МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ЗАМОВЛЕННЯ ТА ДОСТАВКИ ТОВАРІВ ЗАСОБАМИ FLUTTER ТА LARAVEL»**

Виконав: студент 2 курсу, групи 8.1211-іпз
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
Д.О. Дребезов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.ф.-м.н., Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, доцент, к.ф.-м.н., Клименко М.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.

(підпис)

« 04 » 05 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Дребезову Денису Олеговичу

(прізвище, ім'я та по– батькові)

1. Тема роботи (проекту) Розробка інформаційної системи замовлення та доставки товарів засобами Flutter та Laravel

керівник роботи (проекту) Кудін Олексій Володимирович, к.ф.-м.н., доцент

(прізвище, ім'я та по– батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 04 » 05 2022 року № 500- с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково– пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Моделювання та проектування програмного доповнення.
3. Реалізація системи.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Кудін О. В, доцент		
2	Кудін О. В, доцент		
3	Кудін О. В, доцент		

7.Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану виконання кваліфікаційної роботи магістра.	15.05.2022	
2.	Збір вихідних даних та аналіз предметної області.	20.06.2022	
3.	Обробка методичних та теоретичних джерел.	15.07.2022	
4.	Специфікація вимог до системи. Робота над першим розділом.	29.07.2022	
5.	Моделювання та проектування системи. Робота над другим розділом.	19.08.2022	
6.	Реалізація та тестування системи. Робота над третім розділом.	27.09.2022	
7.	Оформлення та нормоконтроль кваліфікаційної роботи бакалавра.	12.12.22	
8.	Захист кваліфікаційної роботи бакалавра.	15.12.22	

Студент _____ Д.О. Дребезов
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.В. Кудін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка інформаційної системи замовлення та доставки товарів засобами Flutter та Laravel»: 67с., 24 рис., 14 табл., 19 джерел.

API, FLUTTER, BACK-END, DART, LARAVEL, LARAVEL SANCTUM, DOCKER, MYSQL, ANDROID, GOOGLE MAPS.

Об’єкт дослідження – фреймворк Flutter, бібліотеки для роботи з ним, засоби обробки даних на стороні клієнта та на сервері.

Мета роботи: розробка застосунку для замовлення та доставки товарів засобами Flutter та Laravel.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проектування, конструювання та тестування програмного забезпечення.

Предмет дослідження – створення платформи, що дозволяє користувачам виконувати замовлення та доставки товарів у приватному сегменті.

При розробці додатку був проведений аналіз предметної області, обрано та спроєктовано архітектуру системи за допомогою UML у вигляді наборів діаграм варіантів використання, послідовностей, компонентів, розгортання та діяльності. Реалізовано серверну частину з використанням Laravel, клієнтська частина була розроблена за допомогою фреймворку Flutter та бібліотеки Google Maps.

В результаті роботи отримано програмне забезпечення для замовлення та доставки товарів.

SUMMARY

Master's Qualifying Paper “ Development of the Products Ordering and Delivering Informational System using Flutter and Laravel”: 67 pages, 24 figures, 14 tables, 19 references.

API, FLUTTER, BACK-END, DART, LARAVEL, LARAVEL SANCTUM, DOCKER, MYSQL, ANDROID, GOOGLE MAPS.

The object of the study is – Flutter framework, libraries for working with it, data processing tools on the client side and on the server.

The aim of the study is – development of an application for ordering and delivery of goods using Flutter and Laravel.

The method of research is – methods of collecting and analyzing software requirements, methods of modeling, designing, constructing and testing software.

The subject of research is the creation of a platform that allows users to order and deliver goods in the private segment.

During the development of the application, the subject area was analyzed, the system architecture was selected and designed using UML in the form of sets of diagrams of use cases, sequences, components, deployment and activities. The server part was implemented using Laravel, the client part was developed using the Flutter framework and Google Maps library.

The result of the work is the software for ordering and delivery of goods.

ЗМІСТ

ВСТУП

Протягом останніх років спостерігається тенденція до збільшення використання різних сервісів доставок, та застосунків для цього. Часто системи для доставки містять в собі декілька застосунків для різних платформ. Це дає змогу створити унікальний досвід користувача, але накладає велику складність підтримки системи, та й сама розробка виростає в часі, та фінансах. Полегшити підтримку та зменшити час розробки може створення одного застосунку для різних платформ.

Задача створення кросплатформових застосунків, які виглядають та працюють як розроблені під специфічну платформу – складна задача. Звісно застосунок можна перевести в веб формат, та відкривати в браузері на кожній окремій платформі, але продуктивність такого рішення не буде оптимальним. Для вирішення цієї задачі в наш час все більше використовуються платформи та фреймворки для розробки кросплатформових застосунків.

Кросплатформові застосунки дозволяють використовувати одну кодову базу на декількох пристроях, що зменшує час розробки додатку, а отже створити систему з додатків, наприклад додаток для замовлення та додаток для доставки, буде легше, швидше та дешевше.

Кажучи про серверну частину, то можна створити систему на базі веб фреймворку, та використати екосистему бібліотек. Це зменшить час на розробку, але система буде залежна від сторонніх бібліотек, та стабільності їх роботи. Іншим підходом є створення всіх необхідних інструментів з нуля, що значно збільшить час розробки та вартість системи.

Для рішення проблеми створення безлічі додатків для системи доставки, було вирішено створити систему в якій користувачі зможуть додавати свої товари в список доставки, а користувачі які будуть виконувати доставку будуть отримувати замовлення на виконання, незалежно від пристрою яким вони користуються.

З точки зору розробки серверної частини системи, а саме покладаючись на ті самі принципи спрощення розробки та зменшення часу на сам процес, було вирішено використовувати веб - фреймворк з екосистемою бібліотек.

Актуальність дослідження: актуальність теми зумовлена високою вартістю розробки додатків для кожної платформи.

З огляду на це, можна виділити наступні цілі і задачі дослідження:

Мета: Розробка застосунку для замовлення та доставки товарів засобами Flutter та Laravel.

Задачі:

1) огляд фреймворку Flutter, його концепцій та аналогів;
2) сформулювати вимоги до системи;
3) спроєктувати та побудувати архітектуру системи;
4) реалізувати додаток для користувача;
5) реалізувати серверну складову системи.

Об’єкт дослідження – фреймворк Flutter, бібліотеки для роботи з ним, засоби обробки даних на стороні клієнта та на сервері.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проектування, конструювання та тестування програмного забезпечення.

Предмет дослідження – створення платформи, що дозволяє користувачам виконувати замовлення та доставки товарів у приватному сегменті.

Перший розділ присвячено огляду фреймворку Flutter, порівнянню з аналогами, огляду фреймворку Laravel.

У другому розділі схематично розглянуто етапи проєктування системи.

В межах третього розділу роботи розроблено систему обліку обладнання засобами Flutter та Laravel.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Flutter

1.1.1 Огляд фреймворку

Flutter – програмний каркас (фреймворк) із відкритим кодом компанії Google для створення додатків для платформ Android, iOS, Windows, MacOS та вебу [1].

