудова бізнес логіки додатку разом з Provider. Розділ тестування розповідає про роботу користувача з основними можливостями додатка. Останнім розділом стала охорона праці під час розробки та конструювання додатку.

Ключові слова: мобільний додаток, застосунок, розробка, Flutter, Dart, Provider, Firebase, UI/UX, Android, iOS, вебвигляд.

ANNOTATION

The topic of the diploma project is "Automation of business processes using a mobile application on Flutter".

This paper describes the principles of development of a modern mobile application and the process of selecting tools and technologies for development.

The analytical part presents the technical task for the project, as well as the analysis of development languages in the field of cross-platform solutions. The technological part provides tools for developing mobile applications and justification for choosing a specific solution. The software section describes the development of the UI / UX interface, setting up the Firebase database, and building the business logic of the application together with the Provider. The testing section tells about the user's work with the main features of the application. The last section was labor protection during the development and design of the application.

Keywords: mobile application, application, development, Flutter, Dart, Provider, Firebase, UI / UX, Android, iOS, web view.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Кафедра *комп'ютерної інженерії*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР ***Беркань І.В.***

"_____" _____202____р.

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра

Здобувачеві (здобувачці) освіти ***Скляров Ладон Спартакович***

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи **Автоматизація бізнес-процесів за допомогою мобільного додатка на Flutter**

затверджена наказом по коледжу від "30" грудня 2021р. № 306-А2-ОД

2. Термін здачі кваліфікаційної роботи 20.06.2022р.

3. Вихідні дані до роботи

1. *Технічне завдання*
2. *Програмне забезпечення для створення мобільних додатків*
3. *Вимоги до функціоналу та графічного дизайну;*
4. *Засоби тестування мобільних додатків*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Вступ. Аналітичній огляд засобів автоматизації бізнесу. Проектування мобільного додатку розробка мобільного додатку. Розділ охорони праці. Висновок. Перелік використаних джерел інформації.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

1. *Архітектура мобільного додатку*
2. *Алгоритм роботи мобільного додатку*
3. *Схема взаємодії контенту мобільного додатку*
4. *Графічний дизайн мобільного додатку*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів роботи, що стосується їх

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний	*Іванова Л.В.*	*30.11.2021*	*17.05.2022*
Охорона праці	*Чорновол В.І.*	*4.05.2022*	*17.05.2022*
Нормоконтроль	*Петрашова В.І.*	*4.05.2022*	*17.05.2022*
Старший консультант	*Скорнякова О.В.*	*4.05.2022*	*17.05.2022*

7. Дата видачі завдання *30.11.2021*

Керівник роботи *Іванова Л.В.*
(підпис)

Завдання прийняв до виконання *Скляров Л.С.*
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Передмова*	*4.05.2022*	
2.	*Аналітичній огляд засобів автоматизації бізнесу*	*8.05.2022*	
3.	*Визначення технічного завдання на роботу*	*10.05.2022*	
4.	*Проектування мобільного додатку*	*15.05.2022*	
5.	*Розробка мобільного додатку*	*17.05.2022*	
6.	*Розділ охорони праці*	*22.05.2022*	
7.	*Висновок*	*26.05.2022*	
8.	*Перелік літератури*	*28.05.2022*	
9.	*Оформлення пояснювальної записки*	*31.05.2022*	
10.	*Оформлення графічної частини*	*2.06.2022*	
11.	*Малий захист кваліфікаційної роботи*	*15.06.2022*	
12.	*Захист кваліфікаційної роботи*	*18.06.2022*	

Виконавець *Скляров Л.С.*
(підпис)

Керівник роботи *Іванова Л.В.*
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 26.15.000.00 КР	Кваліфікаційна робота		
А4			БКС 26.15.000.00 КР ПЗ	Пояснювальна записка	1	
			БКС 26.15.000.00 КР МП	Мультимедійна презентація		

Ізм.	Арк.	№ докум.	Підпис	Дата	БКС 26.15.000.00 БКР ПЗ		
Розробив		Скляров Л..С.			**Автоматизація бізнес-процесів за допомогою мобільного додатка на Flutter**	Літ.	Аркуш 7 / Аркушів 88
Перевірив		Іванова Л.В.				Н Д П	
Н. Контр.		Петрашова В.				ВСП «ОТФК ОНТУ» гр.БКС-26	
Затверд.		Скорнякова О.					

ЗМІСТ

ПЕРЕДМОВА

Підприємство, бізнес, стартап, не можуть інсувати без IT-доповнення свого бізнесу у будь-якому вигляді. За цим стає прогрес - теперішнє і майбутнє.

Ілон Маск каже - "Люди вже як кіборги і ми існуємо у двох світах, у справжньому та цифровому, просто ми цього не дуже помічаємо, так як швидкість отримання інформації у нашому телефоні, більше ніж її відправлення".

У книзі "From good to Great"(Від хорошого до великого) провели дослідження кампаній, які стають лідерами ринку та залишаються на цьому рівні більш ніж 15 років. Обрали не трендові низини, які у свої часи були навіть з кращими показниками, ніж у топових кампаній, таких як Coca-Cola. Окрім якостей лідера та працівників кампанії, вони підтвердили висновок - треба використовувати нові технології, та бути першими, хто буде відкривати їх на ринку. Тому, як каже український підприємець збудувавший місто-екосистему для айтішників в Україні UNIT.City, UNIT.Home - Василій Хмельницький "Бізнес без IT не може існувати, на такий бізнес я навіть не дивлюсь і не вкладаюсь".

Я вважаю, спробувавши різні сфери самодіяльності, я відмовився від розвитку без IT, ця індустрія та звя'зок звичайної низини з цифровими технологіями це обов'язково і саме перспективне у наш час.

Засновник Youtube mobile Андрій Доронічев, написав у одному з постів Інстаграм - "Сайти - минуле, мобільні додатки - теперішнє, VR/AR - майбутнє".

Україна вже показує це всьому всесвіту своїми мобільними додатками:

Дія - держава у смартфоні, додаток який зменшує документообіг, який навіть під час війни України допомагає тисячам українців перетнути кордони за допомогою цього застосунку.

Monobank - перший в країні необанк без відділень, банк у смартфоні. Цей додаток спростив використовування банківськими послугами та став головним банком в Україні.

У розвитку мобільних додатків, вже декілька років відкріли такий напрямок як кросплатформене програмування, коли застосунок може працювати с однією кодовою базою на різних операційних системах та пристроях. Що спрощує фінансування і швидкість розробки. Лідерами з мов програмування є React JS, Kotlin Multiplatform та мова про яку я буду розповідати у дипломній роботі Flutter SDK з мовою Dart.

Планую розробити додаток для автоматизації бізнес-процесів для магазину посуду, у якому можна буде переглядати каталог товарів та замовляти їх зворотним зв’язком або залишаючи повноцінне замовлення без зворотного зв’язку. Використовувати для бізнесу LY.Odessa для підтримання бренду, додаючи посилання у сторінці Instagram на застосунок. Щоб людям було зручніше зробити замовлення, без контактування клієнта в Instagram.

Додаток буде працювати у вебвигляді, як сайт для смартфону, але який буде адаптивним. У майбутньому буде працювати на IOS і Android, використовуючи отриману кодову базу.

1. АНАЛІТИЧНИЙ ОГЛЯД ЗАСОБИВ АВТОМАТИЗАЦІЇ БІЗНЕСУ

1.1 Додаток на Flutter SDK з мовою програмування Dart. Переваги автоматизації бізнесу за допомогою Flutter додатку над сайтом для бізнесу

Flutter — це комплект засобів розробки та фреймворк з відкритим вихідним кодом для створення мобільних додатків під Android та iOS, вебзастосунків, а також додатків під Windows, macOS та Linux з використанням мови програмування Dart, розроблений і розвивається корпорацією Google. Логотипи можно побачити на рисунку 1.1.

Рисунок 1.1 – Логотипи Flutter, Dart

Перша версія випущена у 2015 році під назвою Sky, працювала тільки для Android-додатків. Основна оголошена особливість – висока графічна продуктивність (можливість відображення 120 кадрів за секунду).

Повна підтримка створення вебпрограм з'явилася у версії 2.0, з цієї ж версії реалізована підтримка створення програм для Windows, macOS, Linux, Google Fuchsia. Поруч з цим, можливо переписати нове ядро Flutter і використати його на іншій платформі. Таким чином Flutter перемагає інші доступні мови, через те вони не мають такого гнучкого засобу розробки.

Через обмеження на динамічне виконання коду в iOS, Flutter використовує AOT-компіляцію. Широко використовується можливість платформи Dart – «гаряче перезавантаження», коли зміна вихідного коду

застосовується одразу у робочого додатку без необхідності його перезапуску. Що є переважним фактором у часі, витраченому програмістами на розробку.

Двигун Flutter написаний переважно на C++, підтримує низькорівневий рендеринг за допомогою графічної бібліотеки Google Skia, має можливість взаємодіяти з платформо-залежними SDK під Android та iOS.

Дизайн інтерфейсу програм Flutter передбачає використання віджетів, що описуються як незмінні об'єкти будь-якої частини інтерфейсу користувача. Усі графічні об'єкти, включаючи текст, форми та анімацію, створюються за допомогою віджетів; комбінуванням простих віджетів створюються складні віджети. З фреймворком надаються два основні набори віджетів – Material Design (стиль Google) та Cupertino (стиль Apple). Тобто, Flutter не треба індивідуального налаштування до платформ запуску. Інше кажучи, всі необхідні елементи він має у собі та ці елементи мають однаковий вигляд з елементами платформи.

Dart – це мова програмування, створена Google. Позиціонується як заміна JavaScript. Один з розробників мови Марк Міллер зазначив, що JavaScript має фундаментальні вади, які неможливо виправити, тому створили Dart.

Dart – це C-подібна мова, перш за все схожа на мову Java, тому розробникам цієї мови програмування простіше вивчати Dart з Flutter.

Андрій Доронічев – засновник Youtube Mobile, зазначив у Інстаграм публікації наступне:
“Минуле – сайти, теперішнє – мобілка, майбутнє – AR”.
Насамперед можна спостерігати тенденцію, як кількість розробки сайтів за допомогою мов програмування не зростає, тому що бізнес користується конструкторами, а кількість і підтримка розробки мобільних додатків зростає.

Головні банки нашої країни, які працюють по всьому світу, працюють за допомогою мобільного додатку, тому що смартфон є у кожної людини та завжди з собою, якщо на ньому встановлений додаток, то деякі функції можуть працювати без інтернету у порівнянні з сайтами.

Гордість України — застосунок Дія, який у собі має офіційну, електронну версію паспорта, студентського квитка, водійські права. Усе це у мобільному додатку і якщо немає інтернет-з'єднання, можна використати головні функції офлайн.

1.2 Віджети. Stateless, Stateful. Дерево віджетів

Слід виділити, що у Flutter програма повністю будується на віджетах, візуально видимих і візуально невидимих, за допомогою яких будується дерево віджетів, зразок якого можна побачити на рисунку 1.2

Віджети – це всі об’єкти в Dart за яких можна не тільки представляти елементи інтерфейсу користувача але і функціональні компоненти, такі як виявлення жестів GestureDetector, передача даних теми(кольору, частин дизайну) програми й так далі.

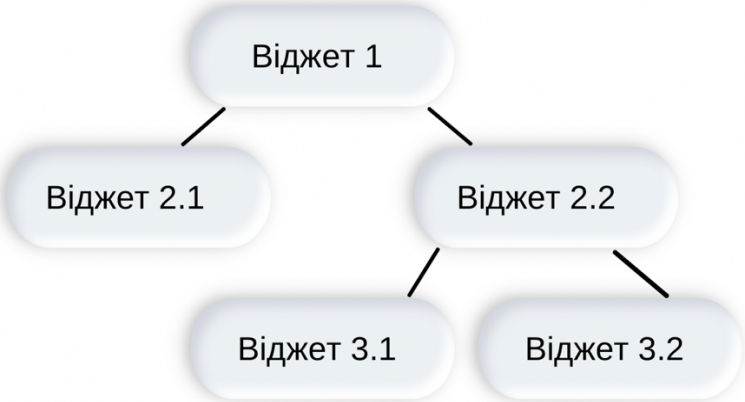


Рисунок 1.2 – Зразок дерева віджетів

Є лише два типи віджетів:

Stateless Widgets – віджети не мають стану, незмінні віджети, такі як Row, Column, Expanded, Container.

Stateful Widgets – віджети, які мають стан, які змінюють свою форму або показники в процесі користування програмою. Віджети, такі як TextField, Checkbox, Radio, Slider.

Основні віджети для розмітки та побудови програми. Container, Column, Row, Expanded.

- Container – містить будь-який віджет у собі та обрамляє його собою.
- Column – вертикальний стовпчик, який містить будь-який віджет у собі та обрамляє його собою.
- Row – горизонтальний стовпчик, який містить будь-який віджет у собі та обрамляє його собою.
- Expanded – міститься лише в рамках Column, Row та Flex. Обертаючи в нього віджет, він займає весь доступний простір віджета у який Expanded вкладений, впритул до інших спадкоємців.

За допомогою цих віджетів можна зробити просту розмітку для свого додатка.

Віджети каркаса додатка на Flutter.

Material App – відповідає за всю візуалізацію додатка, створення нової програми.

Scaffold – Головний макет для додатка, який вже має в собі частини візуалізації, такий як appBar: AppBar(), або body:. Просту розмітку для додатку можна побачити на рисунку 1.3.

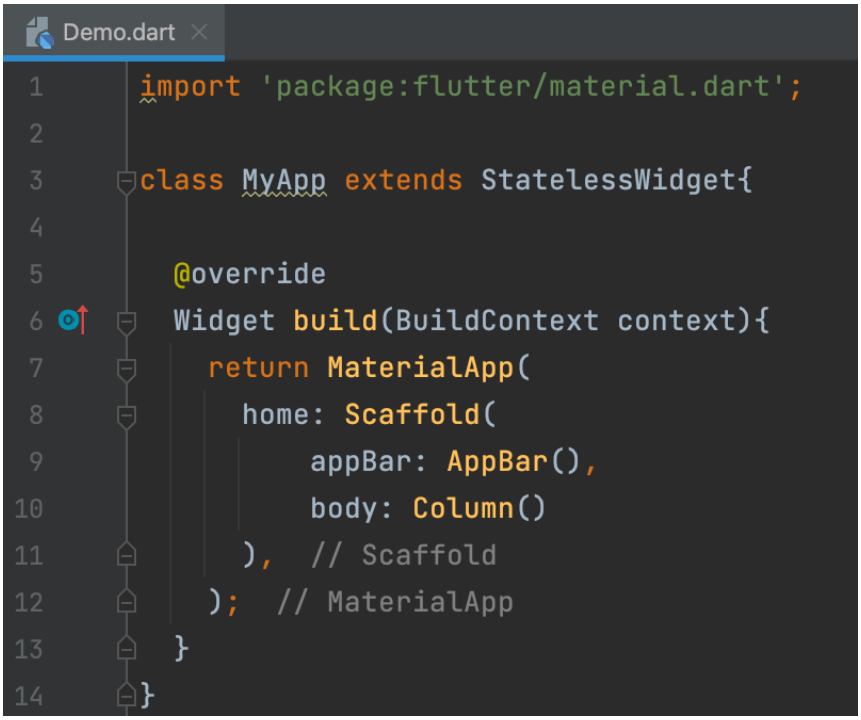


Рисунок 1.3 – Зразок стандартного каркаса додатка

1.3 Переваги розробки Flutter з нативними мовами розробки. Кросплатформеність

Додаток створений за допомогою Flutter є кросплатформеним, тому може використовувати єдину кодову базу для платформ Android, IOS, Web, Google Funchosia. Завдяки цьому, наряду з нативними мовами розробки, використовуючи Flutter, треба мінімум вдвоє менше спеціалістів, коли додаток небхідно розробити на дві платформи IOS, Android. Вдвоє менше часу, вдвоє менше грошей на розробку. Якщо технічне завдання дозволяє зробити додаток на Flutter, обов‘язково треба використовувати цю можливість, тому що він здатен інтегруватись з різними мовами програмування, які підтримують 3D, AR, та Assembler технології.

Також поруч з нативними мовами розробки, можна швидше зробити мінімально робочу модель свого проекту, щоб подивитись як ідея буде працювати, багато часу не знадобиться.

Завдяки тому що Flutter має у собі нативний UI Google і IOS, у бібліотеках Material і Cupertino. Це дозволяє запускати додаток з графікою іншого

пристрою, та у випадку точно визначеної платформи зробити користувачу відчуття свого системного додатка від користування додатком.

Слід виділити наступні переваги:

Менше робоче навантаження – не потрібна комунікація між розробниками різних платформ.

Швидше розробка – вдвоє менше часу на розробку для двох платформ, натомість нативним додаткам для IOS, Android, необхідно більше часу для розробки.

Хмарна інтеграція – завдяки єдиній кодовій базі, додаток легше інтегрується з хмарними сервісами.

1.4 Переваги Flutter над наявними кросплатформеними React JS. KotlinMultiplatform.

Швидкість розробки коду

Одна з ключових функцій Flutter – Hot Reload, що дозволяє розробникам швидко переглядати ефект змін у коді. Натискаючи на неї зміни оновлюються за 1-3 секунди. Якщо працювати у середі розробки VS Code, можна використати старшого брата цієї функції Hot Restart, натискаючи на неї оновлюються зміни у додатку що оновлюють його до початкового стану. Ці функції дуже зручні для спільної роботи розробників і дизайнерів, а така паралельна робота дуже збільшує швидкість розробки. Коли в інших мовах потрібно заново робити повний запуск додатка.

Пунктом у списку недоліків Kotlin є нижча швидкість компіляції у порівнянні з Java. У поодиноких випадках Kotlin виграє у Java у швидкості компіляції але найчастіше вона набагато нижча. При компіляції інкрементальних змін Kotlin добре справляється зі своєю роботою але він все одно не такий швидкий, як сучасний варіант Flutter.

У React Native є функція Fast Refresh: вона дозволяє розробникам встановлювати новий код безпосередньо в діячу програму але під час роботи з

React Native бувають деякі проблеми. Наприклад, спроби оновити версію React Native часто призводять до зламання бібліотек.

Інструменти автоматичного оновлення версії іноді не працюють, тому розробникам доводиться оновлювати деякі компоненти вручну.

React Native не такий гарний і з погляду безпеки. Наприклад, будь-хто може отримати доступ до коду JavaScript у складання релізу, а це серйозна загроза для всієї програми.

За вищевикладеним можемо зазначити, що швидкість Flutter перемагає не тільки за швидкістю розробки але і за швидкістю роботи додатка.

Порівняння UI

У Flutter є два набори віджетів: Material Design та Cupertino. Перша група імітує нативний дизайн Android (оскільки Material Design – це мова дизайну, орієнтована на Android, розроблена Google), друга група – iOS. Завдяки цьому програми Flutter виглядають нативно та працюють на обох платформах. Можна їх не використовувати та зробити свої віджети для інших платформ але це рідко використовують для всього додатку.

У Kotlin Multiplatform є SDK, який допомагає спростити використання загального коду на Android та iOS. Мета технології — винесення бізнес-логіки. UI-шар залишається нативним, що добре позначається на досвіді користувача та зовнішньому вигляді додатків але кросплафтормена тільки бізнес-логіка, насамперед робота з UI відбувається окремо з напрямком Android, IOS.

React Native має набір основних компонентів, таких як Button, Image, ScrollView та інші; він також має деякі компоненти для Android і iOS але вони не ідентичні з нативними.

На закінчення слід сказати, що UI у Flutter кращий за наявні рішення кросплатформеної розробки.

Популярність запитів мов програмування у Google Trends станом на 2022 рік 15 квітня, можна побачити на рисунку 1.4.

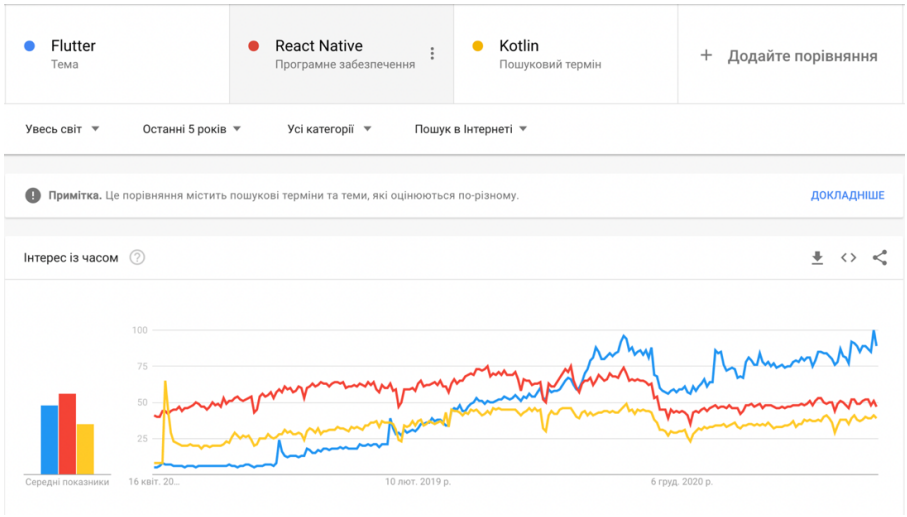


Рисунок 1.4 –

Графік запитів головних кросплатформених мов програмування

Бачимо, що переважно перемагає Flutter, наступним є React Native, останнім є Kotlin Multiplatform, за його запитом не було навіть багато запитів, тому порівнюємо зі звичайним Kotlin.

1.5 Середовище розробки Android Studio. Емулятори додатка – web, IOS, Android

Android Studio дозволяє запускати свій додаток за допомогою емуляторів IOS, Android, Web. Логотип Android Studio можна побачити на рисунку 1.5.

Для роботи, емулятори необхідно створити в Android Studio, тільки як було встановлено середовище розробки. Середовище дозволяє встановити емулятор практично будь-якої сучасної моделі смартфона у системах Android та IOS. Додаток у веб вигляді запускає у будь-якому браузері, в нашому випадку це – Google Chrome.

Має у собі емулятори для IOS, Android, Web, Пропонує в головному меню зробити новий Flutter проект. Нижче показую як виглядають емулятори із запущеним тестовим додатком, який є у кожному новому проекті на Flutter, можна побачити на рисунку 1.6.