Вперше даний фреймворк народився під ім’ям Sky в 2015 році. Працював на Android, а згодом був портований на iOS.

Перші версії були випущені одразу після анонсу а кульмінацією став випуск стабільної версії 4 грудя 2018 року. З того часу дуже багато розробників зацікавились цим фреймворком. Популярність Flutter можна назвати метиоритною, і на то є вагомі причини.

Одна з них це те, що першочерговою було заявлено, як один із каменів стояння, що Flutter буде підтримувати рендеринг інтерфейсу користувача з частотою 120 кадрів на секунду. Цієї мети досягають лише деякі крос-платформові фреймворки.

Крім того відрізняється принцип побудови компонентів інтерфейсу. Flutter надає користувачу власні компоненти на від аналогів. Іншими словами коли ви хочете щоб Flutter відобразив кнопку він відображає її сам, а не просить це зробити операційну систему, як це робить Xamarin та React Native. Саме такий підхід робить інтерфейс користувача однаковим на різних пристроях. Важливо що компоненти можуть бути додані до фреймворку легко та швидко, не хвилюючись про їх підтримку операційною системою.

Це також дозволяє Flutter надавати користувачу специфічні набори компонентів в стилістиках Material (реалізує стиль Android за замовчування) та Cupertino (реалізує стиль iOS).

Flutter розроблений як розширювана, система шарів. Він існує як низка незалежних бібліотек, кожна з яких залежить від базового шару. Жоден шар не має привілейованого доступу до шару нижче, і кожна частина рівня рамки розроблена як додаткова і змінна.

До базової операційної системи додатки Flutter упаковуються так само, як і будь-яка інша вбудована програма. Вбудова, що стосується платформи, забезпечує точку входу; координує роботу з базовою операційною системою для доступу до таких послуг, як надання поверхонь, доступність та вхід; і керує циклом події повідомлення. Вбудова написана мовою, відповідною для платформи: в даний час Java та C ++ для Android, Objective-C / Objective-C ++ для iOS та macOS та C ++ для Windows та Linux. Використовуючи вбудовувач, код Flutter може бути інтегрований у існуючу програму як модуль, або код може бути усім вмістом програми. Flutter включає ряд вбудовувачів для загальних цільових платформ, але існують і інші вбудовувачі (див. рис. 1.1)[2].

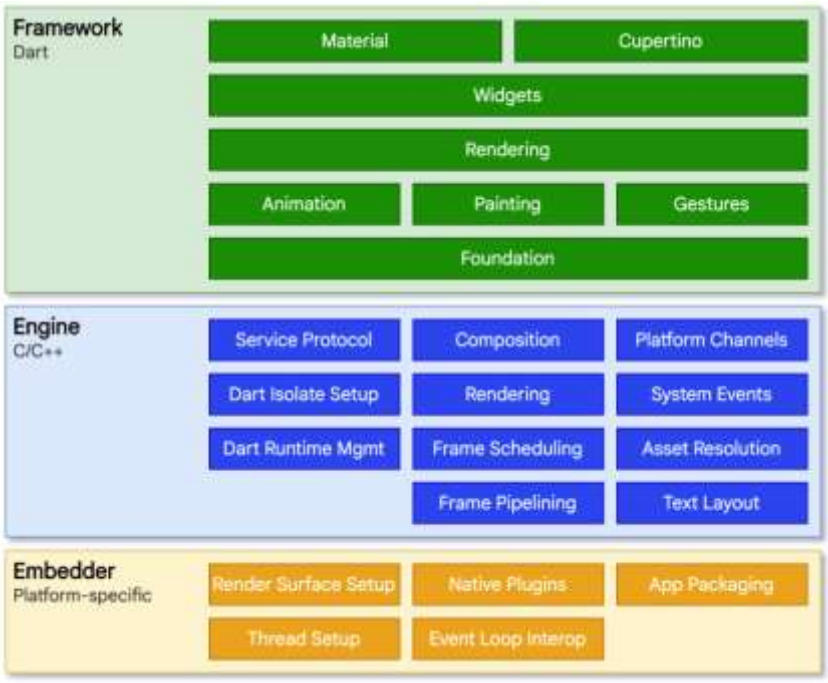

Рисунок 1.1 – Архітектурні шари Flutter

В своїй основі Flutter використовує рушій, що здебільшого написаний на C++ та з використанням бібліотеки Skia. Skia – це компактна графічна бібліотека написана на тій же мові що й двигун. Але окрім Flutter вона ще використовується у Google Chrome, Chrome OS, Chromium OS, Mozilla Firefox, Android (частково замінена на HWUI, починаючи з Android 3.0), Firefox OS, Flutter, LibreOffice (починаючи з версії 7.0) та Sublime Text 3. Якщо ж говорити про внутрішню будову Skia має звичайний процес рендеру на основі процесора, але нещодавно була інтегрована нова внутрішня частина, яка прискорена OpenGL ES2, під назвою Ganesh. Ganesh експериментував з двома прискореними підходами. Перший використовував трафаретний буфер для рендеру шляхів, через накладні витрати API був замінений другим підходом. Другий використовує процесорний растеризатор, обчислює маску покриття, яка завантажується як текстура на кожному із шляхів, щоб забезпечити належне згладжене покриття GPU [3].

Крім того Flutter надає уніфікований доступ до можливостей підтримуваних операційних систем. Іншими словами код для запуску камери на Android та на iOS буде однаковим, а для цього можна використати готові методи Flutter.

Коротше кажучи Flutter – це комплект для розробки програмного забезпечення, що включає в себе компоненти інтерфейсу, двигун рендеру, інструменти тестування, інструменти маршрутизації та багато чого іншого.

1.1.2 Мова програмування Dart

Фреймворк Flutter написано з використанням мови програмування під назвою Dart. Застосунок Flutter складається з коду Dart. І якщо поглянути з такої сторони то Flutter це лише бібліотека Dart.

Dart був створений в компанії Google в 2011 році. Реліз першої версії вийшов в листопаді 2013 року. Цікаво, що мова Dart була розроблена Ларсом

Баком, а ще він розробив JavaScript двигун V8, що використовується в Chrome та Node.js.

```dart
import "dart:math" as math

class Point {
    final num x,y;
    Point (this.x, this.y);
    Point.origin() : x=0, y=0;
    num distanceTo(Point other) {
        var dx = x - other.x;
        var dy = y - other.y;
        return math.sqrt(dx * dx + dy * dy);
    }
    Point operator + (Point other) => Point(x + other.x, y = other.y);
}

void main() {
    var p1 = Point(10, 10);
    var p2 = Point.origin();
    var distance = p1.distanceTo(p2);
    print(distance);
}
```

Рисунок 1.2 – Приклад коду на мові Dart

Dart – це лаконічна мова, яка набирає оберти. Оскільки це мова загального призначення, її широко використовують для створення веб застосунків, серверного коду, та застосунків «інтернету речей». І хоча Flutter основна причина популярності мови Dart, він розвивається в різних напрямках, та має спільноту, що дуже швидко розвивається. У наведеному фрагменті коду можна побачити ключові моменти синтаксису, та деякі з переваг (див. рис. 1.1):

- Dart повністю об’єктно-орієнтована мова, це навіть дозволяє писати користувацький інтерфейс без використання жодної мови розмітки;
- мова включає в себе збиральника сміття, тому не має необхідності слідкувати за пам’ятю;
- систаксис основано на мові C, що підходить більшості розробників, але є ряд особливостей які спочатку можуть збити з толку;

- мова підтримує загальні конструкції, такі як інтерфейси, наслідування, абстрактні класи, шаблонні класи (generics) та статичну типізацію;
- Dart містить перевірку відповідності типів;
- підтримує багатопотоковість;
- Dart підтримує як динамічну компіляцію так і компіляцію при збирані застосунку:
    - компіляція при збирані (Ahead of time compilation) переводить Dart в байтовий код, що наближає його за продуктивністю до нативних платформ, настільки близько накільки можливо;
    - динамічна компіляція (Just in Time compilation) дозволяє використовувати гаряче перезавантаження при розробці для зменшення часу розробки та збільшення швидкості тестування.
- велика кількість репозиторіїв з готовими бібліотеками, що забезпечують додаткову функціональність;
- ядро Dart підтримує створення зліпків (snapshot) які дозволяють запаковати весь код що виконується в єдиний двійковий файл, що прискорює запуск застосунку [4].

1.1.3 Принцип роботи Flutter

На високому рівні Flutter – це реактивна, декларативна та композитна бібліотека шару представлення, дуже схожа на ReactJS в Інтернеті (але більше схожа на React, змішаний з браузером, тому що Flutter також є повноцінним рушієм рендерингу). У двох словах, ви створюєте мобільний інтерфейс шляхом поєднуючи разом купу менших компонентів, які називаються віджетами. Все є віджетом, а віджети - це просто класи Dart, які знають, як описати своє представлення. Структура визначається за допомогою віджетів, стилі визначаються за допомогою віджетів, так само як і анімація і все, що ви можете придумати, з чого складається інтерфейс. Це основна ідея Flutter. Все

є віджетом. Знову ж таки, це не означає, що у Flutter немає інших типів об'єктів – вони є. Справа в тому, що у Flutter немає моделей і моделей представлення або будь-якого іншого специфічного типу класів. Віджет може визначати будь-який аспект вигляду програми. Деякі віджети, такі як *Row*, визначають аспекти макета. Деякі менш абстрактні і визначають структурні елементи, такі як *Button* та *TextField*. Навіть корінь вашої програми є віджетом (див. рис. 1.3).

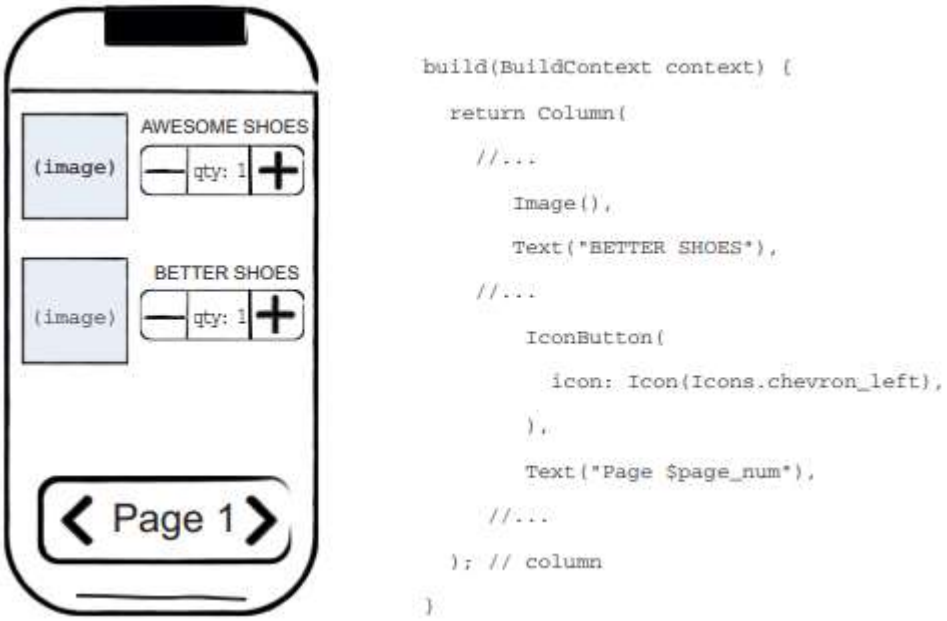

Рисунок 1.3 – Зразок компонування віджетів

Віджети, як правило, складаються з багатьох інших невеликих, одноцільових віджетів, які в поєднанні дають потужні ефекти.

Там, де це можливо, кількість концепцій дизайну зводиться до мінімуму, але при цьому загальний словниковий запас може бути великим. Наприклад, у шарі віджетів Flutter використовує одну і ту ж основну концепцію (віджет) для представлення відображення на екрані, макета (позиціонування і розмір), інтерактивності користувача, управління станом, тематики, анімації та навігації. У шарі анімації пара концепцій, Анімації та Твіни, охоплюють більшу частину простору дизайну. У шарі рендерингу RenderObjects

використовуються для опису макета, малювання, хіт-тестування та доступності. У кожному з цих випадків відповідний словниковий запас виявляється великим: існують сотні віджетів і об'єктів рендерингу, десятки типів анімації[5].

Ієрархія класів навмисно неглибока і широка, щоб максимізувати можливу кількість комбінацій, зосереджуючись на невеликих віджетах, що складаються, кожен з яких добре робить щось одне. Основні функції є абстрактними, навіть базові функції, такі як відступ та вирівнювання, реалізовані як окремі компоненти, а не вбудовані в ядро. (Це також контрастує з більш традиційними API, де такі функції, як відступи, вбудовані в загальне ядро кожного компонента макета). Так, наприклад, щоб вирівняти віджет по центру, замість того, щоб регулювати умовну властивість *Align*, ви обертаєте його у віджет *Center*.

Існують віджети для відступів, вирівнювання, рядків, стовпців і сіток. Ці віджети макета не мають власного візуального представлення. Замість цього, їх єдиною метою є управління деякими аспектами макета іншого віджета. Flutter також включає в себе службові віджети, які використовують переваги цього композиційного підходу.

Наприклад, віджет *Container*, який часто використовується, складається з декількох віджетів, що відповідають за макет, малювання, позиціонування та розмір. Зокрема, *Container* складається з віджетів *LimitedBox, ConstrainedBox, Align, Padding, DecoratedBox* та *Transform*, як можна побачити з його вихідного коду. Визначальною характеристикою Flutter є те, що ви можете заглибитися у вихідний код будь-якого віджета і вивчити його. Таким чином, замість того, щоб підкласифікувати *Container* для отримання індивідуального ефекту, ви можете скомпонувати його та інші віджети новими способами або просто створити новий віджет, використовуючи Container як натхнення.