Рисунок 1.5 – Логотип Android Studio

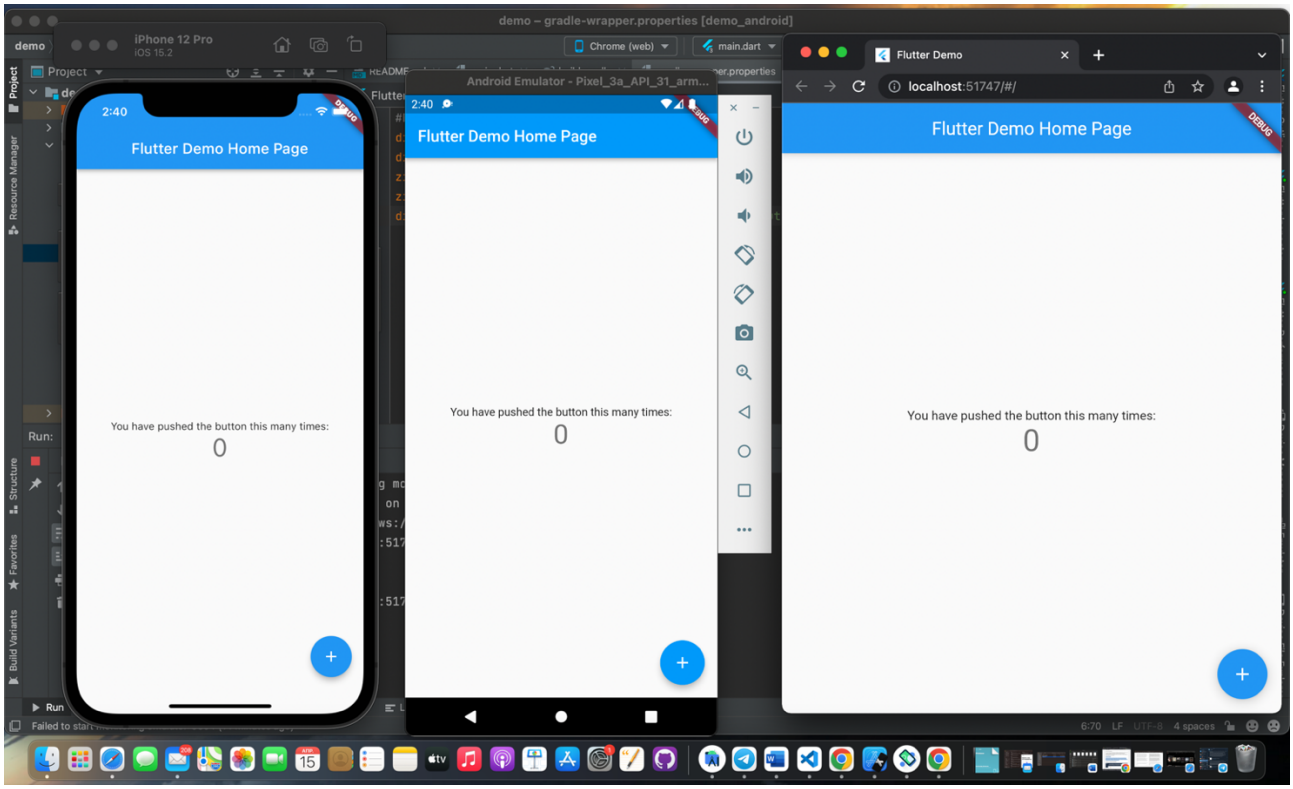


Рисунок 1.6 – Емулятори запуску додатка на платформах: IOS – Iphone 12 Pro, Android – Google Pixel 3A, Web – Chrome

1.6 Firebase бази даних, замість SQL-подібних баз даних

Firebase — американська компанія, постачальник хмарних послуг, заснована у 2011 році Ендрю Лі та Джеймсом Темпліном, і поглинена у 2014 році корпорацією Google.

Основний сервіс — хмарне сховище класу NoSQL, що дозволяє розробникам додатків зберігати та синхронізувати дані між кількома клієнтами.

Підтримані особливості інтеграції з додатками під операційні системи Android та iOS, реалізовано API для додатків на JavaScript, Java, Objective-C та Node.js, також можна працювати безпосередньо з базою даних у стилі REST з JavaScript-фреймворків, включаючи AngularJS, React, Vue.js, Ember.js та Backbone.js. Передбачено API для шифрування даних.

Серед інших послуг, що надавалися компанією, запущений 13 травня 2014 року хостінг для зберігання статичних файлів (таких як CSS, HTML, JavaScript), що забезпечує доставлення через CDN та сервіс аутентифікації клієнта з використанням коду тільки на стороні клієнта з підтримкою входу через Facebook, GitHub, Twitter та Google (Firebase Simple Login). Логотип Firebase можна побачити на рисунку 1.7.

Рисунок 1.7 – Логотип Firebase бази даних

Звичайний сучасний додаток працює з SQL базою даних, для використовування якої розробнику потрібно використовувати не тільки JSON-конвертер даних для розуміння базою даних але і http, GET і POST запроси до бази даних. Схему роботи звичайної бази даних можна побачити на рисунку 1.8.

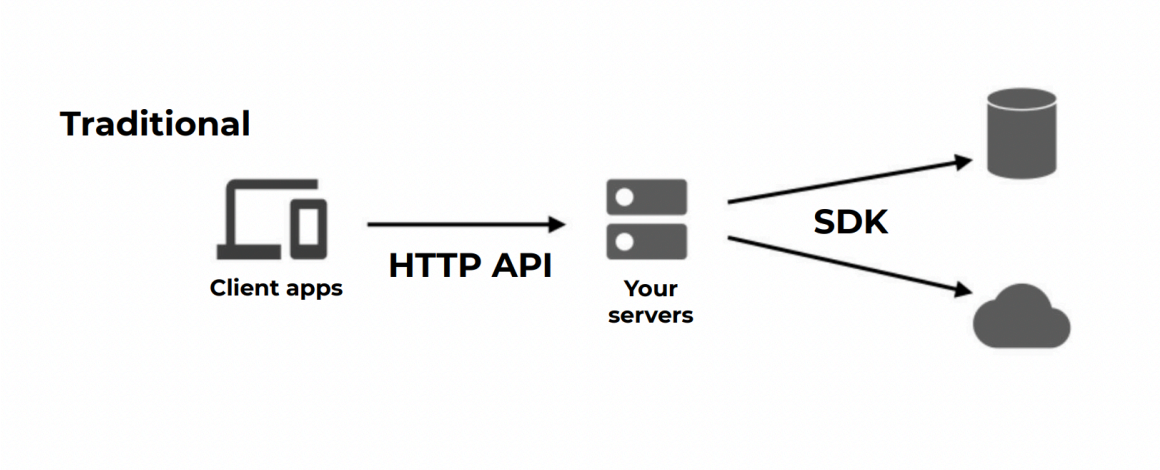


Рисунок 1.8 – Схема взаємодії додатка зі звичайною SQL базою даних

Firebase являє собою NoSQL базу даних. Це говорить про те, що коли ми з'єднаємо базу даних з нашим мобільним додаток нам не потрібен обробник між додатком і базою даних. На початку дерева прокидаємо потік даних з Firebase на весь додаток. Далі використовуємо методи з Firebase за допомогою бібліотек аутентифікації, сховище і багато інших сервісів. Особисто за допомогою сервісу Firebase підтримки Flutter – Flutterfire. Тобто нам потрібно тільки конвертувати JSON файли, а Firebase за нас зробить запроси до бази даних, без створення запитів. Що прискорює роботу та є зручним. Схему роботи Firebase можна побачити на рисунку 1.8.

Рисунок 1.8 – Схема взаємодії додатка з базою даних Firebase

1.7 Технічне завдання по розробці мобільного додатку.

Призначення розробки

Додаток повинен у вигляді онлайн-магазину охопити та інтегруватись в усі головні апаратні системи існування людини в інтернеті. Користуватись додатком в інтернеті як сайтом, користуватись в App Store/Play Market як мобільним додатком, на системах IOS або Android. Використовуючи єдину кодову базу для усіх платформ. Що збільшує можливість присутності бренду у житті клієнта. Це сприяє продажу, охопленню маркетингової компанії, покращить рівень бренду та автоматизує процес замовлення.

Мета розробки

Розробити зручний додаток для клієнтів та адміністраторів магазину посуду. Користувач повинен знайти, обрати бажаний товар, можливість ознайомитись з ним детальніше. Після чого, замовити його у будь-якій кількості до кошика, редагувати кількість, заповнити контактні дані для відправки замовлення, або замовити товар швидким шляхом, відправляючи номер телефона для зворотного зв‘язку.

Користувачі додатка

Користувачами будуть клієнти наявного інстаграм-магазина посуду «LY.Odesa». Також власники бізнесу зможуть адмініструвати замовлення у додатку.

Вимоги до програмного та апаратного забезпечення

Для користування додатком, потрібен смартфон з операційною системою IOS/Android, або ноутбук, комп‘ютер, те що має доступ для користування онлайн-сайтами. Додатками можна користуватись на будь-яких версіях IOS/Android, це сприяє тільки швидкості. Рекомендовано на пристроях системи IOS починати з 9 версії, сучасність моделі Iphone не обов‘язкова.

Рекомендовано на пристроях системи Android починати з 5 версії, сучасність моделі телефону не обов‘язкова.

Стосовно ноутбука, комп‘ютера тощо, додаток у вебвигляді працює практично на кожному пристрої.

Для розробки рекомендовано спільним поглядом спеціалістів, станом на 2022 використовувати Macbook Air M1 з 16Гб оперативної пам'яті, на якому і буде проходити розробка застосунку. Звісно можливе використовування і більш застарілих моделей Mac, це сприяє тільки на швидкість. Можна використовувати пристрої з системою Windows але щоб запускати додаток потрібно встановити емулятор ОС Mac, щоб мати підпис розробника Apple. Також для розробки можливе використання справжніх пристроїв IOS,Android при правильному підключенні їх до пристрою розробки.

2. ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКА

2.1 Архітектура мобільного додатку

Досліджено, що всі наявні кампанії використовують правила Роберта Мартіна – чистої архітектури(Clean Architecture). У правилах чистої архітектури зазначено, що елементи мають бути незалежними один від одного. Щоб програму було легше тестувати, масштабувати за допомогою додавання нових елементів. Тому ми використаємо ці правила на рівні директорій і надалі в проекті, на рівні класів та віджетів.

Чиста архітектура проекту буде заснована на трьох шарах:

1.Шар UI – тут необхідно зберігати всі графічні дані. Екрани користувача, окремі елементи, такі як галерея детального екрана, кнопки, форми заповнення, сторінка замовлення і т.д.

2.Шар бізнес-логіки – тут необхідно зберігати методи, функції обробки дій користувача, локальні дані користувача, абстрактні класи.

3.Шар даних – тут необхідно зберігати все, що пов’язано с базами даних, всі товари магазину, Firebase.

Схему архітектури можна побачити на рисунку – 2.1.

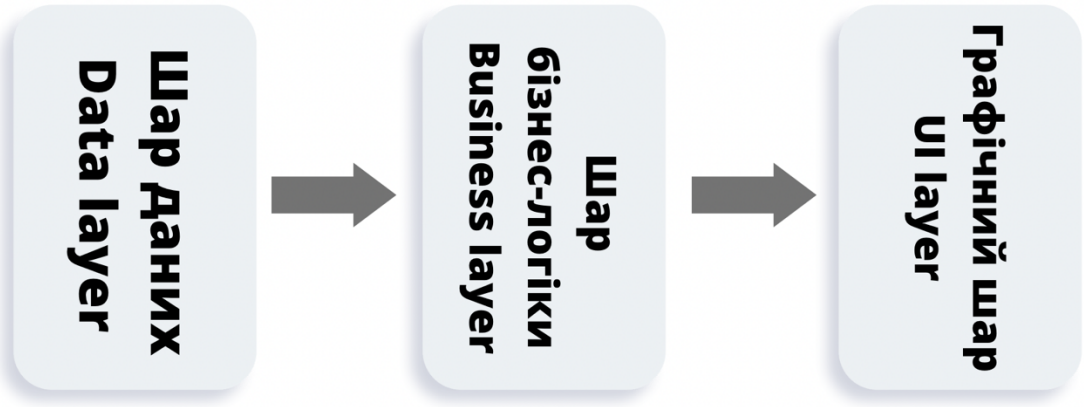


Рисунок 2.1 – Схема архітектури додатка

2. 2. Застосування принципів чистої архітектури, ООП у мобільному додатку

Об'єктноорієнтоване програмування (скор. ООП) – методологія програмування, заснована на представленні програми у вигляді сукупності взаємодійних об'єктів, кожен з яких є екземпляром певного класу, а класи утворюють ієрархію спадкування. Переважно всі компанії дотримуються правил ООП, має три основні принципи дотримання правил:

Спадкування (англ. inheritance) – концепція об'єктноорієнтованого програмування, згідно з якою абстрактний тип даних може успадковувати дані та функціональність певного типу, сприяючи повторному використанню компонентів програмного забезпечення.

Інкапсуляція (англ. encapsulation, від лат. in capsula) – в інформатиці, процес поділу елементів абстракцій, що визначають її структуру (дані) та поведінку (методи); інкапсуляція призначена для ізоляції контрактних зобов'язань абстракції (протокол/інтерфейс) від реалізації. Насправді це означає, що клас повинен складатися з двох частин: інтерфейсу та реалізації. У реалізації більшості мов програмування (Dart, C++, C#, Java та інші) забезпечує механізм приховування, що дозволяє розмежовувати доступ до різних частин компонента.

Поліморфізм(англ. polymorphism) – здатність однією функцію однаково вдало обробляти дані різних класів спадкованих від класу родича.

2.3 Визначення необхідних, додаткових пакетів для взаємодії з додатком

Firebase Core – це ядро для обробки джерела баз даних Firebase.

Назва пакета – firebase_core.

Firebase Cloudstore – це бази даних Firebase, де ми будемо зберігати інформацію та обмінюватись даними за допомогою JSON-конвертації.

Назва пакета – cloud_firestore.

Firebase Authentication – це база даних аутентифікації, для взаємодії з користувачами додатка.

Назва пакета – firebase_auth.

Provider – це пакет для зберігання бізнес-логіки додатка. Зберігання та надання локальних даних.

Назва пакета – provider.

Lottie – це пакет для анімації Gif в додатку для додаткової графіки.

Назва пакета – lottie.

Всі пакети ми будемо шукати на сайті pub.dev разом з інструкціями їх встановлення. Пошук на офіційному сайті можна побачити на рисунку – 2.2.

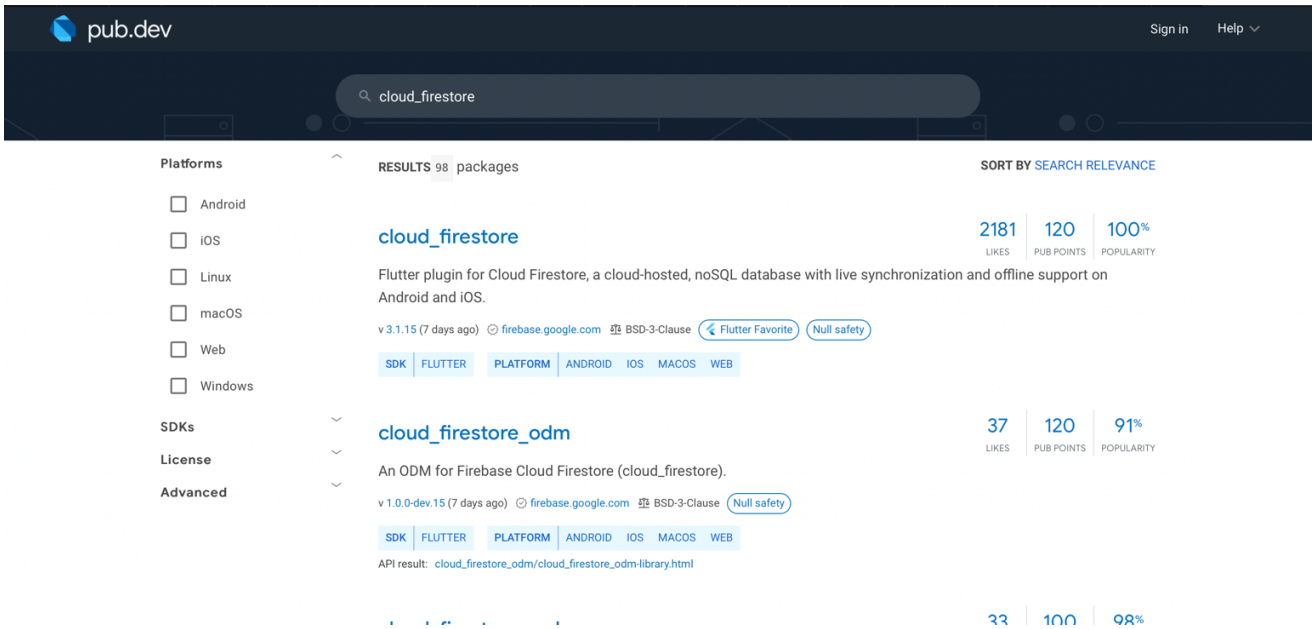


Рисунок 2.2 – Зразок пошуку необхідного пакета cloud_firestore

2.4 Проектування UI-частини з Flutter

Нам потрібно декілька екранів, які будуть виконувати наступні функції:

1. Головна сторінка(англ. Home Page) – тут ми будемо розташовувати всі товари, верхню графічну панель= на якій закріпимо посилання за кнопками на сторінки: аутентифікації, детальну сторінку, пошук товару за назвою.

2. Детальна сторінка(англ. Details Screen) — тут ми розташуємо повний опис товару, додаткові фото, корисну для покупця інформацію, функцію звичайної та швидкої покупки.

3. Сторінка кошика(англ. Folder Screen) – тут буде кошик користувача, в якому він може дивитись загальну суму товару однієї назви, суму всіх загалом, видаляти товари та переходити на сторінку замовлення.

4. Сторінка замовлення(англ. Order Screen) – тут користувач заповнить свої контактні дані для надіслання, відправить замовлення.

5. Сторінка аутентифікації(англ. Authentication Screen) – тут наша програма буде додавати базу користувачів, у випадку підключення спеціального логіна та пароля перенаправить адміністратора на сторінку перегляду замовлення.

6. Сторінка адміністрації(англ. Admin Screen) – тут адміністратор зможе дивитись замовлення, які зробили користувачі за звичайними та швидкими замовленнями.

Взаємодію навігації екранів можна побачити на рисунку 2.3.

Рисунок 2.3 – Схема взаємодії екранів між собою

2.5 Проектування бази даних із взаємодією з додатком

Розроблена система, яка буде обмінюватись даними з локальною пам'яттю додатка, а саме за допомогою екрана користувача, буде надсилати дані з локальної пам'яті до баз даних, використовуючи – Provider. Це пакет розроблений Ремі Руслє, який підтримують офіційні розробники Flutter спеціальною позначкою. Пакет допомагає набагато легше користуватись локальною пам'яттю. Використовує оригінальні віджети Flutter – Inherited Widget, спрощує роботу з ними.

Inherited Widget – саме він фундаментально підтримує клас для зберігання локального стану даних.

Принцип Provider у тому, що він підписує слухачів (віджети) на свої дані, надає їх але значно зручніше ніж InheritedWidget.

У дипломній роботі демонструємо прикладом клас – CartData, який спадкуємо словом with від ChangeNotifier. У ньому зберігатимуться товари кошика з методами їх обробки та відправки.

Звертаючись до цього файлу слід імпортувати provider.dart у файл, в якому треба з ним працювати.

Звертаючись до provider.dart, доступні дві ключових функції:

1. Зчитування даних для показу на екрані – context.watch<‘Назва класу’>().‘Метод/Дані з класу‘
2. Використовування даних для роботи з методами, та даними – context.read<‘Назва класу’>()‘Метод/Дані з класу‘

Слід виділити, що це є зручним методом для необхідного використання взаємодії з локальною пам’яттю.

При надісланні замовлення, користувач звертається до бази даних. За допомогою JSON-конвертації, конвертує дані у формат JSON, в якому зберігаються дані у Firebase.

Перевагою цього метода взаємодії з базами даних, є те що не треба користуватись http-запитами GET, POST як у звичайних SQL базах даних, що спрощує роботу, взаємодію з даними.

Водночас замовлення одразу приходить до всіх, хто під‘єднаний до баз даних. Інакше кажучи, на сторінці адміністратора з‘явиться замовлення. Схему взаємодії можемо побачити на рисунку 2.4.

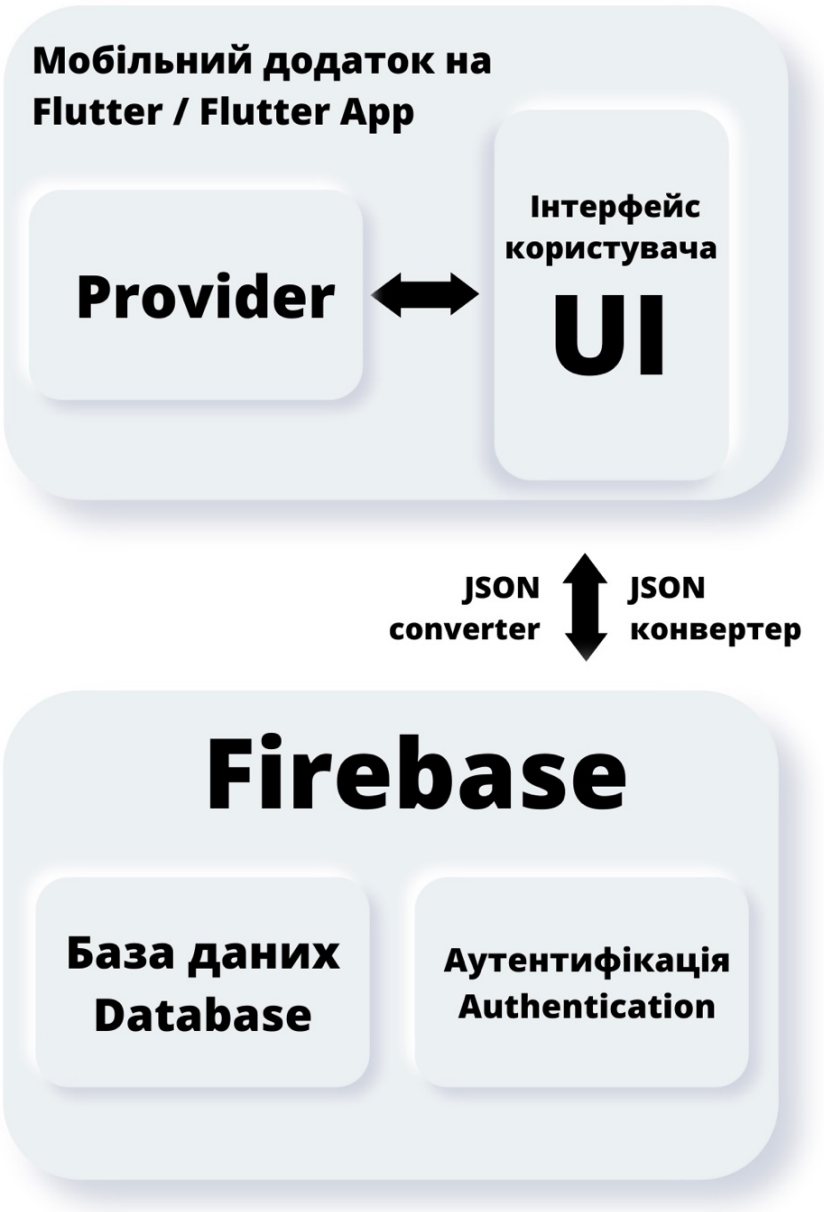


Рисунок 2.4 – Схема взаємодії даних додатка

2.6 Етапи створювання мобільного додатка

Етапи розробки:

1.Створимо локальну інформацію про товари.

2.Головний екран, детальний, пошук, кошик, сторінка замовлення, сторінка аутентифікації, сторінка адміністратора.

3.Provider як стан додатка, товар та його кількість.

4.База даних Firebase.

5.Створюємо конвертер, вкладаємо інстанс колекції в сторінку адміна.

6.Тестуємо додаток

3. РОЗРОБКА МОБІЛЬНОГО ДОДАТКА

3.1 Створення проекту в Android Studio.

Цей розділ присвячений створенню проекту в Android Studio.

Треба завантажити Android Studio з офіційного ресурсу, для своєї операційної системи.

Відкрити встановлений Android Studio та встановити необхідні емулятори. В розділі Device Manager запропонують багато моделей смартфонів з операційною системою Android.

Щоб завантажити модель телефону Iphone з операційною системою IOS. Необхідно встановити XCode та завантажити собі модель Iphone з яким зручно працювати розробнику.

Емулятори встановлено.

Для створення проекту, необхідно натиснути на «New Flutter Project»

Проект створено.

Необхідно розробити класи товарів, з якими даними буде працювати додаток та база даних:

Product – зразок товару для спадкування окремих категорій. Plate, Glasses, Cutlery, Cup, Other – категорії товарів.

ProductCart – товар типу Product з новими параметрами кількості та загальною сумою за товар.

ProductOrder – товар типу ProductCart з новими параметрами контактних даних клієнта.

ProductSpeedOrder – товар типу Product з новими параметрами для швидкого замовлення окремого товару.

Створені класи можна побачити на рисунку 3.1.

```dart
class Product {
  final int id, price;
  final String title, image, description, color, size;
  final List<String>? images;

  Product(this.id, this.price, this.title, this.image, this.description, this.color, this.size, this.images);}

class Plate extends Product{
  Plate({required String size, required int id, required int price, required String title,
        required String image, required String description, required String color, List<String>? images,
      }) : super(id, price, title, image, description, color, size, images);}

class Glasses extends Product {
  Glasses({required String size, required int id, required int price, required String title,
        required String image, required String description, required String color, List<String>? images
      }) : super(id, price, title, image, description, color, size, images);}

class Cup extends Product {
  Cup({required String size, required int id, required int price, required String title,
        required String image, required String description, required String color, List<String>? images
      }) : super(id, price, title, image, description, color, size, images);}

class Cutlery extends Product {
  Cutlery({required String size, required int id, required int price, required String title,
        required String image, required String description, required String color, List<String>? images
      }) : super(id, price, title, image, description, color, size, images);}

class Other extends Product {
  Other({required String size, required int id, required int price, required String title,
        required String image, required String description, required String color, List<String>? images
      }) : super(id, price, title, image, description, color, size, images);}

class ProductCart extends Product{
  int amountItem; int totalPrice;

  ProductCart({
    required id, required price, required title, required image, required description,
    required color, required List<String>? images, required size, required this.amountItem, required this.totalPrice
  }) : super(id, price, title, image, description, color, size, images);

  factory ProductCart.fromJson(Map<String, dynamic> json) {
    return ProductCart(
      id: json['id'] as int, price: json['price'] as int, title: json['title'] as String,
      image: json['image'] as String, description: json['description'] as String,
      color: json['color'] as String, size: json['size'] as String,
      images: (json['images'] as List).map((e) => e as String).toList(),
      amountItem: json['amountItem'] as int, totalPrice: json['totalPrice'] as int,);}

  Map<String, Object?> toJson() => {
    'id': id, 'price': price, 'title': title, 'image': image, 'description': description, 'color': color,
    'images': images, 'size': size, 'amountItem': amountItem, 'totalPrice': totalPrice,};}

class ProductOrder{
  final List<ProductCart> order; final int orderNumber; final int dayTime;
  final int monthTime; final int hourTime; final int minuteTime;
  final String name; final String post; final String city; final String phoneNumber;

  ProductOrder({
    required this.orderNumber, required this.order, required this.dayTime, required this.monthTime,
    required this.hourTime, required this.minuteTime, required this.name, required this.post,
    required this.city, required this.phoneNumber,
  });

  factory ProductOrder.fromJson(Map<String, dynamic> json) {
    return ProductOrder(
        order: (json['order'] as List<dynamic>).map((dynamic e) => ProductCart.fromJson(e as Map<String,
dynamic>)).toList(),
        orderNumber: json['orderNumber'] as int, dayTime: json['dayTime'] as int,
        monthTime: json['monthTime'] as int, hourTime: json['hourTime'] as int,
        minuteTime: json['minuteTime'] as int, name: json['name'] as String,
        post: json['post'] as String, city: json['city'] as String, phoneNumber: json['phoneNumber'] as String,);}

  Map<String, Object?> toJson() => {
    'order': order.map((e) => e.toJson()), 'orderNumber': orderNumber, 'dayTime': dayTime,
    'monthTime': monthTime, 'hourTime': hourTime, 'minuteTime': minuteTime,
    'name': name, 'post': post, 'city': city, 'phoneNumber': phoneNumber,};}

class ProductSpeedOrder{
  final int userPhoneNumber; final String title; final int id; final int amountItems;
  final int dayTime; final int monthTime; final int hourTime; final int minuteTime;
  final int price; final int totalPrice;


  ProductSpeedOrder({
    required this.userPhoneNumber, required this.title,
    required this.id, required this.amountItems,
    required this.dayTime, required this.monthTime,
    required this.hourTime, required this.minuteTime,
    required this.price, required this.totalPrice
  });

  factory ProductSpeedOrder.fromJson(Map<String, dynamic> json){
    return ProductSpeedOrder(
        userPhoneNumber: json['number'] as int, title: json['title'] as String, id: json['id'] as int,
        amountItems: json['amountItems'] as int, dayTime: json['dayTime'] as int,
        monthTime: json['monthTime'] as int, hourTime: json['hourTime'] as int,
        minuteTime: json['minuteTime'] as int, price: json['price'] as int, totalPrice: json['totalPrice'] as int);}

  Map<String, Object?> toJson() => {
    'number': userPhoneNumber, 'title': title, 'id': id,
    'amountItems': amountItems, 'dayTime': dayTime, 'monthTime': monthTime,
    'hourTime': hourTime, 'minuteTime': minuteTime, 'price': price, 'totalPrice': totalPrice,};}
```

Рисунок 3.1 – Створені класи для додатка classes.dart

3.2 Створення структури проекту за правилами чистої архітектури

Нарешті час розробити архітектуру проекту. Мета якої – бути зрозумілою для іншого розробника, який зможе працювати з кодом отриманої програми. Бути ефективною для подальшого тестування окремих частин або елементів додатка для визначення частини в якій сталася помилка. Слід зазначити що чиста архітектура також сприяє зручному масштабуванню.

Розподілимо директорії на 3 типи:

1.data – шар даних.

2.business-logic – шар бізнес-логіки.

3.ui – шар дизайну.

Створені директорії можна побачити на рисунку 3.2.

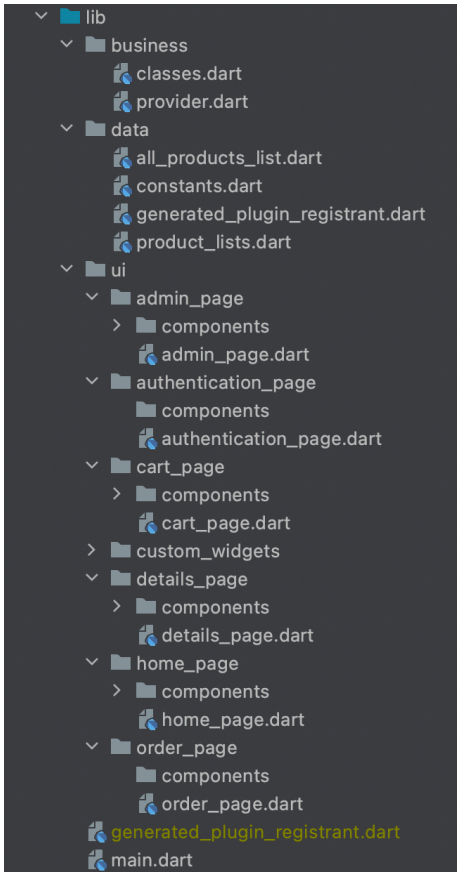


Рисунок 3.2 – Розподілений на окремі частини проект

3.3 Імпорт та встановлення необхідних пакетів

Користувавшись інформацією у розділу 2.3, зазначили необхідні пакети для додатка.

Для встановлення їх, необхідно зробити наступні кроки:

1.Подивитись останню версію пакета на його сторінці в pub.dev.

2.Відкрити файл pubspec.yaml, у розділі dependencies: вписати пакет з версією.

3.У терміналі Android Studio написати команду flutter pub get. Таким чином Flutter проаналізує проект на оновлення пакетів, та встановить їх у програму.

4.Пакет встановлено, за необхідністю використання, необхідно імпортувати його у файл в якому він буде працювати. Командою import ‘package:flutter/«Назва пакета».dart’;

3.4 Створення UI/UX-дизайну графічних елементів макета додатка, елементів, навігації та інтерфейсу

3.4.1 Головний екран – Home Page

Основна мета полягає у тому, щоб за допомогою SDK Flutter і мови Dart розробити зручний інтерфейс та логіку його взаємодії для користувачів додатка. На етапі проектування визначено, що для повної комунікації у додатку необхідно 6 екранів.

Сутність цього інтерфейсу, полягає у зображенні всіх товарів на одній сторінці, зі зручним та швидким пересуванням між окремими категоріями товарів. Для цієї мети необхідно використати основоположні віджети – Scaffold(), Column(), Container(), Expanded(), слайдери товарів за допомогою створення окремого віджета з головною частиною використання елементів списку з ListView.builder().

За допомогою DefaultTabBarView(), TabBarView(), TabBar(), буде зроблено пересування між екранами з окремими категоріями, які розташовуються у віджеті AppBar() – верхній частині сторінки. Layout.builder() – за допомогою

якого екран адаптується до встановлених нами значень, за якими екран ініціалізує вебвигляд, що орієнтовно починається з 600 пікселів.

Зазначимо, що CustomSearchDelegate() – це мною створений віджет, за допомогою копіювання с офіційного віджета DataSearch(), який зберігається у створеній директорії custom_widgets.

Різницю між оригінальним та власним віджетом пошуку, здійснено таким чином, що коли користувач повертається зі сторінки за якою він перейшов у пошуку. Екран списку пошуку залишається відкритим і не зникає після повернення назад. Щоб реалізувати це, було змінено параметр:
bool get maintainState false;
З false на true.

Щоб реалізувати цей віджет, необхідно зробити від нього спадкування, та використовувати успадкований віджет з власними налаштуваннями.

Для покращення архітектури, програмування екрана розділено на такі частини:

- Директорія home_page, можна побачити на рисунку 3.3.
- Головний файл – home_page.dart, можна побачити на рисунку 3.4.

Компоненти — директорія components:

- Файл керування сторінкою в AppBar – app_bar_tab.dart, можна побачити на рисунку 3.5.
- Файл з головною частиною екрана Home Page – body_tab_bar.dart, можна побачити на рисунку 3.6.
- Файл віджета пошуку – data_search.dart, можна побачити на рисунку 3.7.
- Файл з віджетами зображення товарів у екранах Home Page – item_cards.dart.
- Файл з віджетом, який здійснює галерею товарів – product_slider_widget.dart.
- Файл з керуванням зображення сторінок категорій у Home Page – tab_bar_view_home.dart, можна побачити на рисунку 3.8.
- Директорія tabs, можна побачити на рисунку 3.9.

- Зразок файлу сторінки окремої категорії товару — файл cup.dart на основі якого 4 ідентичні файли з іншими категоріями товарів, можна побачити на рисунку 3.10.

  Отриманий результат home_page на екрані, можна побачити на рисунку 3.11.

  Отриманий результат data_search.dart на екрані, можна побачити на рисунку 3.12.

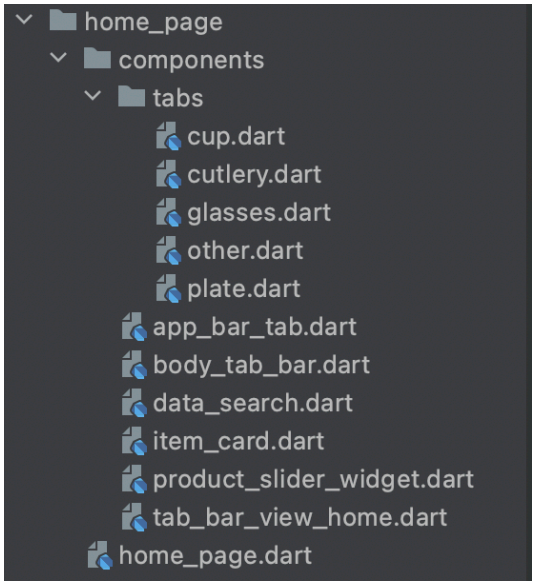


Рисунок 3.3 – Розподілені частини сторінки Home Page

```dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:ly_flutter/ui/home_page/components/app_bar_tab.dart';
import 'package:ly_flutter/ui/home_page/components/tab_bar_view_home.dart';

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

  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage>
    with SingleTickerProviderStateMixin {
  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(vsync: this, length: 6);
    _tabController.addListener(_handleTabSelection);
  }

  void _handleTabSelection() {
    setState(() {});
  }

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
        length: 6,
        child: Scaffold(
          appBar: PreferredSize(
            preferredSize: const Size.fromHeight(100),
            child: AppBarHome(tabController: _tabController)),
            body: TabBarViewHome(tabController: _tabController,)
        )
    );
  }
}
```

Рисунок 3.4 – Файл сторінки Home Page – home_page.dart

```dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:ly_flutter/business/provider.dart';
import 'package:ly_flutter/ui/authentication_page/authentication_page.dart';
import 'package:ly_flutter/ui/cart_page/cart_page.dart';
import 'package:ly_flutter/ui/custom_widgets/custom_search.dart';
import 'package:ly_flutter/ui/home_page/components/data_search.dart';
import 'package:provider/provider.dart';

class AppBarHome extends StatefulWidget{

  final TabController tabController;
  AppBarHome({Key? key, required this.tabController}) : super(key: key);

  @override
  _AppBarHomeState createState() => _AppBarHomeState();
}

class _AppBarHomeState extends State<AppBarHome>{

  @override
  Widget build(BuildContext context){
    return AppBar(
      automaticallyImplyLeading: false,
      actions: <Widget>[
        IconButton(
            onPressed: () {
              customShowSearch(context: context, delegate: CustomDataSearch(hintText: 'Пошук..'));
            },
            icon: Icon(Icons.find_in_page_outlined)),
        IconButton(
          padding: const EdgeInsets.all(10.0),
          icon: const Icon(Icons.account_circle_rounded),
          onPressed: () => Navigator.push(
              context, MaterialPageRoute(
                  builder: (context) => AuthenticationPage())),
        ),
        Stack(
            alignment: Alignment.center,
            children:[
              IconButton(
                padding: const EdgeInsets.only(bottom: 4.0, top: 1.0, left: 4.0, right: 4.0),
                icon: const Icon(CupertinoIcons.house_alt_fill, size: 28),
                onPressed: () => Navigator.push(
                    context, MaterialPageRoute(
                        builder: (context) => const CartPage())),
              ),
              Column(
                children: [Expanded(
                    child: Center(
                      child: Column(
                        crossAxisAlignment: CrossAxisAlignment.center, mainAxisAlignment: MainAxisAlignment.center,
                        children: [Container(
                              decoration: BoxDecoration(borderRadius: BorderRadius.circular(15), color:
Colors.white),

                              child: Padding(padding: const EdgeInsets.symmetric(vertical: 2.0, horizontal: 5.0),
                                child: Text(context.watch<CartData>().cartCart.length.toString(),
                                  style: const TextStyle(color: Colors.blue, fontSize: 12, fontWeight:
FontWeight.bold),),
                              ))],),),),],),]),],
      backgroundColor: Colors.blue,
      bottom: TabBar(
        controller: widget.tabController,
        isScrollable: true,
        indicatorWeight: 5.0,
        indicatorColor: Colors.white,
        unselectedLabelColor: Colors.grey,
        tabs: <Widget>[
          Tab(
              child: Text('Головна',
                  style: TextStyle(
                      color: widget.tabController.index == 0
                          ? Colors.yellow
                          : Colors.white))),
          .................
```

Рисунок 3.5 – Файл керування сторінкою в AppBar – app_bar_tab.dart

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/data/product_lists.dart';
import 'package:ly_flutter/ui/home_page/components/product_slider_widget.dart';

class BodyTabBar extends StatelessWidget {
  const BodyTabBar({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context){
    return
      SingleChildScrollView(
        child: Column(children: <Widget>[
          Container(
            child: Center(
              child: Row(
                children: [
                  Expanded(
                    flex: 2,
                    child: Container(
                      margin: const EdgeInsets.only(left: 70),
                      child: RichText(
                        text: const TextSpan(
                            style: const TextStyle(fontFamily: 'Lora'),
                            children: const [
                              TextSpan(text:
                              'Домівка посуду ',
                                style: TextStyle(color: Colors.white, fontSize: 30),),
                              TextSpan(text:
                              'LY',
                                style: TextStyle(color: Colors.yellow, fontSize: 30),),]
                        ),
                      ),
                    ),
                  ),
                  Column(
                    mainAxisAlignment: MainAxisAlignment.end,
                    children: [
                      Expanded(
                        child: Container(
                          width: 150,
                          height: double.infinity,
                          margin: const EdgeInsets.only(left: 10),
                          decoration: const BoxDecoration(image: DecorationImage(image:
AssetImage('assets/ukraine_is_my_home.png'), fit: BoxFit.fill)),
                        ),),
                    ],
                  ),
                ],
              ),
            ),
            height: 150,
            width: double.infinity,
            decoration: const BoxDecoration(color: Colors.blue),
          ),
          ProductSliderWidget(product_list: plate, title: 'Тарілки'),
          ProductSliderWidget(product_list: glasses, title: 'Бокали'),
          ProductSliderWidget(product_list: cutlery, title: 'Прилади'),
          ProductSliderWidget(product_list: cup, title: 'Стакани'),
          ProductSliderWidget(product_list: other, title: 'Інше'),
        ]),
      );
  }
}
```

Рисунок 3.6 – Файл з головною частиною екрана Home Page – body_tab_bar.dart

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/business/classes.dart';
import 'package:ly_flutter/data/main_list.dart';
import 'package:ly_flutter/ui/custom_widgets/custom_search.dart';
import 'package:ly_flutter/ui/details_page/details_page.dart';

class CustomDataSearch extends CustomSearchDelegate<String>{

  CustomDataSearch({
       required String hintText,
     }) : super(
       searchFieldLabel: hintText);

  List<Product> get allProductsSearch => allProducts;
  final List<Product> recentProducts = [
    Plate(
        id: 4,
        price: 110,
        color: 'Білий, голубий, небесний',
        title: 'Тарілка "Фарби"',
        image: 'https://firebasestorage.googleapis.com/v0/b/lyodesa-
c4045.appspot.com/o/assets%2Fproduct%2Fplates%2Fplate_paints%2Fplate_paints.jpeg?
alt=media&token=4dce368f-adf4-4e98-83c1-c871dff9cf3f',
        description: '',
        size: '20.6 см',
        images:
        ['https://firebasestorage.googleapis.com/v0/b/lyodesa-
c4045.appspot.com/o/assets%2Fproduct%2Fplates%2Fplate_paints%2Fplate_paints_1.jpg?
alt=media&token=994421e0-f158-4496-b4ea-86cb8dcc5876',
          'https://firebasestorage.googleapis.com/v0/b/lyodesa-
c4045.appspot.com/o/assets%2Fproduct%2Fplates%2Fplate_paints%2Fplate_paints_2.jpeg?
alt=media&token=4072aa06-5361-409d-9e91-d01577763cbc',]),
   ............];

  @override
  List<Widget> buildActions(BuildContext context){
    return [IconButton(icon: const Icon(Icons.clear), onPressed: () {query = '';})];
  }

  @override
  Widget buildLeading(BuildContext context){
    return IconButton(
      icon: AnimatedIcon(icon: AnimatedIcons.menu_arrow, progress: transitionAnimation),
      onPressed: () {close(context, '');},);
  }

  @override
  Widget buildResults(BuildContext context){
    final resultList = allProductsSearch.where((element) =>
element.title.toLowerCase().contains(query.toLowerCase())).toList();
    return ListView.builder(
      itemBuilder: (context, index) => ListTile(
        onTap: () {
          Navigator.push(context, MaterialPageRoute(builder: (context) => DetailsScreen(product:
resultList[index])));},
        leading: SizedBox(width: 30, height: 30, child: Image.network(resultList[index].image),),
        title: Text(resultList[index].title),),
        itemCount: resultList.length,
    );
  }

  @override
  Widget buildSuggestions(BuildContext context){
    final suggestionsList = query.isEmpty ? recentProducts : allProductsSearch.where((e) =>
e.title.toLowerCase().startsWith(query.toLowerCase())).toList();
    return ListView.builder(
        itemBuilder: (context, index) => ListTile(
          onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) => DetailsScreen(product:
suggestionsList[index])));
          },
          leading: SizedBox(width: 30, height: 30, child: Image.network(suggestionsList[index].image),
          ), title: Text(suggestionsList[index].title),),itemCount: suggestionsList.length,);
  }
}
```

Рисунок 3.7 – Файл віджета пошуку – data_search.dart

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/ui/home_page/components/body_tab_bar.dart';
import 'package:ly_flutter/ui/home_page/components/tabs/cup.dart';
import 'package:ly_flutter/ui/home_page/components/tabs/cutlery.dart';
import 'package:ly_flutter/ui/home_page/components/tabs/glasses.dart';
import 'package:ly_flutter/ui/home_page/components/tabs/other.dart';
import 'package:ly_flutter/ui/home_page/components/tabs/plate.dart';

class TabBarViewHome extends StatefulWidget{
  final TabController tabController;
  TabBarViewHome({Key? key, required this.tabController}) : super();