Ви визначаєте візуальне представлення віджету, перевизначаючи функцію build(), яка повертає нове дерево елементів. Це дерево представляє частину віджету в інтерфейсі користувача у більш конкретних термінах.

Наприклад, віджет панелі інструментів може мати функцію побудови, яка повертає горизонтальне розташування деякого тексту та різних кнопок. За необхідності, фреймворк рекурсивно запитує кожен віджет на побудову, поки дерево не буде повністю описано конкретними об'єктами, які можна візуалізувати. Потім фреймворк зшиває об'єкти, які можна візуалізувати, у дерево об'єктів, які можна візуалізувати.

Функція побудови віджету не повинна мати побічних ефектів. Кожного разу, коли функція запитується на збірку, віджет повинен повертати нове дерево віджетів, незалежно від того, що віджет повертав раніше. Фреймворк виконує важку роботу, визначаючи, які методи збірки потрібно викликати на основі дерева об'єктів рендерингу [6].

1.1.4 Механізм рендеру

Ви можете задатися питанням: якщо Flutter є крос-платформним фреймворком, то як він може запропонувати продуктивність, порівнянну з одноплатформними фреймворками? Корисно почати з роздумів про те, як працюють традиційні додатки для Android. При малюванні ви спочатку викликаєте Java-код фреймворку Android. Системні бібліотеки Android надають компоненти, відповідальні за малювання, в об'єкт Canvas, який Android потім може візуалізувати за допомогою Skia, графічного движка, написаного на C/C++, який звертається до центрального або графічного процесора для завершення малювання на пристрої.

Крос-платформні фреймворки зазвичай працюють шляхом створення шару абстракції над базовими власними бібліотеками інтерфейсу Android та iOS, намагаючись згладити невідповідності представлення кожної платформи. Код програми часто пишеться на інтерпретованій мові, такій як JavaScript, яка, в свою чергу, повинна взаємодіяти з системними бібліотеками Android на основі Java або iOS на основі Objective-C для відображення інтерфейсу

користувача. Все це додає накладних витрат, які можуть бути значними, особливо там, де є багато взаємодії між інтерфейсом і логікою програми.

На відміну від цього, Flutter мінімізує ці абстракції, обходячи системні бібліотеки віджетів інтерфейсу на користь власного набору віджетів. Код Dart, який малює візуальні ефекти Flutter, компілюється у власний код, який використовує Skia для рендерингу. Flutter також вбудовує власну копію Skia як частину рушія, що дозволяє розробнику оновлювати свій додаток, щоб залишатися в курсі останніх поліпшень продуктивності, навіть якщо телефон не був оновлений до нової версії Android. Те ж саме стосується Flutter на інших нативних платформах, таких як iOS, Windows або macOS [7].

1.1.5 Переваги Flutter

Перелічимо основні з переваг Flutter. Першою з них можна назвати те, що фреймворк дає вам повний контроль, щодо додавання нових плагінів. Насправді багато плагінів для взаємодії з системою та сервісами вже реалізовано (Google Maps, керування камерою, зберігання даних). Але нічого вам не заважає створити власний плагін або віджет (у цьому випадку сам фреймворк надає всі необхідні інструменти).

Наступною перевагою можна назвати відсутність мосту JavaScript. Міст JavaScript, який використовується в більшості крос-платформних варіантів, є основним вузьким місцем в розробці та продуктивності вашого додатку. Прокрутка не є плавною, додатки не завжди продуктивні, і їх важко налагоджувати. Flutter компілюється в фактичний нативний код і рендериться за допомогою власного рушія, тому немає необхідності перекладати Dart під час виконання. Це означає, що додатки не втрачають продуктивність або продуктивність під час роботи на пристрої користувача.

Наступна перевага це час компіляції. У Flutter повна компіляція зазвичай займає менше 30 секунд, а інкрементна компіляція займає менше секунди завдяки гарячому перезавантаженню [8].

Наступна перевага відноситься до багатьох крос-платформових фреймворків, але в нашому випадку є нюанси. А сама перевага це спільне використання коду. Тут я буду справедливим: Я припускаю, що це технічно можливо і в JavaScript. Але це, безумовно, неможливо в нативній розробці. За допомогою Flutter і Dart ваші веб і мобільні додатки можуть ділитися всім кодом, за винятком представлень кожного клієнта. Ви можете використовувати ін'єкцію залежностей, щоб запустити додаток AngularDart і додаток Flutter з тими ж моделями і контролерами. І, очевидно, навіть якщо ви не хочете ділитися кодом між вашим веб-додатком і мобільним додатком, ви ділитеся всім своїм кодом між додатками для iOS і Android [9].

На практиці це означає, що ви надзвичайно продуктивні. Оскільки ми ділимося бізнес-логікою між веб та мобільними додатками, після того, як мобільна функція реалізована, потрібно лише написати подання для веб, які очікують ті ж самі дані контролера.

1.2 Порівняння з аналогами

1.2.1 Загальний огляд аналогів

Перші "мобільні додатки", які були побудовані крос-платформенними, були просто веб-переглядами, які працювали на WebKit (движок рендерингу браузера). Це були буквально вбудовані веб-сторінки. Проблема з цим в основному полягає в тому, що маніпулювання DOM дуже дороге і не працює досить добре, щоб забезпечити відмінний мобільний досвід.

Деякі платформи вирішили цю проблему шляхом створення моста JavaScript, який дозволяє JavaScript спілкуватися безпосередньо з нативним кодом. Це набагато продуктивніше, ніж веб-переглядачі, оскільки виключає DOM з рівняння, але все одно не є ідеальним. Кожен раз, коли вашому додатку потрібно звертатися безпосередньо до движку рендеринга, його потрібно

компілювати в нативний код, щоб "перетнути міст". За одну взаємодію міст повинен бути перетнутий двічі: один раз від платформи до програми, а потім назад від програми до платформи.

Іншим рішенням було піти шляхом Flutter. Деякі фреймворки робили це, але в силу тих чи інших причин не стали такими популярними.

Давайте коротко оглянемо найближчих конкурентів Flutter на ринку крос-протформової розробки. А потім порівняємо за продуктивністю найбільш популярні рішення.

1.2.2 React Native

React Native – це кросплатформений фреймворк із відкритим вихідним кодом для розроблення мобільних і настільних застосунків на JavaScript і TypeScript, створений Facebook. React Native підтримує такі платформи, як Android, Android TV, iOS, macOS, tvOS, Web, Windows і UWP, даючи змогу розробникам використовувати можливості бібліотеки React поза браузером для створення нативних застосунків, що мають повний доступ до системних API платформ.

Мови програмування: JavaScript, HTML, CSS, TypeScript.

Основні принципи роботи React Native практично ідентичні принципам роботи React, за винятком того, що React Native керує не браузерною DOM, а платформними інтерфейсними компонентами. JavaScript-код, написаний розробником, виконується у фоновому потоці, і взаємодіє з платформними API через асинхронну систему обміну даними, яка називається Bridge.