  @override
  _TabBarViewHomeState createState() => _TabBarViewHomeState();
}

class _TabBarViewHomeState extends State<TabBarViewHome>{

  @override
  Widget build(BuildContext){
    return TabBarView(
      controller: widget.tabController,
      children: [
        BodyTabBar(),
        PlateTab(),
        GlassesTab(),
        CupTab(),
        CutleryTab(),
        OtherTab(),
      ],
    );
  }
}
```

Рисунок 3.8 – Файл з керуванням зображення сторінок категорій у Home Page – tab_bar_view_home.dart

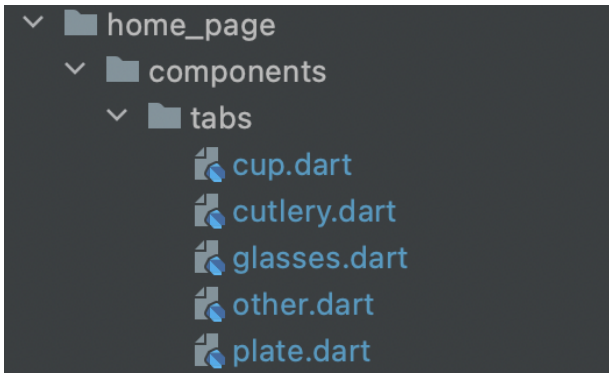


Рисунок 3.9 – Директорія у якій зберігаються файли окремих сторінок з категоріями товарів – tabs

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/data/product_lists.dart';
import 'package:ly_flutter/ui/details_page/details_page.dart';
import 'package:ly_flutter/ui/home_page/components/item_card.dart';

class CupTab extends StatelessWidget{
  const CupTab({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
        builder: (context, constraints){
          if (constraints.maxWidth < 600){
            return Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Expanded(
                  child: GridView.builder(
                      itemCount: cup.length,
                      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                        crossAxisCount: 2,
                        //mainAxisSpacing: kDefaultPadding,
                        //crossAxisSpacing: kDefaultPadding,
                        childAspectRatio: 0.80,
                      ),
                      itemBuilder: (context, index) => ItemCardTabsWeb(product: cup[index],
                        press: () => Navigator.push(context, MaterialPageRoute(builder: (context) =>
                            DetailsScreen(product: cup[index])
                        )
                        ),
                      )
                  ),
                )
              ],
            );
          } else {
            return Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Expanded(
                  child: GridView.builder(
                      itemCount: cup.length,
                      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
                        crossAxisCount: 4,
                        //mainAxisSpacing: kDefaultPadding,
                        //crossAxisSpacing: kDefaultPadding,
                        childAspectRatio: 0.80,
                      ),
                      itemBuilder: (context, index) => ItemCardTabsPhone(product: cup[index],
                        press: () => Navigator.push(context, MaterialPageRoute(builder: (context) =>
                            DetailsScreen(product: cup[index])
                        )
                        ),
                      )
                  ),
                )
              ],
            );
          }
        }
    );
  }
}
```

Рисунок 3.10 – Зразок файлу сторінки окремої категорії товару – cup.dart

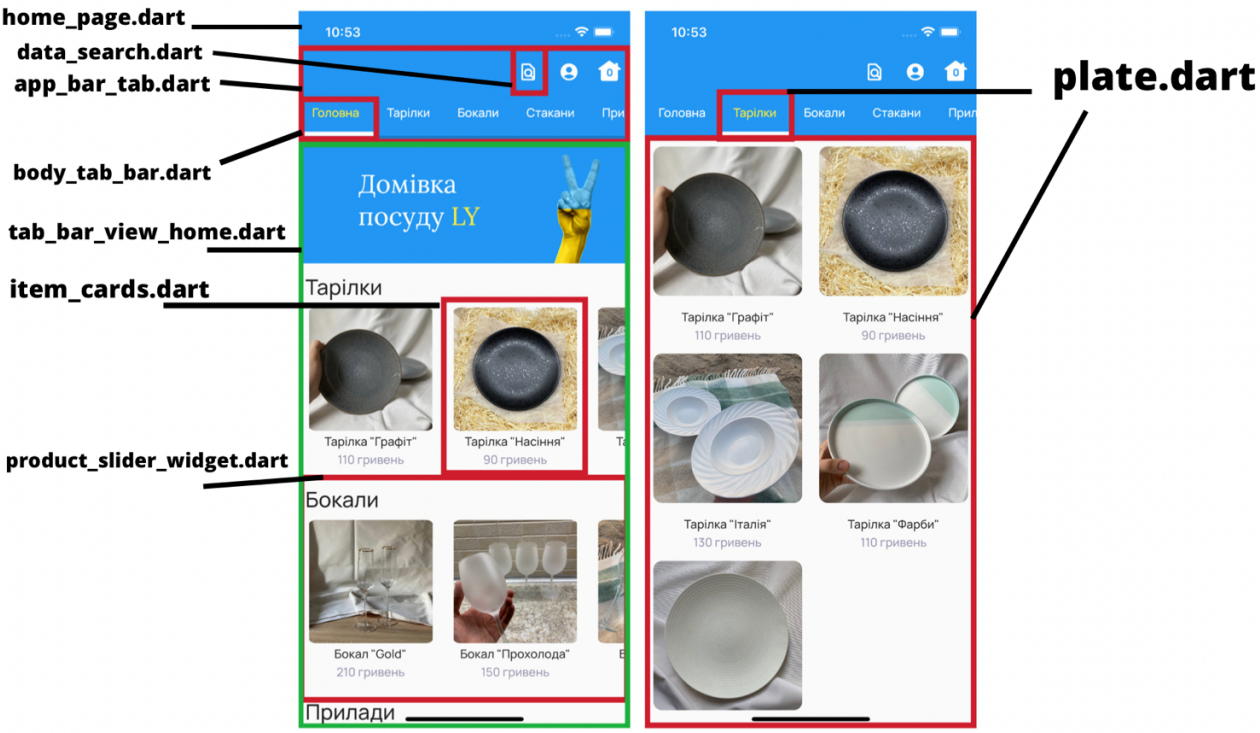
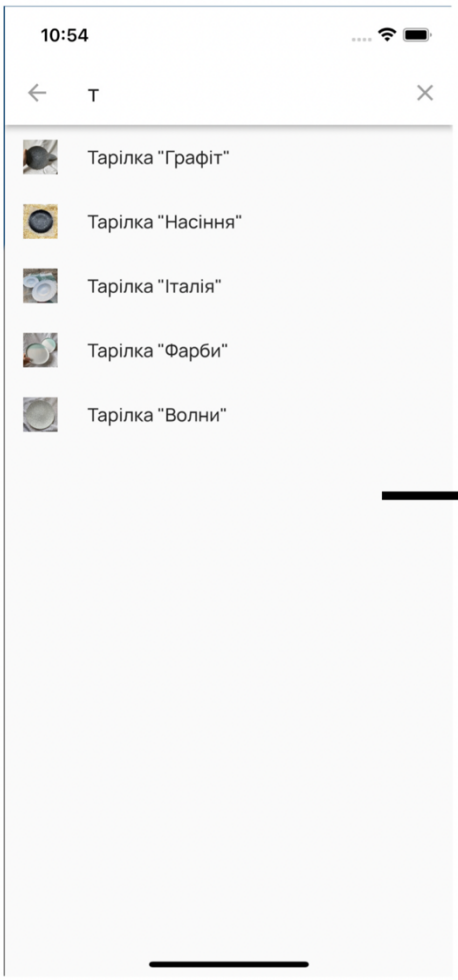


Рисунок 3.11 – Отриманий результат home_page на екрані



data_search.dart

Рисунок 3.12 – Віджет data_search на екрані

3.4.2 Детальний екран – **Details Screen**

Мета екрану – познайомити клієнта з детальною інформацією. Екран буде мати у собі:

- Додаткові фотографії
- Ціну
- Розмір
- Колір
- Опис

Крім того, як головні функції в ньому знаходяться дві кнопки для двох видів замовлення.

Перший – звичайний для користувачів, покласти товари у кошик. Після чого вони переходять, до кошика, де зможуть перейти на сторінку замовлення і надіслати контактні дані для надіслання.

Другий – для користувачів, які бажають зворотного зв‘язку, за більш детальною інформацією, або не витрачати час на заповнення і мати можливість оформити заявку у телефонному дзвінку.

На підставі отриманих даних використаємо наступні віджети:

StatelessWidget(), Scaffold(), AppBar(), Stack(), Column(), Row(), GestureDetector(), Container(), Expanded(), Text().

Для покращення архітектури, екран розділено на такі частини:

- Директорія details_page, можна побачити на рисунку 3.13.
- Головний файл – details_page.dart, можна побачити на рисунку 3.14.

Компоненти — директорія components:

- Файл головної частини екрана Details Page – body_details.dart, можна побачити на рисунку 3.15.
- Файл віджета додавання товарів до кошика – add_to_cart.dart.
- Лічильник кількості товарів – cart_counter.dart.
- Файл віджета зображення кольору і розміру – color_and_size.dart.
- Файл віджета зображення опису – description.dart.
- Файл віджета зображення для галереї детальних фото – images_gallery.dart.

- Файл віджета зображення і назви товару – product_title_with_image.dart.

Отриманий результат details_page на екрані, можна побачити на рисунку 3.16.

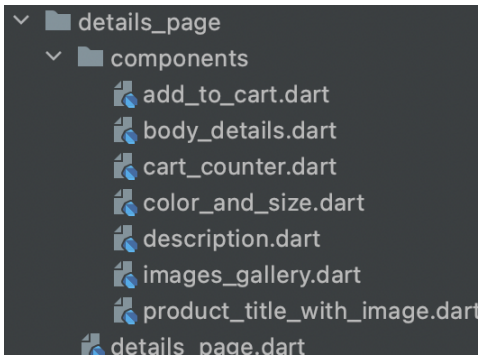


Рисунок 3.13 – Директорія details_page

```dart
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:ly_flutter/business/classes.dart';
import 'package:ly_flutter/business/provider.dart';
import 'package:ly_flutter/ui/authentication_page/authentication_page.dart';
import 'package:ly_flutter/ui/cart_page/cart_page.dart';
import 'package:ly_flutter/ui/custom_widgets/custom_search.dart';
import 'package:ly_flutter/ui/details_page/components/body_details.dart';
import 'package:ly_flutter/ui/home_page/components/data_search.dart';
import 'package:provider/provider.dart';

class DetailsPage extends StatelessWidget{
  final Product? product;

  const DetailsPage({Key? key, this.product}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return Scaffold(backgroundColor: Colors.blue, appBar: buildAppBar(context),body: Body(product: product),);
  }

  AppBar buildAppBar(BuildContext context) {
    return AppBar(backgroundColor: Colors.blue, elevation: 0, leading: IconButton(
        icon: const Icon(Icons.keyboard_arrow_left, size: 30),
        onPressed: () { Navigator.pop(context); context.read<CartData>().clearCount();}
      ),
      actions: <Widget>[IconButton(onPressed: () {
            customShowSearch(context: context, delegate: CustomDataSearch(hintText: 'Пошук..'));},
          icon: Icon(Icons.find_in_page_outlined)),
        IconButton(padding: const EdgeInsets.all(10.0),
          icon: const Icon(Icons.account_circle_rounded),
          onPressed: () => Navigator.push(context, MaterialPageRoute(
                  builder: (context) => const AuthenticationPage())),),
        Stack(
          alignment: Alignment.center,
            children:[IconButton(
                padding: const EdgeInsets.only(bottom: 4.0, top: 1.0, left: 5.0, right: 4.0),
                icon: const Icon(CupertinoIcons.house_alt_fill, size: 28),
                onPressed: () => Navigator.push(
                    context, MaterialPageRoute(
                        builder: (context) => const CartPage())),),
              Column(children: [
                  Expanded(child: Center(
                      child: Column(crossAxisAlignment: CrossAxisAlignment.center,
                        mainAxisAlignment: MainAxisAlignment.center,
                        children: [Container(
                              decoration: BoxDecoration(borderRadius: BorderRadius.circular(15), color:
Colors.white),
                              child: Padding(
                                padding: const EdgeInsets.symmetric(vertical: 2.0, horizontal: 6.0),
                                child: Text(context.watch<CartData>().getCartAmount.toString(), style: const
TextStyle(color: Colors.blue, fontSize: 12, fontWeight: FontWeight.bold),),)),],),),),],),]),],
    );
  }
}
```

Рисунок 3.14 – Файл детальної сторінки Details Page – details_page.dart

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/business/classes.dart';
import 'package:ly_flutter/data/constants.dart';
import 'package:ly_flutter/ui/details_page/components/add_to_cart.dart';
import 'package:ly_flutter/ui/details_page/components/cart_counter.dart';
import 'package:ly_flutter/ui/details_page/components/color_and_size.dart';
import 'package:ly_flutter/ui/details_page/components/description.dart';
import 'package:ly_flutter/ui/details_page/components/images_gallery.dart';
import 'package:ly_flutter/ui/details_page/components/product_title_with_image.dart';

class Body extends StatefulWidget {
  final Product? product;

  const Body({Key? key, this.product}) : super(key: key);

  @override
  State<Body> createState() => _BodyState();
}

class _BodyState extends State<Body> {
  @override
  Widget build(BuildContext context) {
    Size size = MediaQuery.of(context).size;
    return SingleChildScrollView(
      physics: const ClampingScrollPhysics(),
        child: Column(children: <Widget>[
          SizedBox(child: Stack(
              children: <Widget>[Container(
                  margin: const EdgeInsets.only(top: 260),
                      padding: EdgeInsets.only(top: size.height * 0.03, left: kDefaultPadding,
                      right: kDefaultPadding, bottom: 70,),
                    decoration: const BoxDecoration(color: Colors.white, borderRadius: BorderRadius.only(
                      topLeft: Radius.circular(24), topRight: Radius.circular(24)),),
                    constraints: const BoxConstraints(maxHeight: double.infinity, minHeight: 550),
                    child: Column(children: <Widget>[
                      ColorAndSize(product: widget.product),
                      (widget.product!.images == null ? Container(
                        margin: const EdgeInsets.symmetric(vertical: 30),
                        child: const Text('Немає детальних фото', style: TextStyle(color: Colors.blueAccent,
fontSize: 20),),)
                          : SizedBox(
                          height: 200, width: double.infinity,
                          child: ListView.builder(itemBuilder: (context, index) => SizedBox(
                              width: 200, height: 150, child: ImagesGallery(image: widget.product!.images![index])),
                          itemCount: widget.product!.images!.length, scrollDirection: Axis.horizontal,))),
                      Description(product: widget.product),
                      CartCounter(),
                      AddToCart(product: widget.product)
                  ],),),
                ProductTitleWithImage(product: widget.product),],)),],));}}
```

Рисунок 3.15 – Файл головної частини екрана Details Page – body_details.dart

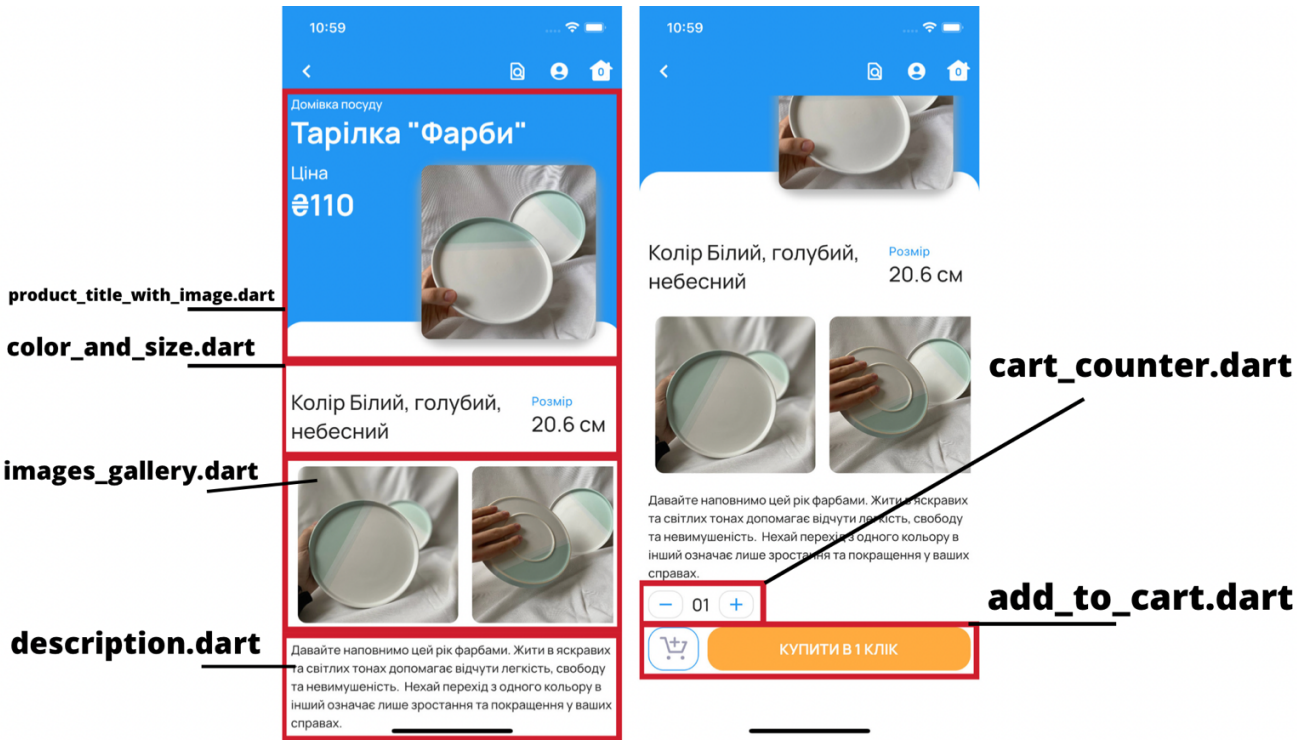


Рисунок 3.16 – Отриманий результат details_page на екрані

3.4.3 Екран кошика – Cart Page

Вищевикладене зазначає те, що у користувача повинен бути екран, на етапі якого він може:

1.Побачити який товар і в якій кількості його додано до кошика

2.Видалити товар з кошика

3.Бачити загальну суму кошика

4.Перейти до сторінки замовлення

На підставі отриманих даних, використаємо наступні віджети:

StatelessWidget(), Scaffold(), AppBar(), Column(), Row(), Container(), Expanded(), ListView.Builder(), Padding(), Text(), використовуючи SnackBar() з запрограмованою перевіркою на наявність товарів у кошику, користувача буде зупиняти додаток, якщо кошик пустий і перейти до замовлення неможливо.

Для покращення архітектури, екран розділено на такі частини:

Директорія cart_page, можна побачити на рисунку 3.17.

Головний файл – cart_page.dart, можна побачити на рисунку 3.18.

Компоненти — директорія components:

- Файл головної частини екрана – body_cart.dart, можна побачити на рисунку 3.19
- Файл з інформацією про загальну суму кошика і кнопка переходу до замовлення cart_bottom.dart
- Файл з фоном для карти замовлення cart_card_background.dart
- Файл з зображенням товар на карті замовлення cart_card_image.dart
- Файл з інформацією на карті замовлення cart_card_info.dart
- Файл що будує список карт замовлень cart_cards_builder.dart

Отриманий результат cart_page на екрані, можна побачити на рисунку 3.20.

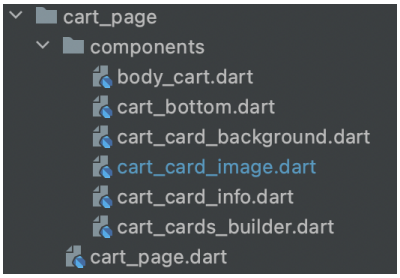


Рисунок 3.17 – Директорія cart_page

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/ui/cart_page/components/cart_bottom.dart';
import 'package:ly_flutter/ui/cart_page/components/cart_cards_builder.dart';

class BodyCartPage extends StatelessWidget {
  const BodyCartPage({Key? key,}) : super(key: key);

  Future errorCart() async {
    return const SnackBar(content: Text('Корзина пуста!'),);}

  @override
  Widget build(BuildContext context) {
    return Column(children: [
        CartListBuilder(),
        CartBottom(),],);}}
```

Рисунок 3.18 – Головний файл – cart_page.dart

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/business/provider.dart';
import 'package:ly_flutter/ui/order_page/order_page.dart';
import 'package:provider/provider.dart';

class CartBottom extends StatelessWidget{

  @override
  Widget build(BuildContext context){
    return Container(
      height: 110, decoration: const BoxDecoration(color: Colors.blue),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [Padding(
            padding: const EdgeInsets.only(left: 15, top: 10),
            child: Container(
                padding: const EdgeInsets.all(5),
                decoration: BoxDecoration(color: Colors.orangeAccent, borderRadius: BorderRadius.circular(10)),
                child: Text(
                  'Загальна сума: ${context.watch<CartData>().totalCartPrice}', style: const TextStyle(color:
Colors.white, fontSize: 20, fontWeight: FontWeight.bold),)),
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.end,
            children: [
              GestureDetector(
                onTap: () {
                  if (context.read<CartData>().totalCartPrice > 0) {
                    Navigator.of(context).push(MaterialPageRoute(
                        builder: (context) => OrderPage()));
                  } else {
                    ScaffoldMessenger.of(context).showSnackBar(
                        const SnackBar(content: SizedBox(height: 50,
                          child: Text('Корзина пуста!',
                            style: TextStyle(fontSize: 15),),),));}},
                child: Container(
                  padding: const EdgeInsets.all(14),
                  decoration: BoxDecoration(color: MaterialStateColor.resolveWith((states) => Colors.white),
                    borderRadius: const BorderRadius.only(
                      topLeft: Radius.circular(10),
                      bottomLeft: Radius.circular(10),),),
                  child: const Text('Оформлення', style: TextStyle(color: Colors.blue, fontWeight: FontWeight.bold,
fontSize: 20),),),),],),],),);}}
```

Рисунок 3.19 – Файл головної частини екрана – body_cart.

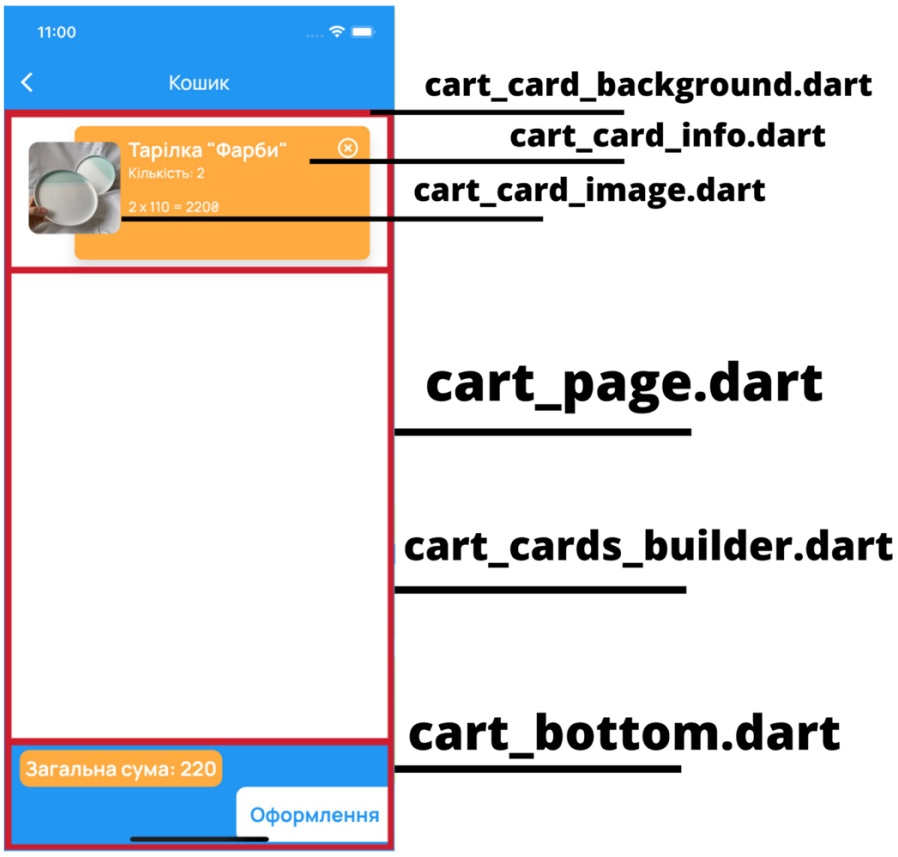


Рисунок 3.20 – Отриманий результат cart_page на екрані

3.4.4 Екран замовлення – Order Page

Слід за кошиком, необхідно розробити сторінку замовлення, за допомогою якої користувач зможе надіслати свої контактні дані для надіслання замовлення.

А саме:

1.ПІБ

2.Номер телефона

3.Місто

4.Номер адреси Нової пошти

Сторінка повинна мати зручний інтерфейс, який сприяє до швидкого замовлення.

Для досягнення мети, використаємо наступні віджети:

StatefulWidget() для то, щоб зберігати в ньому інформацію яку надасть користувач у віджеті DropDownList(), Scaffold(), AppBar(), Column(), Row(), Container(), Expanded(), Lottie(), SizedBox(), Text(), TextFormField() – віджет для заповнення інформації, з можливістю перевірки за допомогою функції validate.

Екран розділено на такі частини:

Директорія order_page, можна побачити на рисунку 3.21.

Головний файл – order_page.dart, можна побачити на рисунках 3.22, 3.23.

Отриманий результат order_page на екрані, можна побачити на рисунку 3.24.

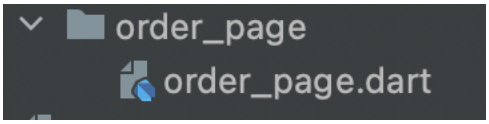


Рисунок 3.21 – Директорія order_page

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:lottie/lottie.dart';
import 'package:ly_flutter/business/classes.dart';
import 'package:ly_flutter/business/provider.dart';
import 'package:ly_flutter/ui/home_page/home_page.dart';
import 'package:provider/provider.dart';

class OrderPage extends StatefulWidget{

  @override
  _OrderPageState createState() => _OrderPageState();
}

class _OrderPageState extends State<OrderPage> with SingleTickerProviderStateMixin{
  final storage = FirebaseFirestore.instance;
  late CollectionReference<ProductOrder> _productOrder;
  late AnimationController controller;
  TextEditingController postController = TextEditingController();
  String name = '';
  String number = '';
  String city = '';

  final formKey = GlobalKey<FormState>();
  final List<String> items = ['№ 1'];
  String selectedItem = '№ 1';
  int userPostCount = 2;

  @override
  void initState() {
    super.initState();
    _productOrder = FirebaseFirestore.instance.collection('productOrder').withConverter<ProductOrder>(
        fromFirestore: (snapshot, _) => ProductOrder.fromJson(snapshot.data()!),
        toFirestore: (productOrder, _) => productOrder.toJson());

    controller = AnimationController(
        duration:  Duration(milliseconds: 700),
        vsync: this
    );

    controller.addStatusListener((status) async {
      if (status == AnimationStatus.completed){
        Navigator.push(context, MaterialPageRoute(builder: (context) => HomePage()));
        controller.reset();
      }
    });
  }

  @override
  void dispose() {
    controller.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context){
    while(items.length != 100){
      items.add('№ ' + userPostCount.toString());
      userPostCount++;
    }
    return Stack(children: [
      Image.asset(
          'assets/ukraine_order.png',
          height: MediaQuery.of(context).size.height,
          width: MediaQuery.of(context).size.width,
          fit: BoxFit.cover
      ),
      Scaffold(
        backgroundColor: Colors.transparent,
        body: SingleChildScrollView(
          child: Container(
            padding: const EdgeInsets.symmetric(vertical: 30, horizontal: 20),
            child: Form(
              key: formKey,
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [const Text('Одержувач замовлення', style: TextStyle(fontWeight: FontWeight.bold,
fontSize: 30)),
                  const SizedBox(height: 30),
                  const Text("Ім'я та Прізвище", style: TextStyle(fontWeight: FontWeight.bold, fontSize: 20)),
                  const SizedBox(height: 5),
                  SizedBox(
                    width: 300,
                    child: Column(
                      children: [
                        TextFormField(
                          decoration: const InputDecoration(
                              hintText: 'Ладон Скляров',
                              border: OutlineInputBorder()
                          ),
                          validator: (value) {
                            if (value != null && value.length < 7){
                              return "Введіть ім'я та прізвище";
                            } else {
                              return null;
                            }
                          },
```

Рисунок 3.22 – Головний файл – order_page.dart

Рисунок 3.23 – Головний файл – order_page.dart

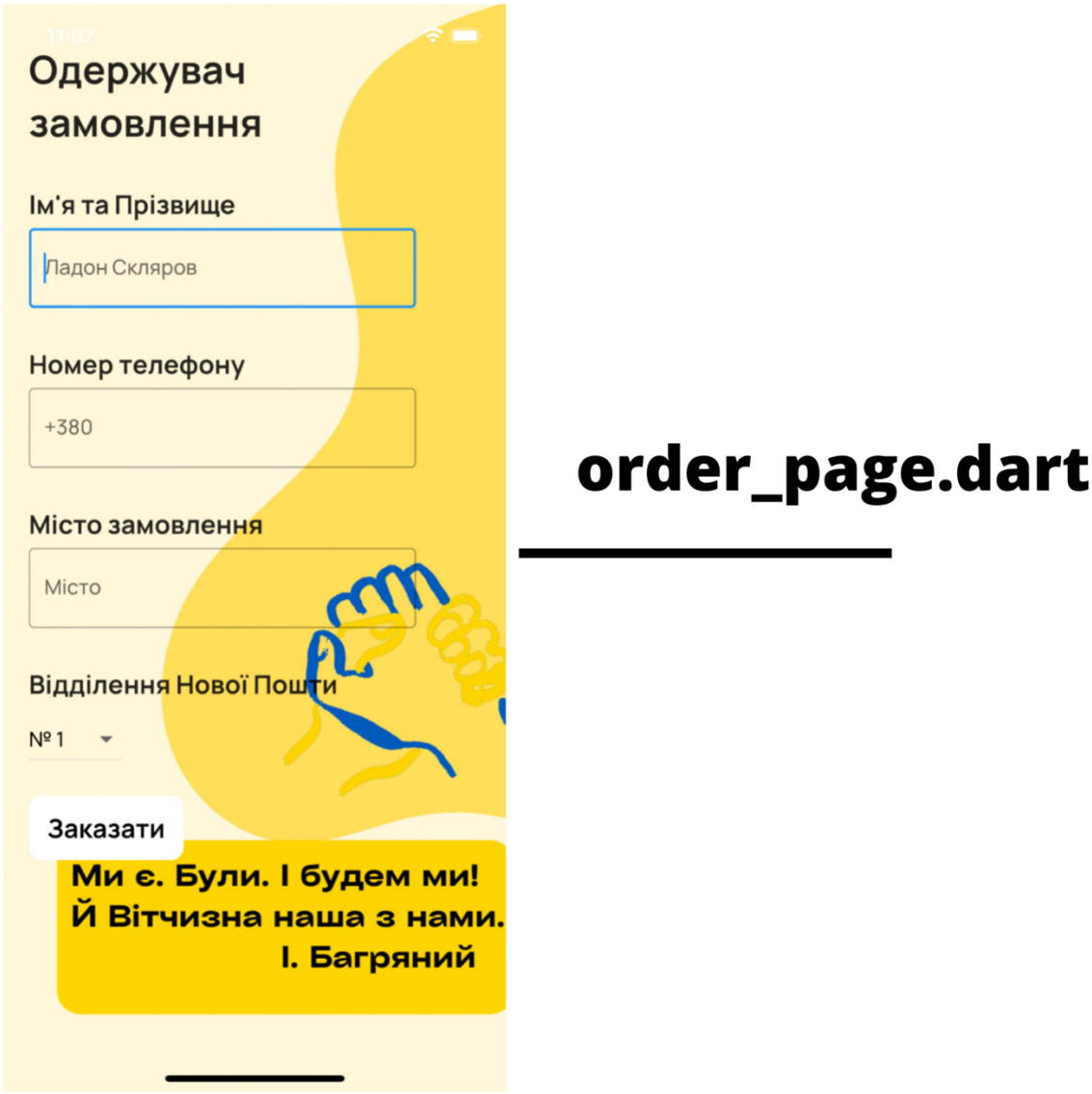


Рисунок 3.24 – Отриманий результат order_page на екрані

3.4.5 Екран аутентифікації – Authentication Page

Власник повинен мати можливість керувати замовленнями зі сторінки адміністратора.

Досягаючи цієї мети, буде розроблено екран аутентифікації, на початку що має функцію перенаправлення адміністратора на його сторінку, за окремим логіном і паролем. У майбутньому ця функція може мати додаткові можливості для клієнта за допомогою його авторизації.

Необхідно розробити інтерфейс для входу за наявними даними та для реєстрації нового користувача.

Для досягнення мети, використаємо наступні віджети:

StatefulWidget() для того, щоб зберігати в ньому інформацію яку надасть користувач у віджеті Scaffold(), AppBar(), Column(), Row(), Container(), Expanded(), SizedBox(), Text(), TextFormField() – віджет для заповнення інформації, з можливістю перевірки за допомогою функції validate.

Для покращення архітектури, екран розділено на такі частини:

Директорія authentication_page, можна побачити на рисунку 3.25.

Головний файл – authentication_page.dart, можна побачити на рисунку 3.26, 3.27.

Отриманий результат authentication_page на екрані, можна побачити на рисунку 3.28.

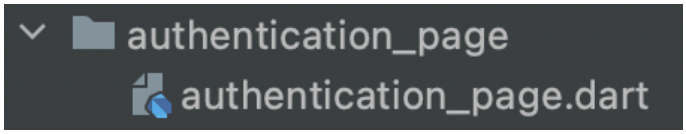


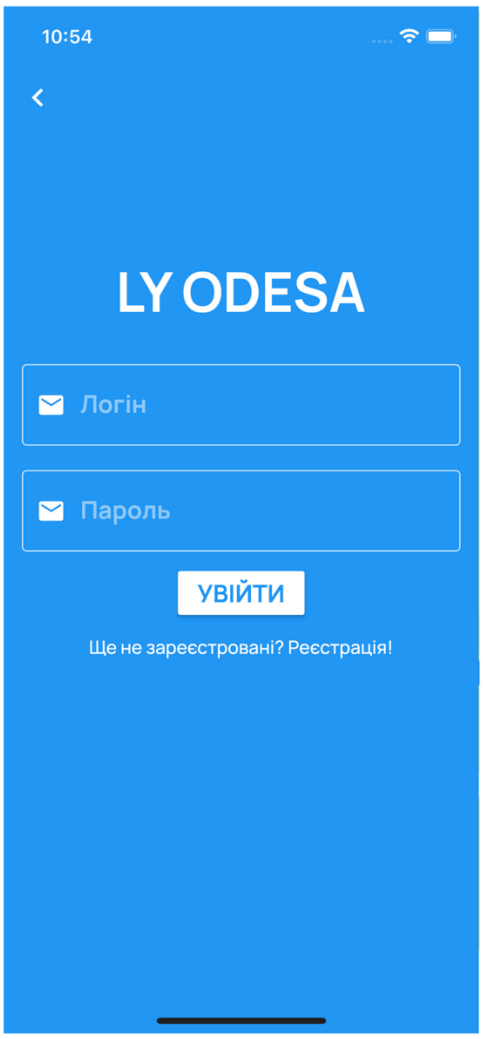
Рисунок 3.25 – Директорія authentication_page

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:ly_flutter/ui/admin_page/admin_page.dart';
import 'package:ly_flutter/ui/home_page/home_page.dart';


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


  @override
  _AuthenticationPageState createState() => _AuthenticationPageState();
}

class _AuthenticationPageState extends State<AuthenticationPage> {

  Future<FirebaseApp> _initializeFirebase() async {
    FirebaseApp firebaseApp = await Firebase.initializeApp();
    return firebaseApp;
  }


  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();

  static Future<User?> loginUsingEmailPassword({required String email, required String password, required
BuildContext context})async{
    FirebaseAuth _auth = FirebaseAuth.instance;
    User? user;
    try{
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(email: email, password: password);
      user = userCredential.user;
    } on FirebaseAuthException catch (e){
      if(e.code == "user-not-found"){
        print("Користувач з таким email не знайден");
      }
    }
    return user;
  }

  static Future<User?> registerInWithEmailAndPassword({required String email, required String password, required
BuildContext context})async {
    User? user;
    try {
      UserCredential userCredential = await FirebaseAuth.instance
          .createUserWithEmailAndPassword(email: email, password: password);
    } on FirebaseAuthException catch (e) {
      if (e.code == 'weak-password') {
        print('The password provided is too weak.');
      } else if (e.code == 'email-already-in-use') {
        print('The account already exists for that email.');
      }
    } return user;
  }

  bool showLogin = true;

  Widget _button(String text, void func()){
    return RaisedButton(
      splashColor: Theme.of(context).primaryColor,
      highlightColor: Theme.of(context).primaryColor,
      color: Colors.white,
      child: Text(text, style: TextStyle(fontWeight: FontWeight.bold, color: Theme.of(context).primaryColor,
fontSize: 20)),
      onPressed: () {
        func();},);}

  Widget _logo(){
    return const Padding(
        padding: EdgeInsets.only(top: 100),
        child: Align(
            child: Text('LY ODESA', style: TextStyle(fontSize: 45, fontWeight: FontWeight.bold, color:
Colors.white))
        )
    );
  }

  Widget _form(String label, void func()){
    return Column(children: <Widget>[
            Container(
              margin: EdgeInsets.symmetric(horizontal: 15, vertical: 10),
              child: TextField(
                 controller: _emailController,
                 style: TextStyle(fontSize: 20, color: Colors.white),
                 decoration: InputDecoration(
                   hintText: 'Логін',
                   hintStyle: TextStyle(fontWeight: FontWeight.bold, fontSize: 20, color:
Colors.white.withOpacity(0.5)),
                  focusedBorder: OutlineInputBorder(
                  borderSide: BorderSide(color: Colors.white, width: 2),),
                  enabledBorder: OutlineInputBorder(
                     borderSide: BorderSide(color: Colors.white.withOpacity(0.8), width: 1)),
                 prefixIcon: Padding(
                   padding:  EdgeInsets.only(left: 10, right: 10),
                   child: IconTheme(
                       data: IconThemeData(color: Colors.white),
                       child: Icon(Icons.mail)),),)),),
```

Рисунок 3.26 – Головний файл – authentication_page.dart

```
          Container(
            margin: EdgeInsets.symmetric(horizontal: 15, vertical: 10),
            child: TextField(
              obscureText: false,
              controller: _passwordController,
                style: TextStyle(fontSize: 20, color: Colors.white),
                decoration: InputDecoration(
                  hintText: 'Пароль',
                  hintStyle: TextStyle(fontWeight: FontWeight.bold, fontSize: 20, color:
Colors.white.withOpacity(0.5)),
                  focusedBorder: OutlineInputBorder(
                    borderSide: BorderSide(color: Colors.white, width: 2),
                  ),
                  enabledBorder: OutlineInputBorder(
                      borderSide: BorderSide(color: Colors.white.withOpacity(0.8), width: 1)
                  ),
                  prefixIcon: Padding(
                    padding:  EdgeInsets.only(left: 10, right: 10),
                    child: IconTheme(
                        data: IconThemeData(color: Colors.white),
                        child: Icon(Icons.mail)),),)),),
          _button(label, func)]);}

  Future<void> _buttonActionSign() async {
    User? user = await loginUsingEmailPassword(email: _emailController.text, password: _passwordController.text,
context: context);
    print(user);
    if(_emailController.text == 'ceoly@gmail.com'){
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=> OrdersOwner()));
    } else if(user != null){
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=> const HomePage()));
    }
    _emailController.clear();
    _passwordController.clear();
  }

  void backFunc(){
    Navigator.push(context, MaterialPageRoute(builder: (context) => AuthenticationPage()));
  }

  Future<void> _buttonActionRegister() async {
    User? user = await registerInWithEmailAndPassword(email: _emailController.text, password:
_passwordController.text, context: context);
    if (user == null){
      backFunc();
    }
    _emailController.clear();
    _passwordController.clear();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(backgroundColor: Theme.of(context).primaryColor,
        appBar: AppBar(
          backgroundColor: Colors.transparent, elevation: 0,
          leading: IconButton(icon: const Icon(Icons.keyboard_arrow_left, size: 30,), onPressed: ()
{Navigator.pop(context);},)
        ),
        body: FutureBuilder(
            future: _initializeFirebase(), builder: (context, snapshot){
              if (snapshot.connectionState == ConnectionState.done){
                return SingleChildScrollView(
                  child: Column(children: <Widget>[
                      _logo(),
                      const SizedBox(height: 20),
                      (showLogin
                          ? Column( children: <Widget>[
                          _form('УВІЙТИ', _buttonActionSign),
                          Padding( padding: const EdgeInsets.all(10),
                              child: GestureDetector(
                                child: const Text('Ще не зареєстровані? Реєстрація!', style: TextStyle(fontSize: 15,
color: Colors.white)),
                                onTap:() {setState((){
                                    showLogin = false;
                                  });},)),],)
                          :
                      Column(children: <Widget>[
                          _form('РЕЄСТРАЦІЯ', _buttonActionRegister),
                          Padding(
                              padding: const EdgeInsets.all(10),
                              child: GestureDetector(
                                child: const Text('Готові реєструватись? Уперед!', style: TextStyle(fontSize: 15,
color: Colors.white)),
                                onTap:() {
                                  setState((){ showLogin = true;
                                  });},)),],),);}
              return const Center(child: CircularProgressIndicator(),);}));}}
```

Рисунок 3.27 – Головний файл – authentication_page.dart



authentication_ page.dart

Рисунок 3.28 – Отриманий результат authentication_page на екрані

3.4.6 Екран адміністратора – Admin Page

За вищевикладеною метою проекту, необхідно розробити зручний інтерфейс для перегляду замовлень, який відповідає умовам наявності двох видів замовлень:

- Замовлення з повними контактними даними.

- Швидкі замовлення з не повним обсягом даних.

На підставі отриманих даних, використаємо наступні віджети:

StatefulWidget(), Scaffold(), AppBar(), DefaultTabController(), PreferredSize(), TabBar(), TabBar(), Column(), Row(), GestureDetector(), Expanded(), ListView.Builder(), Padding(), Text(), StreamBuilder() – за допомогою цього

віджета програмі надається потік даних з Firebase, користуватись колекціями якого сторінка буде за допомогою CustomExpansionTile() та ListView().

Для покращення архітектури, екран розділено на такі частини:

Директорія admin_page, можна побачити на рисунку 3.29.

Головний файл екрана – admin_page.dart, можна побачити на рисунку 3.30.

Компоненти – директорія components:

- Файл 1 частини екрана зі звичайними замовленнями – admin_order.dart
- Файл 2 частини екрана зі швидкими замовленнями – admin_speed_order.dart
- Файл з фоновою частиною картки звичайного замовлення – background_order_card.dart
- Файл з карткою для зображення інформації звичайного замовлення – order_card.dart
- Файл з карткою для зображення інформації швидкого замовлення – speed_order_card.dart

Отриманий результат admin_page на екрані, можна побачити на рисунку 3.31.

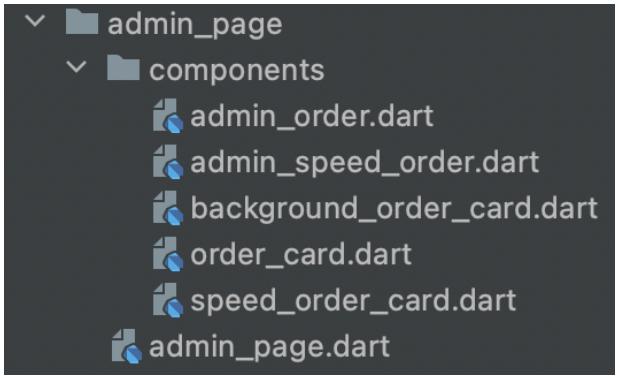


Рисунок 3.29 – Директорія admin_page

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/data/constants.dart';
import 'package:ly_flutter/ui/admin_page/components/admin_order_page.dart';
import 'package:ly_flutter/ui/admin_page/components/admin_speed_order_page.dart';

class OrdersOwner extends StatefulWidget {

  @override
  State<OrdersOwner> createState() => _OrdersOwnerState();
}

class _OrdersOwnerState extends State<OrdersOwner> with SingleTickerProviderStateMixin {
  late TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = TabController(vsync: this, length: 2);
  }

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 2,
      child: Scaffold(
          appBar: PreferredSize(
            preferredSize: const Size.fromHeight(100),
            child: AppBar(backgroundColor: Color(hexcolor('#E38C4C')),
               bottom: TabBar(
                 controller: _tabController,
                 isScrollable: true,
                 indicatorWeight: 5.0,
                 indicatorColor: Colors.white,
                 unselectedLabelColor: Colors.grey,
                 tabs: [
                   Text('Замовлення'),
                   Text('Швидкі замовлення')
               ],
              ),
            ),
          ),
          backgroundColor: Color(hexcolor('#977B60')),
          body: TabBarView(
            controller: _tabController,
              children: [
                AdminOrderPage(),
                AdminSpeedOrderPage()
              ]
          ),
      ),
    );
  }
}
```

Рисунок 3.30 – Головний файл екрана – admin_page.dart

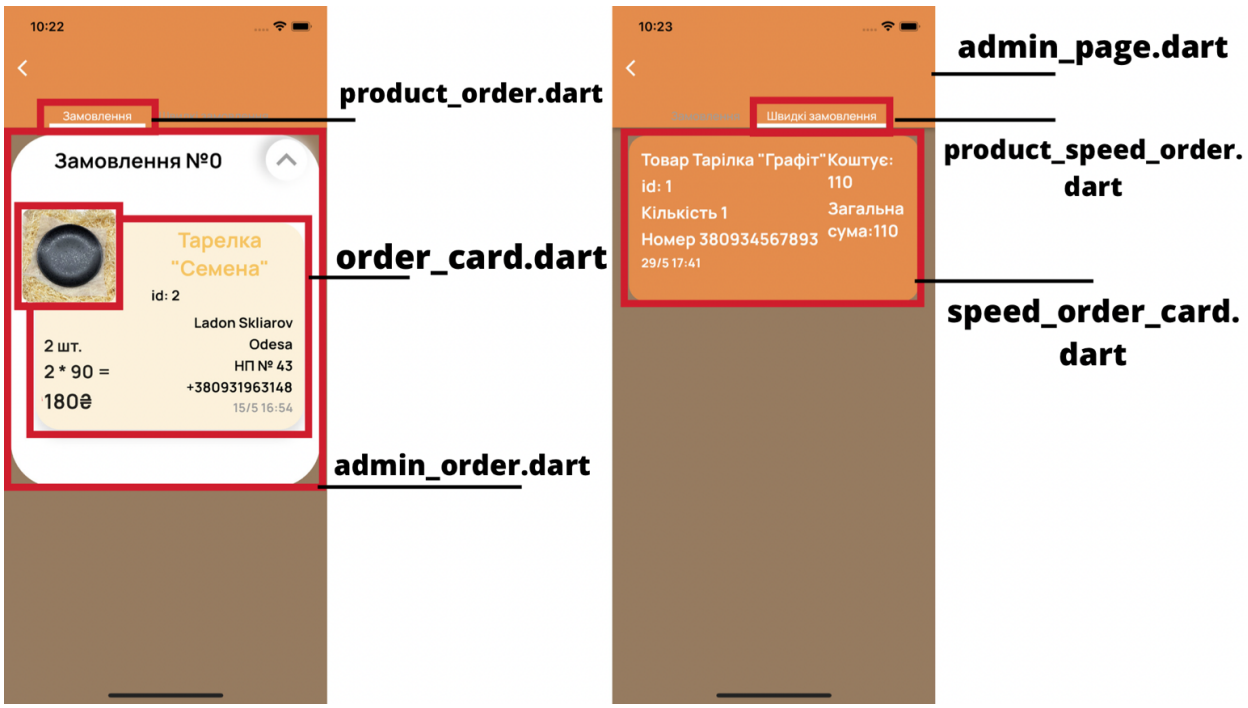


Рисунок 3.31 – Отриманий результат admin_page на екрані

3.5 Розробка стану додатку у Provider та підключення його до інтерфейсу

Необхідне рішення, яке буде з‘єднувати інформацію, котра поступає зі всього додатка і надається всьому додатку.