Хоча система стилів (спосіб конфігурації візуальних властивостей елементів інтерфейсу) React Native має синтаксис, схожий на CSS, фреймворк не використовує технології HTML або CSS як такі. Натомість для кожної з підтримуваних фреймворком операційних систем реалізовано програмні адаптери, що застосовують заданий розробником стиль до платформного

інтерфейсного елемента. Що вже є відмінністю від Flutter який надає власний перелік віджетів, а не використовує елементи платформи.

React Native також дає змогу розробникам використовувати вже наявний код, написаний іншими мовами програмування - наприклад, Java або Kotlin для Android і Objective-C або Swift для iOS. Також React Native підтримує інтеграцію в уже наявні застосунки - наприклад, частина інтерфейсу мобільного застосунку може бути реалізована на React Native, а частина - за допомогою суто платформних засобів [10].

1.2.3 Ionic

Ionic – це набор компонентів CSS та JS для розробки гібридних мобільних застосунків. З Ionic можна створювати кросплатформені програми. Він тісно взаємодіє з фреймворком Apache Cordova, який перетворює веб – додатки в мобільні програми.

Мови програмування: JavaScript, HTML, CSS.

Ionic, як і React Native і Flutter, пропонує концепцію єдиного коду для різних платформ, але на новому рівні. Всі його компоненти автоматично адаптуються до платформи, на якій запускається додаток, а отже, розробка стає швидшою. Також з Ionic ви можете вільно використовувати фреймворки JavaScript, такі як: Angular, React або Vue. А ось в продуктивності Ionic програє і сильно відстає від Flutter оскільки для візуалізації додатків він використовує веб-технології та зовсім не застосовує власні компоненти. Такий підхід значно знижує швидкість роботи.

Але з боку розробки є і плюси: Ionic дозволяє проводити швидке тестування, яке можна запустити прямо в браузері [11].

1.2.4 Xamarin

Xamarin – платформа для створення кросплатформових додатків від Microsoft. Найближчий аналог Flutter. З конкурентів вирізняється відстуністю мосту JavaScript.

Мова програмування: C #.

У Xamarin є два основні інструменти: Xamarin.Android/iOS і Xamarin.Forms. Xamarin.Android і Xamarin.iOS наділяють додаток тими ж можливостями і інтерфейсом, які є у нативних рішень. У разі Xamarin.iOS програма компілюється безпосередньо в машинний код (AOT-компіляція), тоді як в Xamarin.Android спочатку відбувається компіляція в байт-код, який потім інтерпретується віртуальною машиною (JIT-компіляція).

Якщо ж потрібно прискорити процес написання коду, краще використовувати Xamarin.Forms - більш простий інструмент, в якому майже всі елементи повністю сумісні з будь-якими платформами.

Продуктивність Xamarin також вважається близькою до нативної, але залежить від того, використовуєте ви Xamarin.Android, Xamarin.iOS або Xamarin.Forms. У Xamarin.Android/iOS хороша оптимізація завдяки нативним компонентів. Xamarin.Forms же заснований на 100% спільному використанні коду, що в цілому знижує його продуктивність в порівнянні з Xamarin.Android/iOS.

Не знайшов особливої популярності через специфіку платформи C#, та деякі архітектурні рішення обрані при розробці. Наприклад використання окремої мови верстки інтерфейсу користувача.

Хоча фреймворк як і сама платформа розвивається і можливо в майбутньому він перейме деякі рішення у Flutter [12].

1.2.6 Порівняння продуктивності

Як видно із порівняння з аналогами Flutter є одною з найбільш перспективніших технологій для розробки крос-платформових додатків.

Та давайте порівняємо продуктивність двох крос-платформових рішень(Flutter та React Native) з нативною розробкою [13].

Першим тестом буде оцінка продуктивності при рендері списку. Відображати будемо список з 1000 елементів та автоматично прокручувати список до останього елементу.

Таблиця 1.1 – Рендер списку з 1000 елементів на Android

Назва інструменту	Кількість кадрів	Використання процесору %	Максимальне значення використання ОЗП	Використання батареї
Native (Android)	60	2.4	58	49.7 mAh
React Native	58	11.7	139	79.01 mAh
Flutter	60	5.4	114	65.28 mAh

Подивившись на результати можемо побачити що React Native використовує більше ресурсів, що викликано використанням JavaScript мосту. Також через те, що міст виконує запит до платформи та назад використання батареї також більше, а отже пристрій протримається меньше часу на одному заряді. Це може бути критично при довгостроковому використанні застосунку.

Таблиця 1.2 – Рендер списку з 1000 елементів на iOS

Назва інструменту	Кількість кадрів	Використання процесору %	Використання відеокарти %	Використання ОЗП
Native (iOS)	60	12,72	21,24	154
React Native	59	113,13	19,56	220
Flutter	60	33,3	10,75	159

Як можна побачити з результатів тестування на iOS Flutter знову показує себе краще ніж React Native. Цікавий також момент, що нативна розробка на мові Swift використовує більше ресурси відеокарти, а Flutter відображає власні компоненти, а отже не використовує компоненти операційної системи заточені на використання відеокарти, і використання процесору більше.

Наступним тестом буде одночасне обертання 200 картинок на екрані

Таблиця 1.3 – Обертання 200 картинок на Android

Назва інструменту	Кількість кадрів	Використання процесору %	Максимальне значення використання ОЗП
Native (Android)	58	6,53	80
React Native	7	8,5	424
Flutter	19	10,28	168

І в даному тесті Flutter знову показує гарний результат. Цікаво після даного тесту згадати обіцянки команди фреймворку про стабільні 120 кадрів в секунду. Хоча в даному тесті цей параметр сягнув лише 19 пунктів, але використання ОЗП змушує радіти.

Таблиця 1.4 – Обертання 200 картинок на iOS

Назва інструменту	Кількість кадрів	Використання процесору %	Використання відеокарти %	Використання ОЗП
Native (iOS)	59	61	48,28	158
React Native	59	118,6	19,8	220
Flutter	59	69	81,91	191

При проведені тесту на iOS отримані результати показують, що наблизився за використанням ресурсів до нативної розробки. І як можна побачити використання відеокарти у фреймворку Flutter в даному випадку більше ніж в додатку на мові Swift. І якщо згадати як працює механізм рендеру, то це логічний результат.

1.3 Фреймворк Laravel

1.3.1 Загальний огляд фреймворку

Laravel – це безкоштовний PHP фреймворк загального призначення з відкритим кодом, який з'явився на світ у 2011 році, але, завдяки стрімким темпам розвитку та величезній армії прихильників, сьогодні він є одним із найпопулярніших PHP-движків.Самі творці Laravel назвали його "framework for artisans", що в перекладі означає "фреймворк для ремісників", натякаючи на те, що ця платформа дає розробникам повну свободу творчості, не створюючи перед ними жодних перешкод у процесі розробки [14].

Уже наприкінці 2013 року Laravel мав версію 4.1 і був названий "найбільш багатообіцяючим проектом на 2014 рік" за версією sitepoint.com.

До речі, на сьогоднішній день поточною версією фреймворка є 9 з використанням PHP версій 8.0 – 8.1.

Мова PHP – скриптова мова програмування, була створена для генерації HTML-сторінок на стороні вебсервера. PHP є однією з найпоширеніших мов, що використовуються у сфері веброзробок (разом із Java, .NET, JavaScript, Python, Ruby). PHP підтримується переважною більшістю хостинг-провайдерів. PHP – проєкт відкритого програмного забезпечення.

PHP інтерпретується вебсервером у HTML-код, який передається на сторону клієнта. На відміну від скриптової мови JavaScript, користувач не бачить PHP-коду, тому що браузер отримує готовий html-код. Це є перевагою з

точки зору безпеки, але погіршує інтерактивність сторінок. Але ніхто не забороняє використовувати PHP для генерування JavaScript-кодів, які виконуються вже на стороні клієнта.

Кажучи про Laravel, буде помилкою не відзначити його плюси, які власне й опишуть цей фреймворк:

- MVC (Model – View - Controller) структура коду – архітектурний шаблон, який використовується під час проєктування та розробки програмного забезпечення. Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача .Мета шаблону – гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності;
- Eloquent ORM, що входить до складу Laravel, надає красиву, просту реалізацію ActiveRecord для роботи з вашою базою даних. Кожна таблиця бази даних має відповідну "Модель", яка використовується для взаємодії з цією таблицею;
- у Laravel з коробки доступні інструменти організації черг процесів (наприклад, для масового надсилання email).Ця функція незамінна для HighLoad-проектів, оскільки дає змогу розвантажити сервер від постійної роботи;
- міграції – Свого роду, контроль версій для структури таблиць БД. Кожен файл міграції містить або структуру таблиць, або зміни її

структури.Тобто процес створення нових сутностей БД у Laravel фреймворку є створенням міграції та запуском її за допомогою спеціальних консольних команд artisan;

- artisan - це консоль Laravel, в арсеналі команд якої є робота з міграціями, контролерами і моделями, авторизацією та іншими базовими компонентами фреймворка;
- спільнота – даний фреймворк має дуже велику базу поціновувачів. Стрімко розвивається, а завдяки відкритому коду кожен може стати частиною команди розробки.

1.3.2 Огляд екосистеми Laravel

За останні роки фреймворк Laravel перейшов від стану звичайного фреймворку до чогось більшого. Багато хто називає сам фреймворк та широку коло його бібліотек екосистемою. Та й на офіційному сайті фреймворку про це сказано. І дійсно виходячи з широкого кола бібліотек які створені спільнотою, може здатися, що це організм, або екосистема в центрі якої розташовано фреймворк. Оглянемо деякі з частин екосистеми [15]:

- Inertia – це новий підхід до побудови класичних серверних веб-додатків. Inertia дозволяє створювати повністю рендерингові односторінкові додатки на стороні клієнта, без особливих складнощів, які притаманні сучасним SPA. Це досягається за рахунок використання існуючих серверних фреймворків. Inertia не має маршрутизації на стороні клієнта і не вимагає API. Це одна з частин екосистеми яка має спірне відношення до неї самої, але не згадати про дану технологію було б помилкою;
- Livewire – це повностековий фреймворк для Laravel, який робить створення динамічних інтерфейсів простим, не виходячи з комфорту Laravel;

- Laravel Breeze - це мінімальна, проста реалізація всіх функцій аутентифікації Laravel, включаючи вхід, реєстрацію, скидання пароля, перевірку електронної пошти та підтвердження пароля. Крім того, Breeze включає просту "профільну" сторінку, де користувач може оновити своє ім'я, адресу електронної пошти та пароль;
- Laravel Sanctum забезпечує легку систему аутентифікації для SPA (односторінкових додатків), мобільних додатків і простих API на основі токенів. Sanctum дозволяє кожному користувачеві вашого додатку генерувати кілька токенів API для свого облікового запису. Цим токенам можуть бути надані можливості / сфери застосування, які визначають, які дії токенам дозволено виконувати;
- Laravel Dusk надає виразний, простий у використанні API для автоматизації та тестування браузерів. За замовчуванням Dusk не вимагає встановлення JDK або Selenium на локальному комп'ютері. Замість цього Dusk використовує автономну установку ChromeDriver. Однак ви можете використовувати будь-який інший драйвер, сумісний з Selenium, за вашим бажанням;
- Laravel Horizon надає красиву інформаційну панель і конфігурацію, керовану кодом, для ваших черг Redis на базі Laravel. Horizon дозволяє легко відстежувати ключові показники вашої системи черг, такі як пропускна здатність, час виконання і збої в роботі. При використанні Horizon вся конфігурація працівника черги зберігається в одному простому файлі конфігурації. Визначивши конфігурацію працівника вашої програми у файлі з контролем версій, ви можете легко масштабувати або змінювати працівників черги вашої програми під час розгортання вашої програми;
- Laravel Scout надає просте рішення на основі драйверів для додавання повнотекстового пошуку до ваших моделей Eloquent. Використовуючи спостерігачі моделей, Scout автоматично синхронізує ваші пошукові індекси з вашими записами в Eloquent. В

даний час Scout поставляється з драйверами Algolia, MeiliSearch і MySQL / PostgreSQL (бази даних). Крім того, Scout включає в себе "колекційний" драйвер, який призначений для використання локальними розробниками і не вимагає ніяких зовнішніх залежностей або сторонніх сервісів. Крім того, написання користувацьких драйверів є простим, і ви можете вільно розширювати Scout власними реалізаціями пошуку;

- На додаток до типової аутентифікації на основі форм, Laravel також надає простий і зручний спосіб аутентифікації з постачальниками OAuth за допомогою Laravel Socialite. Наразі Socialite підтримує автентифікацію через Facebook, Twitter, LinkedIn, Google, GitHub, GitLab та Bitbucket;
- Laravel Sail - це легкий інтерфейс командного рядка для взаємодії з середовищем розробки Laravel Docker за замовчуванням. Sail є чудовою відправною точкою для створення Laravel-додатків з використанням PHP, MySQL і Redis, не вимагаючи попереднього досвіду роботи з Docker.В основі Sail лежить файл docker-compose.yml і скрипт sail, який зберігається в корені вашого проекту. Скрипт sail надає CLI зі зручними методами взаємодії з контейнерами Docker, визначеними файлом docker-compose.yml. Laravel Sail підтримується на macOS, Linux і Windows (через WSL2).

1.4 Технічне завдання

1.4.1 Введення

Система призначена для доставки товарів, а саме:

- додавати замовлення на доставку;
- видаляти замовлення на доставку;

- додавати, редагувати та видаляти точки маршруту для доставки;
- додавати, редагувати, та видаляти точки на карті, з які можна буде використати для більш швидкого заповнення маршруту доставки;
- переглядати на мапі місця звідки слід забрати та куди доставити об’єкт доставки;
- переглядати список доставок доступних для виконання;
- обирати один з можливих варіантів розміру об’єкту доставки;
- обирати доставку яку хочете виконати, за умови відповідності типу;
- отримувати список доставок до виконання.