Provider – це модель State Management, яка заснована розробником Ремі Русле, який не являє собою офіційного представника Flutter/Dart але активно підтримує платформу додаючи до неї свої рішення, які зазнають у своїх відео офіційні розробники від Google. Один з таких – Provider.

Він заснований на принципах InheritedWidget але покращений до швидкого і зручного доступу до State Management.

Коли у моделі Inherited Widget необхідно 4 об'єкти, щоб підтримувати взаємодію між його інформацією, у Provider необхідні лише 2 об‘єкти:

- Необхідно спадкувати клас з інформацією від ChangeNotifierProvider

- Для використовування інформації цього класу, розробити наступний рядок кода:

Для читання даних

context.watch<CartData>.”взаємодія”

Для редагування даних

context.read<CartData>.”взаємодія”

Створимо файл provider.dart, користувачі взаємодіють з більшістю функціями та параметрами для замовлення, які будуть зберігатися у provider.dart. До Provider надходять дані, які клієнт заповнює для замовлення — доданий товар і його кількість, інформацію про контактні дані для доставлення клієнта, список звичайних, швидких замовлень.

Отримаємо наступні, необхідні функції:

getTime – отримати параметри часу в Provider

getSpeedOrder – отримати об‘єкт швидке замовлення до списку

getOrder – отримати об‘єкт звичайного замовлення до списку

updateCost – оновити параметри вартості

getPostInfo – отримати контактні дані клієнта в Provider

addItemCart – додати товари з детальної сторінки до кошика

void removeItemCart – видалити товар з кошика

void minusItemCount – зменшити кількість обраного товару на 1

void plusItemCount – додати кількість обраного товару на 1

void updateItemCount – оновити кількість обраних товарів до 1

Створений файл provider.dart, можна побачити на рисунку 3.32.

```dart
import 'package:flutter/material.dart';
import 'package:ly_flutter/business/classes.dart';

class CartData with ChangeNotifier {
  List<ProductOrder> cartOwner = [];
  List<ProductCart> cartCart = [];
  List<Product> cart = [];
  List<ProductSpeedOrder> cartSpeedOrder = [];
  int price = 0;
  int amountItem = 1;
  int get getAmount => amountItem;
  int totalCartPrice = 0;
  int totalPrice = 1;
  int priceBuy = 0;
  int get getCartAmount => cartCart.length;
  int dayTime = DateTime.now().day;
  int monthTime = DateTime.now().month;
  int hourTime = DateTime.now().hour;
  int minuteTime = DateTime.now().minute;
  int orderNumber = 0;
  String name = 'Aleksandr Skliarov';
  String city = 'Default';
  String phoneNumber = '380931963148';
  String post = "Default";

  void getTime(){
    dayTime = DateTime.now().day; monthTime = DateTime.now().month;
    hourTime = DateTime.now().hour; minuteTime = DateTime.now().minute;
    notifyListeners();}

  getSpeedOrder({
    required int number, required String title,
    required int id, required int amountItems,
    required int price,}){
    cartSpeedOrder.add(
        ProductSpeedOrder(userPhoneNumber: number, title: title, id: id, amountItems: amountItems,
            dayTime: dayTime, monthTime: monthTime, hourTime: hourTime, minuteTime: minuteTime,
            price: price, totalPrice: amountItems * price,));
    notifyListeners();}

  getOrder(){ dayTime = DateTime.now().day; monthTime = DateTime.now().month;
    hourTime = DateTime.now().hour; minuteTime = DateTime.now().minute;
    cartOwner.add(ProductOrder(order: cartCart,
        orderNumber: orderNumber, dayTime: dayTime,
        monthTime: monthTime, hourTime: hourTime,
        minuteTime: minuteTime, name: name,
        post: post, city: city, phoneNumber: phoneNumber));
    notifyListeners();}

  updateCost(ProductCart item){item.totalPrice = item.price * amountItem;}

  getPostInfo({required String name, required String city, required String post, required String phoneNumber}){
    this.name = name; this.city = city;
    this.post = post; this.phoneNumber = phoneNumber;
    notifyListeners();}

  addItemCart(Product item){totalPrice = item.price * amountItem;
    totalCartPrice += totalPrice;
    cartCart.add(ProductCart(id: item.id, price: item.price, title: item.title, image: item.image, description:
item.description, color: item.color, images: [''], size: item.size, amountItems: amountItem, totalPrice:
totalPrice));
    notifyListeners();}

  removeItemCart(int index){totalCartPrice -= cartCart[index].totalPrice;
    cartCart.removeAt(index); notifyListeners();}

  minusItemCount(){if(amountItem > 1) {amountItem -= 1;} notifyListeners();}

  plusItemCount(){if(amountItem < 100) {amountItem += 1;}notifyListeners();}

  updateItemCount(){amountItem = 1; notifyListeners();}}
```

Рисунок 3.32 – Файл Provider з бізнес-логікою

3.6 Створення проекту Firebase, баз даних та сервісу аутентифікації

Для створення баз даних Firebase, необхідно перейти на головну сторінку офіційного сайту Firebase, авторизуємося за допомогою Gmail, натискаємо створити проект, можна побачити на рисунку 3.33.

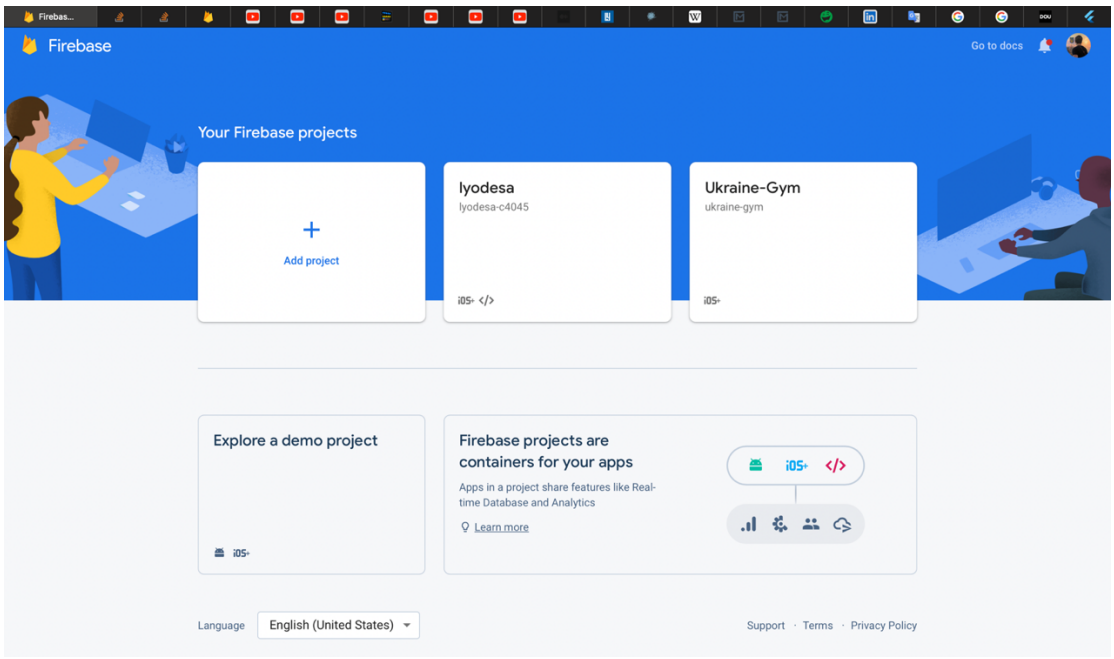


Рисунок 3.33 – Головна сторінка Firebase

На головній сторінці проекту, обираємо до якого типу нашого додатка ми приєднаємо наш застосунок IOS, Android, Web. У нашому випадку Web. Меню можна побачити на рисунку 3.34.

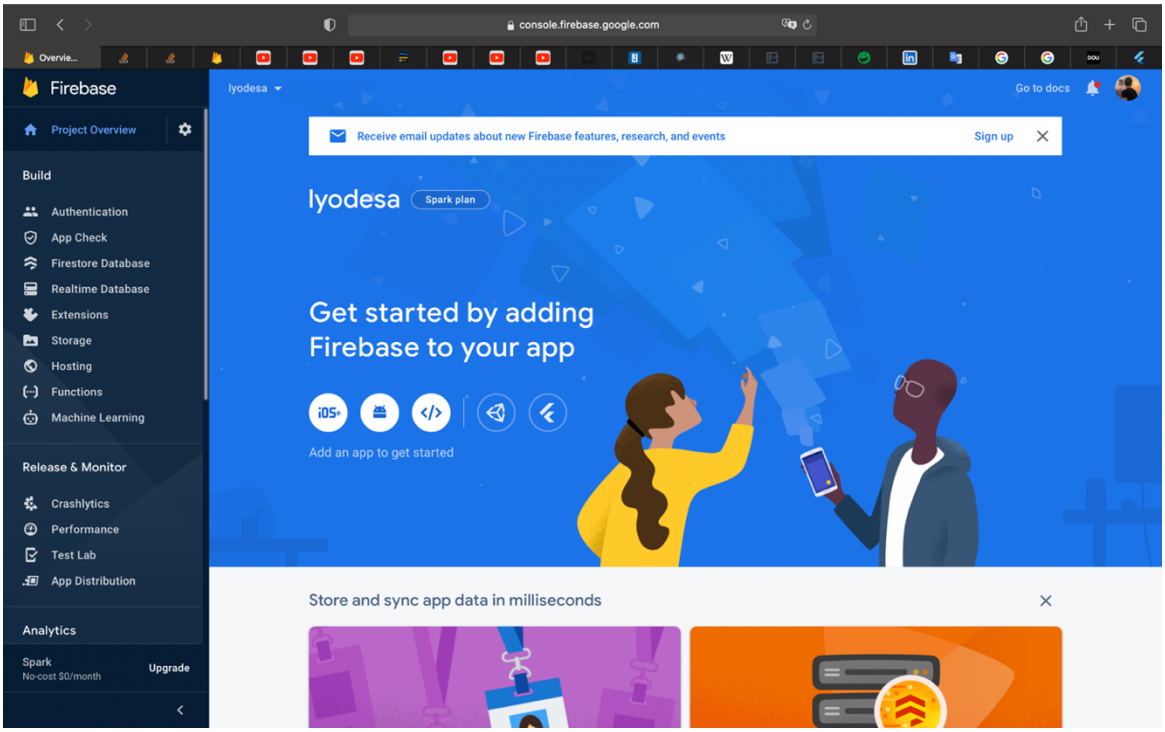


Рисунок 3.34 – Сторінка створеного проекту в Firebase

1.Створюємо назву проекта.

2.Вмикаємо сервіси Google Anlytics(Гугл аналітики) щоб мати можливість у майбутньому відстежувати аудиторію користувачів.

3.Створюємо йому акаунт який буде бачити аналітику, або обираємо вже наявний, який приєднаний до іншого додатка.

Погоджуємось з правилами Google Analytics.

Створення та реєстрацію проекту закінчено.

Ліворуч на сайті – функції, які використовують для додатка. Обираємо Firebase Database для створення бази даних.

1.Натискаємо ‘Get Started’, обираємо за якими умовами будуть використовуватися бази даних Firebase, у тестовому режимі або у режимі комерційної взаємодії.

2.Обираємо сервер в якому регіоні буде зберігатися наша база даних.

Базу даних створено.

Приєднаємо базуданих аутентифікації користувачів, яку побачимо також на панелі ліворуч.

1.Натискаємо ‘Get Started’.

2.На панелі у Sign-in-method обираємо за допомогою якого сервісу аутентифікувати користувачів, використаємо Email/Password.

Усі сервіси безкоштовні, за виключенням SMS.

3.Після чого переходимо за цією функцією та одразу приєднаємо до бази акаунт та пароль, за яким будемо перенаправляти адміністратора на його сторінку.

Ми закінчили створення баз даних.

3.7 Підготовка мобільного додатку до підключення з Firebase.

3.7.1 Створення конверторів JSON-даних

Для того, щоб використовувати дані, які програма отримує з бази даних Firebase, їх необхідно конвертувати з JSON-формату в якому вони зберігаються,

у вигляд звичайних для програми змінних. У випадку Dart, це: Number(int, double), String, List, Map, Bool, Null.

Необхідні нам конвертори даних, являють собою factory-конструктори, які необхідно встановити в рамках класу для якого вони створюються. Factory-конструктори використовують для класів, щоб повертати й використовувати вже наявний об'єкт, а не тільки створювати новий як роблять класи в Dart. Створені конвертори даних можна побачити на рисунках 3.35, 3.36.

```dart
class ProductSpeedOrder{
  final int userPhoneNumber;
  final String title;
  final int id;
  final int amountItems;
  final int dayTime;
  final int monthTime;
  final int hourTime;
  final int minuteTime;
  final int price;
  final int totalPrice;


  ProductSpeedOrder({
    required this.userPhoneNumber,
    required this.title,
    required this.id,
    required this.amountItems,
    required this.dayTime,
    required this.monthTime,
    required this.hourTime,
    required this.minuteTime,
    required this.price,
    required this.totalPrice
  });

  factory ProductSpeedOrder.fromJson(Map<String, dynamic> json){
    return ProductSpeedOrder(
        userPhoneNumber: json['number'] as int,
        title: json['title'] as String,
        id: json['id'] as int,
        amountItems: json['amountItems'] as int,
        dayTime: json['dayTime'] as int,
        monthTime: json['monthTime'] as int,
        hourTime: json['hourTime'] as int,
        minuteTime: json['minuteTime'] as int,
        price: json['price'] as int,
        totalPrice: json['totalPrice'] as int
    );
  }

  Map<String, Object?> toJson() => {
    'number': userPhoneNumber,
    'title': title,
    'id': id,
    'amountItems': amountItems,
    'dayTime': dayTime,
    'monthTime': monthTime,
    'hourTime': hourTime,
    'minuteTime': minuteTime,
    'price': price,
    'totalPrice': totalPrice,
  };
}
```

Рисунок 3.35 – JSON-конвертор класу ProductSpeedOrder

```dart
class ProductCart extends Product{
  int amountItems;
  int totalPrice;
  ProductCart({
    required id,
    required price,
    required title,
    required image,
    required description,
    required color,
    required List<String>? images,
    required size,
    required this.amountItems,
    required this.totalPrice
  }) : super(id, price, title, image, description, color, size, images);

  factory ProductCart.fromJson(Map<String, dynamic> json) {
    return ProductCart(
      id: json['id'] as int,
      price: json['price'] as int,
      title: json['title'] as String,
      image: json['image'] as String,
      description: json['description'] as String,
      color: json['color'] as String,
      size: json['size'] as String,
      images: (json['images'] as List).map((e) => e as String).toList(),
      amountItems: json['amountItems'] as int,
      totalPrice: json['totalPrice'] as int,
    );
  }

  Map<String, Object?> toJson() => {
    'id': id,
    'price': price,
    'title': title,
    'image': image,
    'description': description,
    'color': color,
    'images': images,
    'size': size,
    'amountItems': amountItems,
    'totalPrice': totalPrice,
  };
}

class ProductOrder{
  final List<ProductCart> order;
  final int orderNumber;
  final int dayTime;
  final int monthTime;
  final int hourTime;
  final int minuteTime;
  final String name;
  final String post;
  final String city;
  final String phoneNumber;

  ProductOrder({
    required this.orderNumber,
    required this.order,
    required this.dayTime,
    required this.monthTime,
    required this.hourTime,
    required this.minuteTime,
    required this.name,
    required this.post,
    required this.city,
    required this.phoneNumber,
  });

  factory ProductOrder.fromJson(Map<String, dynamic> json) {
    return ProductOrder(
        order: (json['order'] as List<dynamic>).map((dynamic e) => ProductCart.fromJson(e as Map<String,
dynamic>)).toList(),
        orderNumber: json['orderNumber'] as int,
        dayTime: json['dayTime'] as int,
        monthTime: json['monthTime'] as int,
        hourTime: json['hourTime'] as int,
        minuteTime: json['minuteTime'] as int,
        name: json['name'] as String,
        post: json['post'] as String,
        city: json['city'] as String,
        phoneNumber: json['phoneNumber'] as String,
    );
  }

  Map<String, Object?> toJson() => {
    'order': order.map((e) => e.toJson()),
    'orderNumber': orderNumber,
    'dayTime': dayTime,
    'monthTime': monthTime,
    'hourTime': hourTime,
    'minuteTime': minuteTime,
    'name': name,
    'post': post,
    'city': city,
    'phoneNumber': phoneNumber,
  };
}
```

Рисунок 3.36 – JSON-конвертори класів ProductCart, ProductOrder

3.7.2 Створення інстанса в файлі

Інстанс – це оголошення створення нової колекції даних з мережі.

Для використовування баз даних Firebase в Flutter-проекті необхідно зробити декілька кроків:

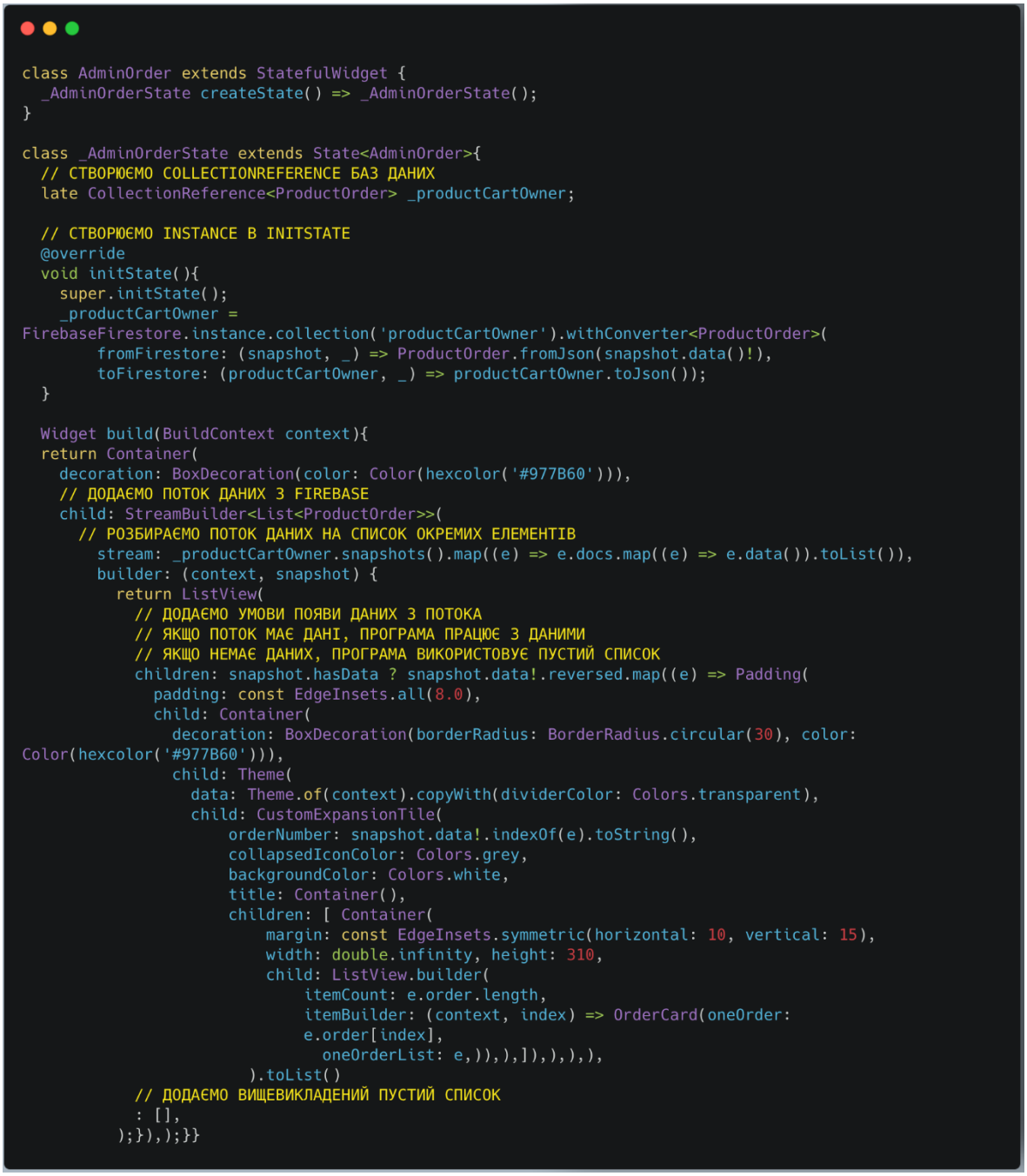
1. Створити CollectionReference – цей об’єкт являє собою назву списку даних Firebase. Використовуючи його, програміст працює з ним такими ж методами, що із локальним списком.