Метою створення системи з точки зору творців системи є:

- створити систему, яку можна буде інтегрувати у виробничі процеси;
- навчитись працювати з новою технологією.

Метою створення системи з точки зору творців клієнта є:

- спростити роботу з даними, що стосуються доставок;
- отримати можливість створювати замовлення на доставку, коли це необхідно;
- отримати простий засіб роботи з даними.

Метою створення системи з точки зору організацій, що можуть інтегрувати системи:

- отримати зручний засіб контролю замовлень;
- сформувати базу клієнтів;
- отримати можливість надавати послуги, з використанням доставки, зменшивши видатки.

1.4.2 Детальні вимоги

Інтерфейс користувача:

- повинен бути адаптованим для використання більшості функцій одною рукою;

- повинен бути виконаним в стилі платформи на якій запушено застосунок;
- відображати повідомлення про помилки в зрозумілому для користувача вигляді;
- додаток повинен правильно відображати інтерфейс на діагоналях екрану від 4 дюймів до 13.

Функціональні вимоги з точки зору користувачів – замовників:

- створити замовлення на доставку ;
- видалити замовлення;
- переглянути статус доставки;
- вказати точки отримання об’єкту доставки
- вказати точки куди слід доставити об’єкт доставки;
- додати точки для швидкого заповнення форми замовлення
- мати можливість переглядати історію доставок;
- мати можливість повторити замовлення;
- обрати один з можливих варіантів оплати;
- перейти зі типу користувача-замовника на тип користувача-виконавця.

Функціональні вимоги з точки зору користувачів – виконавців

- перейти зі типу користувача-замовника на тип користувача-виконавця;
- переглянути список доставок до виконання;
- прийняти доставку;
- вказати власний діапазон цін, за одиницю доставки;
- переглянути точки отримання об’єкту доставки;
- переглянути точку доставки;
- перейти в сторонній застосунок для перегляду маршруту;
- переглядати історію виконаних доставок.

Нефункціональні вимоги:

- система повинна працювати, як на IOS, так і на Android;

- система повинна стабільно працювати з великою кількістю записів в базі даних;
- система повинна стабільно працювати для мінімум 400 одночасніх користувачів;
- система не повинна надавати доступ до інформації для неавторизованих користувачів.

1.5 Висновок до розділу 1

В розділі 1 було розглянуто проведено огляд фреймворку Flutter, його загальні принципи роботи. Розглянуто віджети, виділено їх сильні сторони. Переглянуто механізм рендеру інтерфейсу користувача на пристроях, та процес вбудови застосунку в платформу. Виділено переваги та недоліки фреймворку Flutter.

Проведено порівняння (як і концептуальне, так і за продуктивністю) Flutter з аналогами. Виділено сильні місця в яких даний фреймворк перемагає.

Проведено огляд фреймворку Laravel. Виділено його переваги. Надано інформацію щодо його екосистеми та описано її складові-бібліотеки.

Було проведено аналіз аналогічних технологій.

У першому розділі також представлено технічне завдання до розроблюваної системи.

Відповідно до результатів проведеного аналізу предметної області та доступних засобів реалізації додатку було виділено наступні технології для досягнення мети кваліфікаційної роботи:

- Flutter – для розробки мобільного застосунку;
- СУБД MySQL;
- веб-сервер засобами фреймворку Laravel;
- Laravel Sanctum для авторизації;
- Eloquent ORM для взаємодії з базою даних.

2 ПРОЄКТУВАННЯ

2.1 Загальна логіка системи

Система надає можливості для обліку обладнання. Серверна частина надає простий API, клієнтська частина надає інтерфейс користувача та взаємодію з API.

Для використання функціоналу користувачу слід авторизуватись в системі, за допомогою логіну та паролю. Користувача має можливість самостійно зареєструватись в системі.

Після реєстрації користувача, він отримує можливість обрати власну роль, як користувача, або це буде користувача який буде замовляти доставку певного товару, або речі, або безпосередньо буде виконувати доставку. Відповідно до ролі користувача у них будуть різні функціональні та дизайнерські особливості

2.2 Проєктування системи

Проєктування ПЗ – процес визначення архітектури, компонентів, інтерфейсів, інших характеристик системи й кінцевого результату [16].

Для проєктування інформаційних систем широко використовується мова UML – уніфікована мова візуального моделювання. Це мова для візуалізації, специфікації, конструювання і документування програмних систем [16].

Проєктування програмного забезпечення важливий процес для попереднього виявлення характеристик та можливих слабких місць майбутньої системи. На даному етапі можна, оцінити можливі варіанти архітектур, як систем, так і мобільних застосунків, оцінити можливі варіанти роботи користувача з системою, внести зміни до функціональних особливостей системи, ще до початку розробки системи.

Розробка різноманітних діаграм взаємодію компонентів в системі, потоків даних, взаємодії користувача з застосунками та безпосередньо з системою, дозволяє провести оцінки обсягу робіт, знайти розробників, або команду для виконання процесів створення та підтримки майбутнього продукту.

Також не варто забувати, що діаграма це зручний засіб відображення та доведення інформації до розробників, аналітиків, тестувальників і багатьох інших людей пов’язаних, залучених та зацікавлених в даній системі.

2.2.1 Етап концептуального проєктування

Даний етап дозволяє оцінити такі варіанти взаємодії як користувач – система, користувач – користувач, користувач – застосунок.

Як можна побачити зі списку варіантів взаємодії, то буде легко визначити, що саме користувач є основним об’єктом дослідження на даному етапі.

Тому, щоб відобразити усі можливі дії та процеси, що стосуються користувача було розроблено діаграму варіантів використання.

Діаграма варіантів використання (діаграма прецедентів) (див. рис. 2.1) – це діаграма, на якій зображено відношення між акторами та прецедентами в системі [17].

У той час як сам варіант використання може докладати багато деталей про кожну можливість, діаграма варіантів використання може допомогти забезпечити вищий рівень уявлення про систему. Завдяки своїй спрощеній природі діаграми варіантів використання можуть бути хорошим інструментом спілкування для зацікавлених сторін. Малюнки намагаються імітувати реальний світ і надати зацікавленій стороні бачення, щоб зрозуміти, як буде працювати розроблена система.