2. Додати Instance в initState() – оголосити програмі, що необхідно створити потік даних і конвертувати його створеними JSON-конверторами.

3. Додати StreamBuilder – додати потік даних з Firebase до дерева віджетів.

4. Розібрати потік даних на окремі елементи та зробити з них список.

За зразок з програми візьмем сторінку адміністратора Admin Page, можна побачити на рисунку 3.37.



```
class AdminOrder extends StatefulWidget {
  _AdminOrderState createState() => _AdminOrderState();
}

class _AdminOrderState extends State<AdminOrder>{
  // СТВОРЮЄМО COLLECTIONREFERENCE БАЗ ДАНИХ
  late CollectionReference<ProductOrder> _productCartOwner;

  // СТВОРЮЄМО INSTANCE В INITSTATE
  @override
  void initState(){
    super.initState();
    _productCartOwner =
FirebaseFirestore.instance.collection('productCartOwner').withConverter<ProductOrder>(
        fromFirestore: (snapshot, _) => ProductOrder.fromJson(snapshot.data()!),
        toFirestore: (productCartOwner, _) => productCartOwner.toJson());
  }

  Widget build(BuildContext context){
  return Container(
    decoration: BoxDecoration(color: Color(hexcolor('#977B60'))),
    // ДОДАЄМО ПОТОК ДАНИХ З FIREBASE
    child: StreamBuilder<List<ProductOrder>>(
      // РОЗБИРАЄМО ПОТОК ДАНИХ НА СПИСОК ОКРЕМИХ ЕЛЕМЕНТІВ
        stream: _productCartOwner.snapshots().map((e) => e.docs.map((e) => e.data()).toList()),
        builder: (context, snapshot) {
          return ListView(
            // ДОДАЄМО УМОВИ ПОЯВИ ДАНИХ З ПОТОКА
            // ЯКЩО ПОТОК МАЄ ДАНІ, ПРОГРАМА ПРАЦЮЄ З ДАНИМИ
            // ЯКЩО НЕМАЄ ДАНИХ, ПРОГРАМА ВИКОРИСТОВУЄ ПУСТИЙ СПИСОК
            children: snapshot.hasData ? snapshot.data!.reversed.map((e) => Padding(
              padding: const EdgeInsets.all(8.0),
              child: Container(
                decoration: BoxDecoration(borderRadius: BorderRadius.circular(30), color:
Color(hexcolor('#977B60'))),
                child: Theme(
                  data: Theme.of(context).copyWith(dividerColor: Colors.transparent),
                  child: CustomExpansionTile(
                      orderNumber: snapshot.data!.indexOf(e).toString(),
                      collapsedIconColor: Colors.grey,
                      backgroundColor: Colors.white,
                      title: Container(),
                      children: [ Container(
                          margin: const EdgeInsets.symmetric(horizontal: 10, vertical: 15),
                          width: double.infinity, height: 310,
                          child: ListView.builder(
                              itemCount: e.order.length,
                              itemBuilder: (context, index) => OrderCard(oneOrder:
                              e.order[index],
                                oneOrderList: e,)),),]),),),),
                        ).toList()
            // ДОДАЄМО ВИЩЕВИКЛАДЕНИЙ ПУСТИЙ СПИСОК
            : [],
          );}),);}}
```

Рисунок 3.37. – Сторінка адміністратора з приєднанням instance з Firebase

У випадку функції додавання/видалення даних з бази даних Firebase. Використовується створений на початку список CollectionReference.

3.7.3 Аутентифікація Firebase

Для приєднання сервісу аутентифікації до програми. Слід у файлі аутентифікації використати наступні кроки:

Додати ініціалізатор аутентифікації в Firebase, можна побачити на рисунку 3.38

```
Future<FirebaseApp> _initializeFirebase() async {
    FirebaseApp firebaseApp = await Firebase.initializeApp();
    return firebaseApp;
  }
```

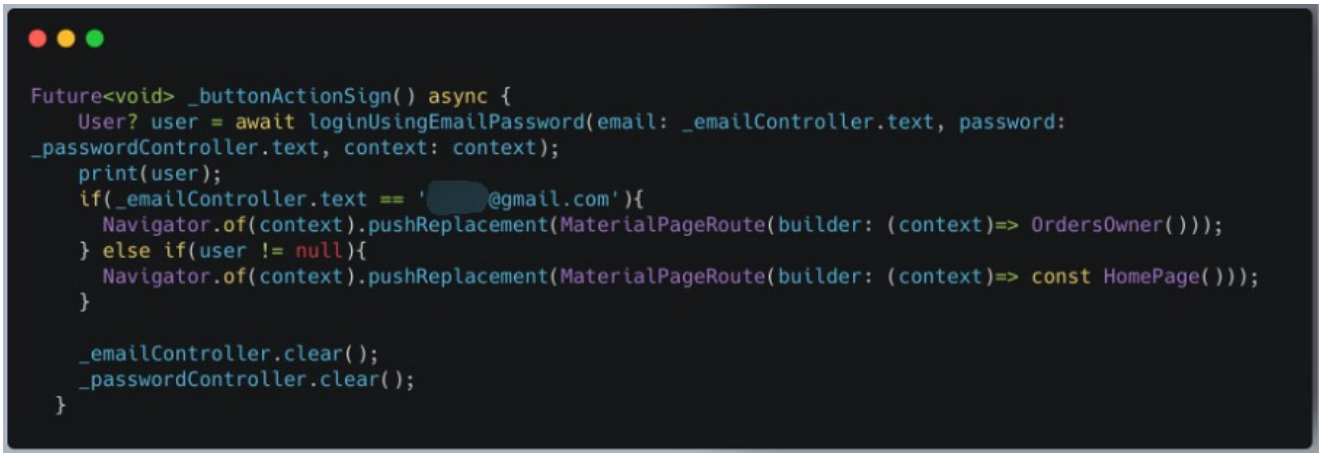
Рисунок 3.38 – Ініціалізатор аутентифікації користувача

Додати функцію входу і реєстрації користувача, можна побачити на рисунку 3.39

```
static Future<User?> loginUsingEmailPassword({required String email, required String password, required
BuildContext context})async{
    FirebaseAuth _auth = FirebaseAuth.instance;
    User? user;
    try{
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(email: email, password:
password);
      user = userCredential.user;
    } on FirebaseAuthException catch (e){
      if(e.code == "user-not-found"){
        print("Користувач з таким email не знайден");
      }
    }
    return user;
  }

  static Future<User?> registerInWithEmailAndPassword({required String email, required String password,
required BuildContext context})async {
    User? user;
    try {UserCredential userCredential = await
FirebaseAuth.instance.createUserWithEmailAndPassword(email: email, password: password);
    } on FirebaseAuthException catch (e) {
      if (e.code == 'weak-password') {
        print('The password provided is too weak.');
      } else if (e.code == 'email-already-in-use') {
        print('The account already exists for that email.');
      }
    } return user;
  }
```

Рисунок 3.39 – Функція входу та реєстрації користувача

У випадку входу адміністратора додати умову переходу до сторінки керування замовленнями, можна побачити на рисунку 3.40.



```
Future<void> _buttonActionSign() async {
    User? user = await loginUsingEmailPassword(email: _emailController.text, password:
_passwordController.text, context: context);
    print(user);
    if(_emailController.text == '      @gmail.com'){
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=> OrdersOwner()));
    } else if(user != null){
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=> const HomePage()));
    }

    _emailController.clear();
    _passwordController.clear();
  }
```

Рисунок 3.41 – Умови входу адміністратора додатка

Файл проекту з аутентифікацією Authentication Page у повному обсягу, можна побачити на рисунках 3.42, 3.43.

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:ly_flutter/ui/admin_page/admin_page.dart';
import 'package:ly_flutter/ui/home_page/home_page.dart';


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


  @override
  _AuthenticationPageState createState() => _AuthenticationPageState();
}

class _AuthenticationPageState extends State<AuthenticationPage> {

  Future<FirebaseApp> _initializeFirebase() async {
    FirebaseApp firebaseApp = await Firebase.initializeApp();
    return firebaseApp;
  }


  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();

  static Future<User?> loginUsingEmailPassword({required String email, required String password, required
BuildContext context})async{
    FirebaseAuth _auth = FirebaseAuth.instance;
    User? user;
    try{
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(email: email, password: password);
      user = userCredential.user;
    } on FirebaseAuthException catch (e){
      if(e.code == "user-not-found"){
        print("Користувач з таким email не знайден");
      }
    }
    return user;
  }

  static Future<User?> registerInWithEmailAndPassword({required String email, required String password, required
BuildContext context})async {
    User? user;
    try {
      UserCredential userCredential = await FirebaseAuth.instance
          .createUserWithEmailAndPassword(email: email, password: password);
    } on FirebaseAuthException catch (e) {
      if (e.code == 'weak-password') {
        print('The password provided is too weak.');
      } else if (e.code == 'email-already-in-use') {
        print('The account already exists for that email.');
      }
    } return user;
  }

  bool showLogin = true;

  Widget _button(String text, void func()){
    return RaisedButton(
      splashColor: Theme.of(context).primaryColor,
      highlightColor: Theme.of(context).primaryColor,
      color: Colors.white,
      child: Text(text, style: TextStyle(fontWeight: FontWeight.bold, color: Theme.of(context).primaryColor,
fontSize: 20)),
      onPressed: () {
        func();},);}

  Widget _logo(){
    return const Padding(
        padding: EdgeInsets.only(top: 100),
        child: Align(
            child: Text('LY ODESA', style: TextStyle(fontSize: 45, fontWeight: FontWeight.bold, color:
Colors.white))
        )
    );
  }

  Widget _form(String label, void func()){
    return Column(children: <Widget>[
            Container(
              margin: EdgeInsets.symmetric(horizontal: 15, vertical: 10),
              child: TextField(
                 controller: _emailController,
                 style: TextStyle(fontSize: 20, color: Colors.white),
                 decoration: InputDecoration(
                   hintText: 'Логін',
                   hintStyle: TextStyle(fontWeight: FontWeight.bold, fontSize: 20, color:
Colors.white.withOpacity(0.5)),
                  focusedBorder: OutlineInputBorder(
                  borderSide: BorderSide(color: Colors.white, width: 2),),
                  enabledBorder: OutlineInputBorder(
                     borderSide: BorderSide(color: Colors.white.withOpacity(0.8), width: 1)),
                 prefixIcon: Padding(
                   padding:  EdgeInsets.only(left: 10, right: 10),
                   child: IconTheme(
                       data: IconThemeData(color: Colors.white),
                       child: Icon(Icons.mail)),),)),),
```

Рисунок 3.42 – Файл authentication.dart

```
          Container(
            margin: EdgeInsets.symmetric(horizontal: 15, vertical: 10),
            child: TextField(
              obscureText: false,
              controller: _passwordController,
                style: TextStyle(fontSize: 20, color: Colors.white),
                decoration: InputDecoration(
                  hintText: 'Пароль',
                  hintStyle: TextStyle(fontWeight: FontWeight.bold, fontSize: 20, color:
Colors.white.withOpacity(0.5)),
                  focusedBorder: OutlineInputBorder(
                    borderSide: BorderSide(color: Colors.white, width: 2),
                  ),
                  enabledBorder: OutlineInputBorder(
                      borderSide: BorderSide(color: Colors.white.withOpacity(0.8), width: 1)
                  ),
                  prefixIcon: Padding(
                    padding:  EdgeInsets.only(left: 10, right: 10),
                    child: IconTheme(
                        data: IconThemeData(color: Colors.white),
                        child: Icon(Icons.mail)),),)),),
          _button(label, func)]);}

  Future<void> _buttonActionSign() async {
    User? user = await loginUsingEmailPassword(email: _emailController.text, password: _passwordController.text,
context: context);
    print(user);
    if(_emailController.text == 'ceoly@gmail.com'){
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=> OrdersOwner()));
    } else if(user != null){
      Navigator.of(context).pushReplacement(MaterialPageRoute(builder: (context)=> const HomePage()));
    }
    _emailController.clear();
    _passwordController.clear();
  }

  void backFunc(){
    Navigator.push(context, MaterialPageRoute(builder: (context) => AuthenticationPage()));
  }

  Future<void> _buttonActionRegister() async {
    User? user = await registerInWithEmailAndPassword(email: _emailController.text, password:
_passwordController.text, context: context);
    if (user == null){
      backFunc();
    }
    _emailController.clear();
    _passwordController.clear();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(backgroundColor: Theme.of(context).primaryColor,
        appBar: AppBar(
          backgroundColor: Colors.transparent, elevation: 0,
          leading: IconButton(icon: const Icon(Icons.keyboard_arrow_left, size: 30,), onPressed: ()
{Navigator.pop(context);},)
          ),
        body: FutureBuilder(
            future: _initializeFirebase(), builder: (context, snapshot){
              if (snapshot.connectionState == ConnectionState.done){
                return SingleChildScrollView(
                  child: Column(children: <Widget>[
                      _logo(),
                      const SizedBox(height: 20),
                      (showLogin
                          ? Column( children: <Widget>[
                          _form('УВІЙТИ', _buttonActionSign),
                          Padding( padding: const EdgeInsets.all(10),
                              child: GestureDetector(
                                child: const Text('Ще не зареєстровані? Реєстрація!', style: TextStyle(fontSize: 15,
color: Colors.white)),
                                onTap:() {setState((){
                                    showLogin = false;
                                  });},)],)
                          :
                      Column(children: <Widget>[
                          _form('РЕЄСТРАЦІЯ', _buttonActionRegister),
                          Padding(
                              padding: const EdgeInsets.all(10),
                              child: GestureDetector(
                                child: const Text('Готові реєструватись? Уперед!', style: TextStyle(fontSize: 15,
color: Colors.white)),
                                onTap:() {
                                  setState((){ showLogin = true;
                                  });},)],),),);}
              return const Center(child: CircularProgressIndicator(),);}));}}
```

Рисунок 3.43 – Файл authentication.dart

3.8 Деплой робочого додатку у вебвигляді.

Зазначимо, що для деплоя сайту в Firebase і мережу інтернет, необхідно встановити Firebase до Terminal наступною командою, можна побачити на рисунку 3.44.

Рисунок 3.44 – Команда встановлення Firebase до Terminal

Слід за цим, ініціалізуємо створений проект для Firebase Terminal, можна побачити на рисунку 3.45.

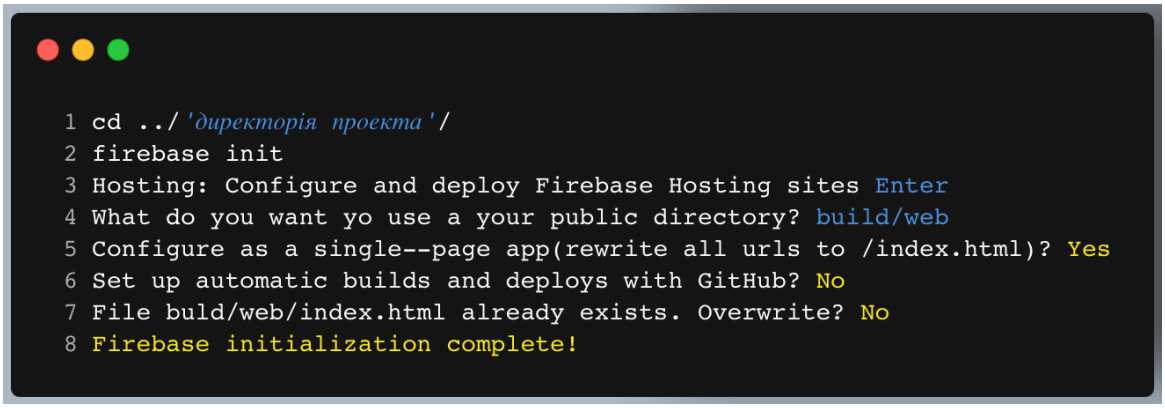


Рисунок 3.45 – Команди ініціалізації проекту Firebase

Наостанок, для деплоя додатка у вебвигляді, необхідно використати наступні команди, можна побачити на рисунку 3.46

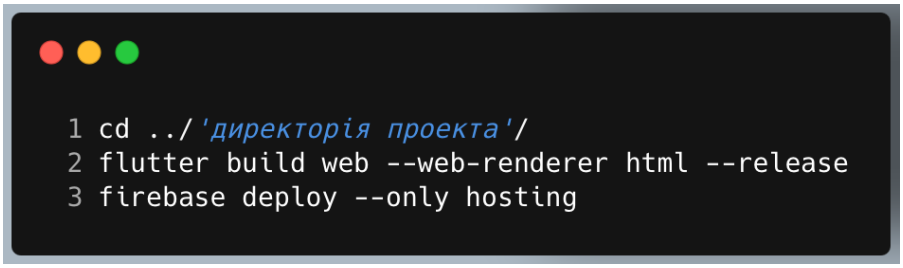


Рисунок 3.46 – Команда деплоя проекта до мережі в Firebase

4. ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

4.1 Інтерфейс готового мобільного додатку

Загальна навігація та інтерфейс готового мобільного додатку, які були зазначені на етапі проектування програми, можна побачити на рисунку 4.1.

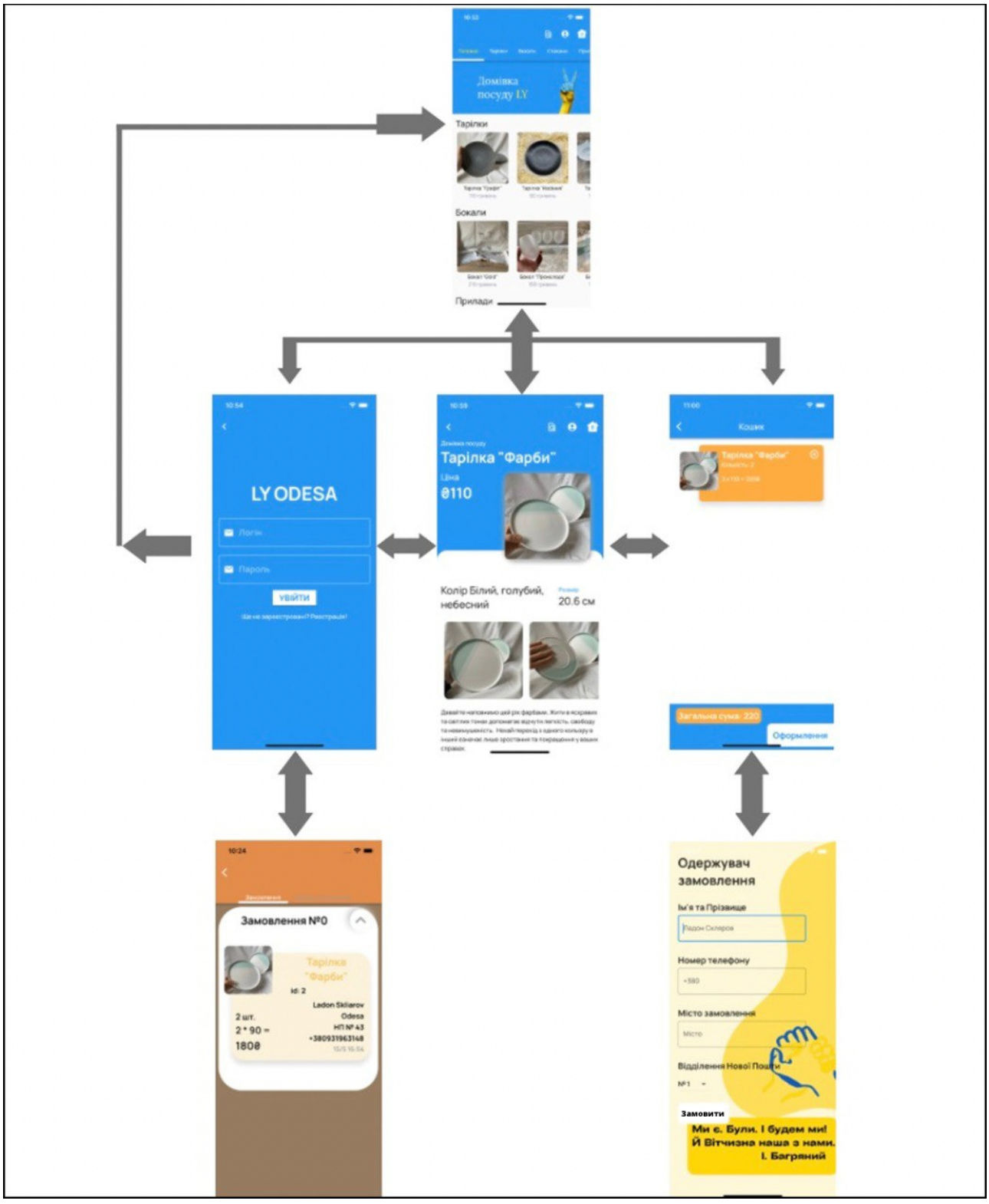


Рисунок 4.1 – Загальна навігація робочого додатку

4.2 Взаємодія зі сторони клієнта та адміністратора.

Необхідно побачити:

Готову навігацію взаємодії потенційного клієнта, який замовляє товари, можна побачити на рисунку 4.2.

Готову навігацію адміністратора магазинам, який переглядає замовлення, що надіслав цей клієнт, можна побачити на рисунку 4.3.

Рисунок 4.2 – Навігація робочого додатку зі сторони клієнта

Рисунок 4.3 – Навігація робочого додатку зі сторони

Необхідно побачити здатність користувача справно виконати замовлення. Взаємодію екранів звичайного замовлення можна побачити на рисунку 4.4.

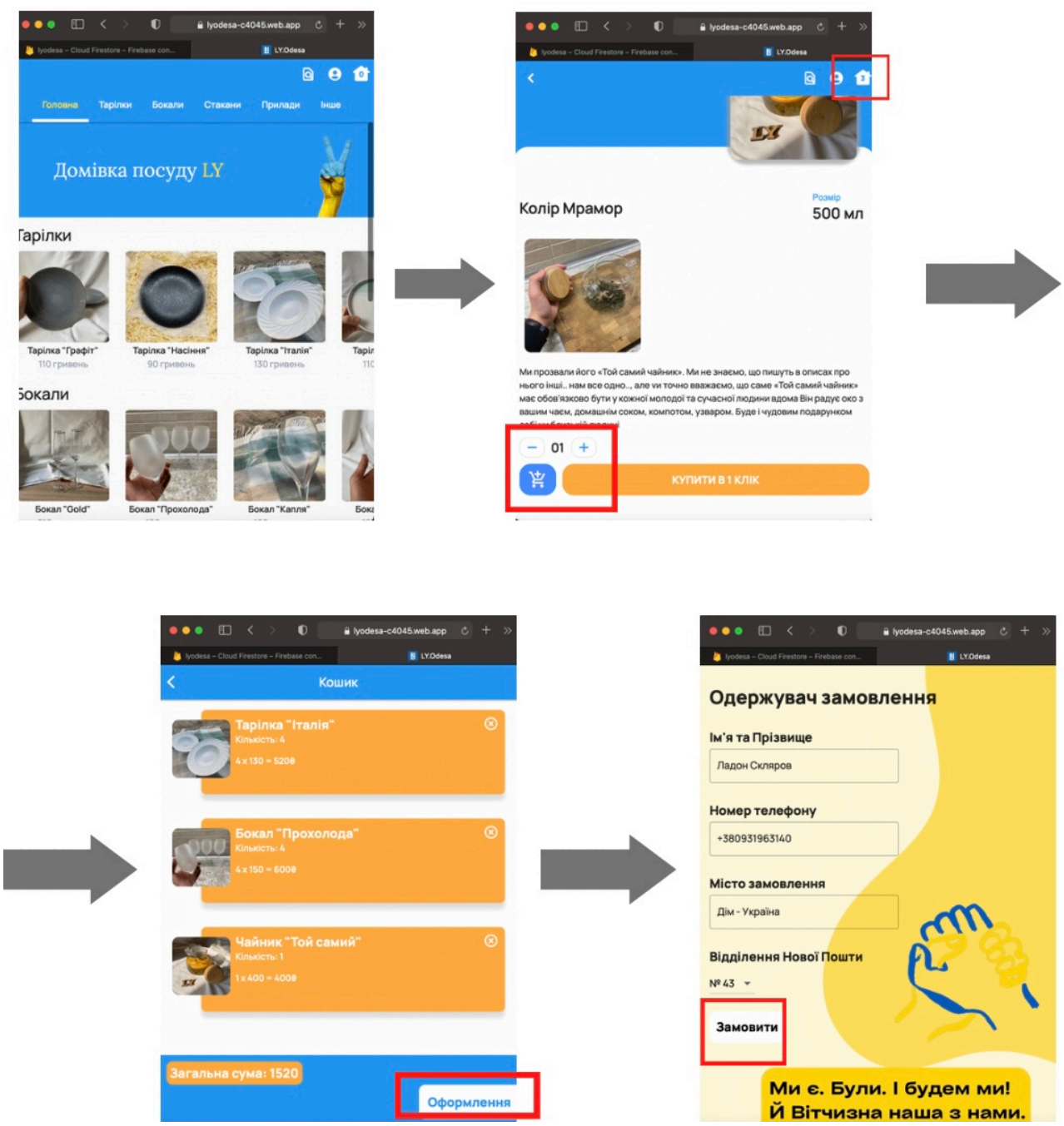


Рисунок 4.4 – Справна взаємодія замовлення користувача

Необхідно побачити здатність користувача справно виконати замовлення. Взаємодію екранів швидкого замовлення можна побачити на рисунку 4.5.

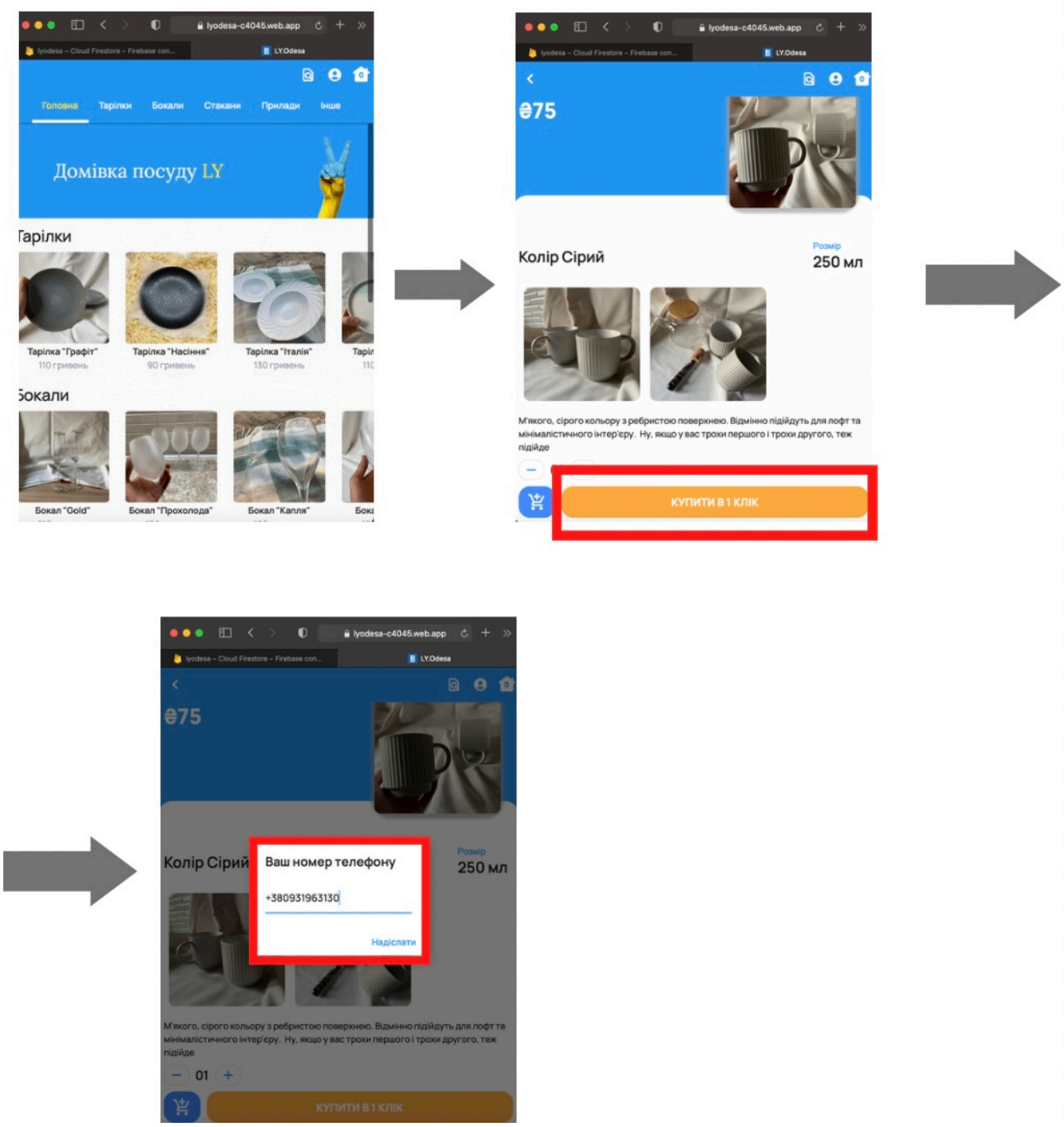


Рисунок 4.5 – Справна взаємодія швидкого замовлення користувача

Необхідно побачити здатність адміністратора справну роботу екранів заказу. Взаємодію екранів можна побачити на рисунку 4.6.

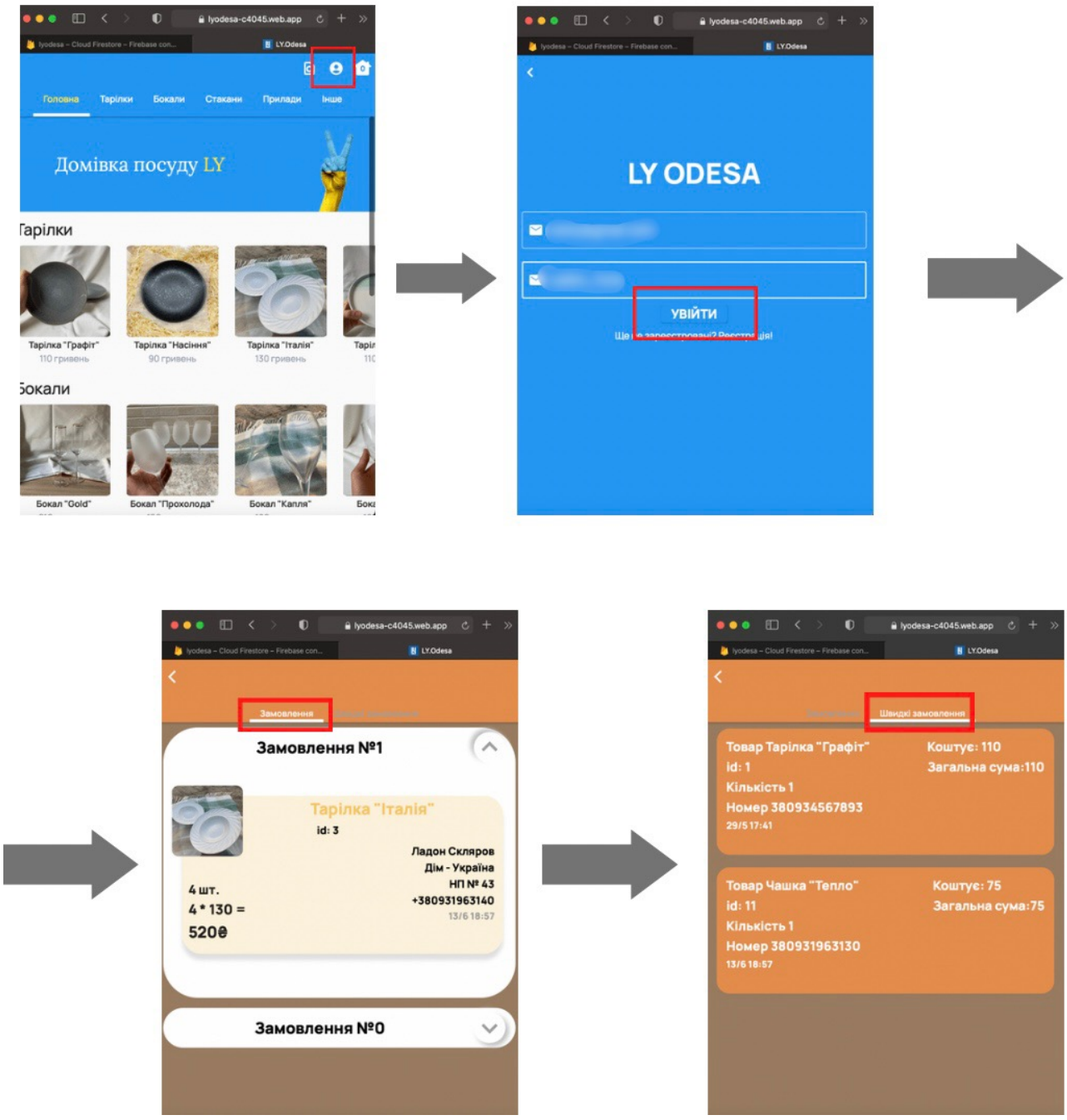


Рисунок 4.6 – Справний перегляд звичайних та швидких замовлень користувача

4.ОХОРОНА ПРАЦІ

ВСТУП

З розвитком комп‘терних технологій, які є великою частиною сучасного світу людини, поширилися і професії пов‘язані з працею за комп‘тером. Згідно з законодавствами України про охорону праці, необхідно ретельно поставитись до безпеки працівника електронно-обчислювальної машини, удосконалити існуючи та розробити нові підходи до організації робочих місць, проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу ПК на здоров'я користувачів

Найбільш повним нормативним документом щодо забезпечення охорони праці користувачів ПК є "Державні санітарні правила і норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин" ДСанПіН 3.3.2.007-98.

У дипломній роботі розглянемо робоче місце програміста/ який зазвичай працює в офісі.

1. Аналіз та безпека умов праці при роботі з ПК

В процесі роботи на користувачів ПК можуть мати вплив наступні небезпечні та шкідливі фактори:

На операторів ПК і програмістів можуть мати вплив такі фізичні небезпечні і шкідливі виробничі фактори, як підвищений рівень шуму, підвищена температура зовнішнього середовища, недостатня освітленість робочої зони, електричний струм та інші. Тому на робочому місці програміста повинні бути створені умови для високопродуктивної праці.

Також у працівника, може бути психологічне перенавантаження, за рахунок довгої праці з комп’ютером, тому необхідно врахувати відпочинок при плануванні часу роботи працівника. Н а робочому місці програміста повинні бути створені умови для високопродуктивної праці.

1.1 Організація робочого місця

Робочі місця програмістів за дисплеями слід розміщувати в спеціально відведеному приміщенні, яке відповідає гігієнічним вимогам щодо площі, умов природного освітлення та вентиляції.

Робочі місця з персональними комп'ютерами в приміщеннях з джерелами шкідливих виробничих факторів розміщуються в ізольованих кабінах з організованим повітрообміном. При виконанні творчої роботи, що вимагає значного розумового напруження або високої концентрації уваги, робочі місця з персональними комп'ютерами рекомендується ізолювати один від одного перегородками висотою 1,5-2,0 м.

Екран ПК має розташовуватися на оптимальній відстані від очей користувача, яка становить 600-700мм, але не ближче між за 700мм з урахуванням розміру літерно-цифрових знаків і символів. Розташування екрана має забезпечувати зручність зорового споглядання у вертикальній площині під кутом 30° до нормалі.

Висота поверхні робочого столу з комп’ютером має регулюватися в межах 680-800мм, а ширині і глибина - забезпечувати можливість виконання операцій у зоні досяжності моторного поля (рекомендовані розміри 600-1400мм, глибина – 800-1000мм).

Робочий стіл повинен мати простір для ніг заввишки не менше ніж 600мм, завширшки не менше ніж 500мм, завглибшки (на рівні колін) не менше ніж 450мм, на рівні простягнутої ноги – не менше ніж 650мм.

Робочий стілець має бути підйомно-поворотним, регульованим за висотою, за кутом нахилу сидіння та спинки і за відстанню від спинки до переднього краю сидіння, поверхня сидіння має бути плоскою, передній край - заокругленим.– 400мм.

Для зниження статичного напруження м'язів верхніх кінцівок слід використовувати стаціонарні або змінні підлокітники завдовжки не менше ніж 250мм, завширшки 50-70мм, що регулюються за висотою над сидінням у межах 230-260мм і відстанню між підлокітниками в межах 350-500мм. Поверхня

сидіння і спинки стільця має бути напівм'якою з нековзним, повітронепроникним покриттям, що легко чиститься і не електризується.

На одне робоче місце відводиться 6 квадратних метрів та 20 кубічних обсягів – тобто санітарні норми виконуються.

1.2 Вимоги безпеки до мікроклімату виробничих приміщень

Мікроклімат

При виконанні робіт операторського типу, пов'язаних з нервово-емоційним напруженням в кабінетах, пультах і постах керування технологічними процесами, в залах обчислювальної техніки та інших приміщеннях повинні дотримуватися оптимальні умови мікроклімату (температура повітря 22 - 24° С, відносна вологість 60 - 40 %, швидкість руху повітря не більш 0,1 м/сек.).

Висока чи низька температура повітря в приміщенні з ПК негативно впливає на функціональний стан користувача. Недостатня вологість в приміщенні призводить до надмірного висихання слизових оболонок очей, носа, горла та до нагромадження зарядів статичної електрики, що утворюються в процесі роботи ПК. Разом з тим недопустима вологість повітря більше 75%. На робочих місцях користувачів ПК параметри мікроклімату мають відповідати вимогам ДСН 3.3.6.042-99 “Санітарні норми мікроклімату виробничих приміщень”, ДСан ПіН 3.3.2-007-98, ДНАОП 0.00-1.31-99.

В наступній таблиці приведені норми мікроклімату для приміщень з ВДТ ЕОМ та ПЕОМ (ПК)

Пора року	Категорія робіт	Температура повітря, °С, не більше	Відносна вологість повітря, %	Швидкість руху повітря, м/с
Холодна	легка – 1а	22-24	40-60	0,1
	легка – 1б	21-23	40-60	0,1
Тепла	легка – 1а	23-25	40-60	0,1
	легка – 16	22-24	40-60	0,2

Рівень шуму

Нормативним значенням еквівалентного рівня звуку для програмістів є 50 дБА. За результатами проведених гігієнічних досліджень, еквівалентні рівні

шуму на робочих місцях працівників офісних приміщень знаходяться у межах 48-59 дБАекв.

Вид трудової діяльності	Рівні звукового тиску в дБ в октавних смугах із середньогеометричними частотами, Гц									
	31,5	63	125	250	500	1000	2000	4000	8000	Рівні звуку, еквівалентні рівні звуку, дБА/дБАекв.
Програмісти ЕОМ	86	71	61	54	49	45	42	40	38	50
Оператори в залах обробки інформації на ЕОМ та оператори комп'ютерного набору	96	83	74	68	63	60	57	55	54	65
В приміщеннях для розташування шумних агрегатів ЕОМ	103	91	83	77	73	70	68	66	64	75

Освітлення

Приміщення, в яких встановлені персональні комп'ютери, повинні мати природне та штучне освітлення відповідно до СНиП II-4-79. Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%.

Штучне освітлення в приміщеннях з робочими місцями має здійснюватись системою загального рівномірного освітлення. У разі переважної роботи з документами, допускається застосування системи комбінованого освітлення (крім системи загального освітлення додатково встановлюються світильники місцевого освітлення).

Зазначення освітленості на поверхні робочого столу в зоні розміщення документів має становити 300-500лк. Якщо ці значення освітленості неможливо забезпечити системою загального освітлення, допускається використовувати місцеве освітлення. При цьому світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300лк.

Як джерела світла в разі штучного освітлення мають застосовуватись переважно люмінесцентні лампи типу ЛБ. У разі влаштування відбитого освітлення у приміщеннях, де переважним чином працюють з документами,

допускається застосування металогалогенних ламп потужністю 250Вт. Допускається застосування ламп розжарювання у світильниках місцевого освітлення.

Електробезпека

Персональні комп'ютери, периферійні пристрої повинні бути підключені тільки до заводських та штепсельних розеток. Не підключати пристрої у звичайну двопровідну електромережу.

Не залишати включеними електропристрої покидаючи робоче місце.
Після закінчення робочого процесу з персональним комп'ютером, усі пристрої повинні бути відключені від електричної мережі.

2. **Пожежна безпека**

Основними напрямками забезпечення пожежної безпеки є усунення умов виникнення пожежі та мінімізація її наслідків. Об'єкти повинні мати системи пожежної безпеки, спрямовані на запобігання пожежі, дії на людей та матеріальні цінності небезпечних факторів пожежі, в тому числі їх вторинних проявів..

До основних причин пожеж на виробництві слід віднести:

• порушення правил монтажу та експлуатації електроустановок;

• необережне поводження з вогнем ;

• порушення технологічного процесу виробництва;

• порушення правил пожежної безпеки при електрогазозварюванні та різанні металів, паяльних роботах, розігріванні бітуму та проведенні інших видів вогневих робіт;

• порушення правил монтажу та експлуатації приладів опалення;

• іскри теплового та механічного походження;

• підпали;

• інші причини .

Всі приміщення повинні бути забезпечені первинними засобами пожежогасіння: пожежним водопостачанням (пожежні крани ПК), пожежні щити з набором пожежного інструменту тощо.

Пожежна безпека приміщень, що мають електричні мережі, регламентується ГОСТ 12.1.033-81, ГОСТ 12.1.004-85. Робота оператора ЕОМ повинна вестися в приміщенні, що відповідає категорії Д пожежної безпеки (негорючі речовини й матеріали в холодному стані).

Приміщення оснащені вуглекислотними або порошковими вогнегасниками. У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефону 101 пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації і приступити до ліквідації пожежі.

3. Висновки

За дотриманням вимог охорони праці, роботодавець забезпечить людині безпечні умови роботи згідно з законодавствами України. Людина в свою чергу повинна прийти на роботу здоровою та піти з роботи живою та здоровою.

ВИСНОВОК

Розвиток інтернет технологій із застосуванням смартфону, впроваджує їх у життя більшості людей. Робота із сайтами, мобільними додатками, уникає контакту с незнайомою людиною, надає весь обсяг необхідної інформації для користувача, сприяє швидкому виконанню власних справ.

Сформульована мета і виконана постановка завдань до дипломного проекту бакалавра. Були затверджені вимоги користувача до ПЗ.

Виконано розробку функціонального, мобільного додатку для продажу товарів посуду у мережі інтернет, а також проведено архітектурне і детальне проектування класів, дерева віджетів, розроблено Firebase бази даних.

Для тестування було зроблено ручне тестування функцій, які забезпечують працездатність роботи додатка. Тестування пройдено успішно, тому можна зробити висновок про повну відповідність програмного забезпечення з поставленим перед ним завданням.

На закінчення можна зробити висновок: застосування мобільних додатків для автоматизації бізнес-процесів сприяє пошидвиденню процесу замовлення, наближення до контакту бізнеса із життям людини. В результаті застосування мобільних додатків на Flutter бізнес покращується.

Постійне використовування додатку дозволить прискорити бізнес-процеси, зменшити людське втручання у контакт із клієнтами. Вважається, що використання мобільних додатків, дозволяє застосувати цей метод в різних сферах бізнесу, це буде зручно як власникам, так і працівникам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Роберт Мартін Чистий Код 2019 464 [Електронний ресурс]. –Режим доступу: https://ru.pdfdrive.com/Чистый-код-создание-анализ-и-рефакторинг-e188599881.html. – Назва з екрану.

2. Tutorials Point Dart Programming 2017 201 [Електронний ресурс]. –Режим доступу: https://www.tutorialspoint.com/dart_programming/dart_programming_tutorial.pdf. – Назва з екрану.

3. Google Офіційна документація з SDK Flutter 2015 [Електронний ресурс]. – Режим доступу: https://flutter.dev. – Назва з екрану.

4. Google Офіційна документація з мови Dart 2011[Електронний ресурс]. – Режим доступу: https://dart.dev. – Назва з екрану.

5. Команда Flutter Офіційний, англомовний Youtube-канал з головними темами для Flutter розробки [Електронний ресурс]. –Режим доступу: https://www.youtube.com/c/flutterdev. – Назва з екрану.

6. Йоханнес Майк англомовний Youtube-канал з головними темами для Flutter розробки [Електронний ресурс]. –Режим доступу: https://www.youtube.com/c/JohannesMilke. – Назва з екрану.

7. Metanit Flutter Посібник з мови Dart 2012 [Електронний ресурс]. –Режим доступу: https://metanit.com/dart/tutorial/. – Назва з екрану.