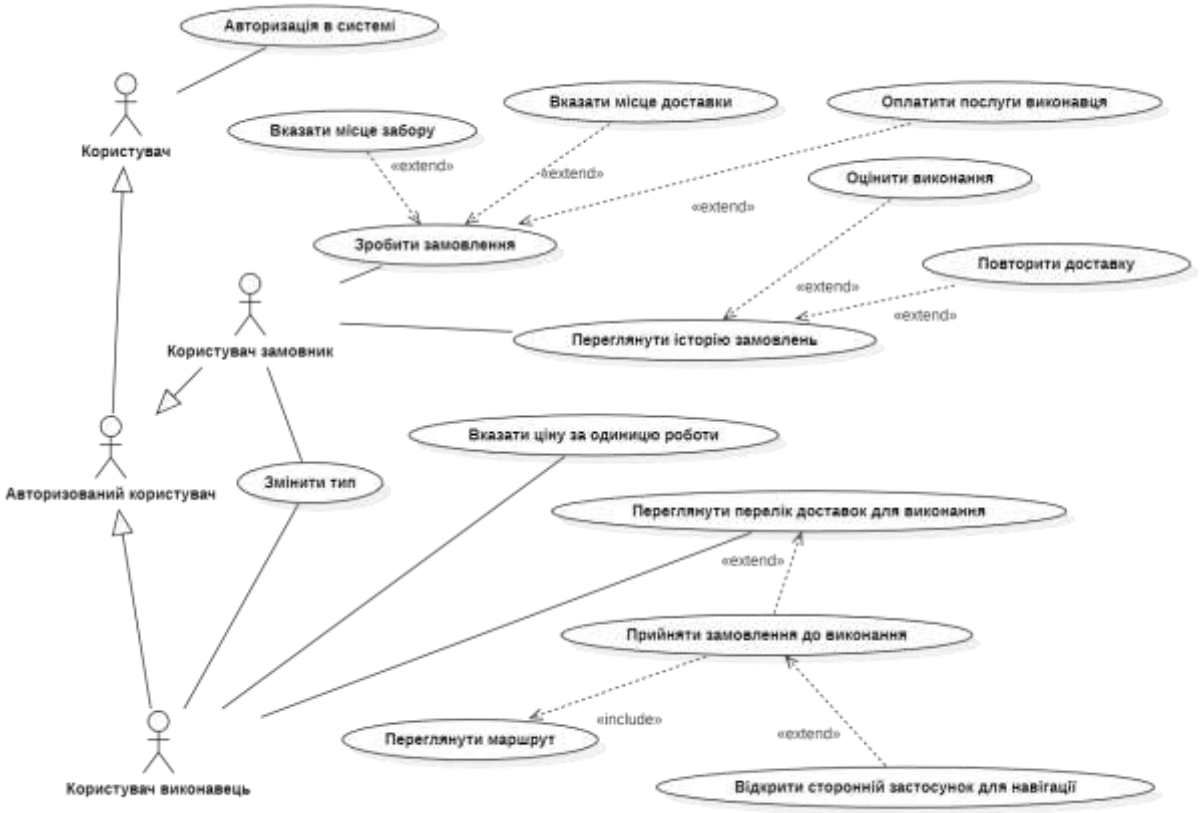

Рисунок 2.1 − Діаграма прецедентів

Доцільним буде описати основні варіанти використання майбутньої системи.

Прецедент «Змінити тип».

Призначення: даний варіант використання надає можливість авторизованому користувачу змінити його тип з користувача – замовника на користувача виконавця та навпаки.

Основний потік подій: прецедент починає виконуватися, коли користувач реєструється в системі та перенаправляється на екран вибору ролі. Також можливий другий варіант основного потоку коли користувач заходить до екрану профілю і натискає кнопку «Змінити тип». Система визначає поточний тип користувача та переводить користувача на інший.

Альтернативний потік: якщо користувач не авторизований перенаправити на екран авторизації.

Передумова: користувач повинен бути авторизованим у системі.

Прецедент «Зробити замовлення».

Призначення: прецедент надає можливість користувачу додати замовлення до списку для виконання.

Основний потік подій: даний варіант використання починає виконуватися, коли авторизований користувач головну сторінку застосунку та натискає кнопку зробити замовлення. Застосунок відображає форму замовлення де користувач може обрати розмір пакунку, або посилки, яку йому слід передати використовуючи сервіс доставки, через інтерфейс камери вказує початкову точку, та кінцеву точку доставки. Далі побачить час для виконання замовлення, та приблизну ціну виконання.

Альтернативний потік: користувач може вказати власну ціну за виконання замовлення, якщо запропонована застосунком його не влаштовує, але вона повинна бути в рамках між найменшою та найбільшою ціною вказаною користувачами, що виконують доставку.

Передумова: користувач повинен бути авторизованим у системі, та мати тип користувач – замовник.

Прецедент «Переглянути історію замовлень».

Призначення: даний прецедент надає можливість історію замовлень доставок який робив цей користувач.

Основний потік подій: варіант виконання починає виконуватися, коли авторизований користувач переходить на екран історії доставок.

Передумова: користувач повинен бути авторизованим у системі та мати тип користувач – замовник.

Прецедент «Повторити доставку».

Призначення: даний прецедент надає можливість повторно замовити таку ж доставку, яка вже була, з тими ж параметрами, але з іншою ціною в залежності від вказаної користувачами – виконавцями.

Основний потік подій: варіант виконання починає виконуватися, коли авторизований користувач – замовник переходить на екран історії замовлень доставок , натисненням на елемент списку обирає одну з доставок з переліку,

переходить на екран деталей доставки, натискає кнопку «Повторити», у відкритій формі підтверджує замовлення.

Альтернативний потік: якщо користувача не влаштовує нова ціна на доставку він може перервати процес повторного замовлення.

Передумова: користувач повинен бути авторизованим у системі та мати тип користувач – замовник.

Прецедент «Вказати ціну за одиницю доставки».

Тут варто додати пояснення про одиницю роботи. Під одиницею роботи варто вважати вагу за відстань, яку користувач – виконавець має перевезти. Будемо вважати, що це буде ціна за відношення ваги/відстань. А саме ціна за перевезення 1 кілограма ваги доставки на один кілометр.

Призначення: даний прецедент надає можливість користувачам – виконавцям вказати ціну за кілограм/кілометр перевезення об’єкту доставки.

Основний потік подій: варіант виконання починає виконуватись під час авторизації користувача після обрання типу користувач – виконавець, та переходу на екран доповнення реєстраційних даних. Заповнює форму цінами за кілограм/кілометр та натискає кнопку зберегти.

Альтернативний потік: користувач – виконавець може змінити ціни за одиницю доставки перейшовши на екран заповнення форми з екрану профіля користувача.

Передумова: користувач повинен бути авторизованим у системі та мати тип користувач – виконавець.

Прецедент «Переглянути перелік замовлень для виконання».

Призначення: даний прецедент надає можливість користувачам – виконавцям переглянути поточний перелік доставок можливих, для виконання.

Основний потік подій: варіант виконання починає виконуватись переходу на екран перегляду переліку доставок. Далі користувач отримує можливість вибрати доставку, переглянути точки доставки на карті та

відкрити маршрут, або перейти в сторонній застосунок для побудови маршруту доставки.

Альтернативний потік: користувач – виконавець може змінити ціни за одиницю доставки перейшовши на екран деталей доставки.

Передумова: користувач повинен бути авторизованим у системі та мати тип користувач – виконавець.

Також на даному рівні проєктування було розроблено діаграму послідовності [17].

На цій діаграмі (див. рис. 2.2) показано обмін повідомленнями (тобто виклик методів) між декількома об’єктами у окремій обмеженій часом ситуації, а конкретно − при авторизації користувача до системи.

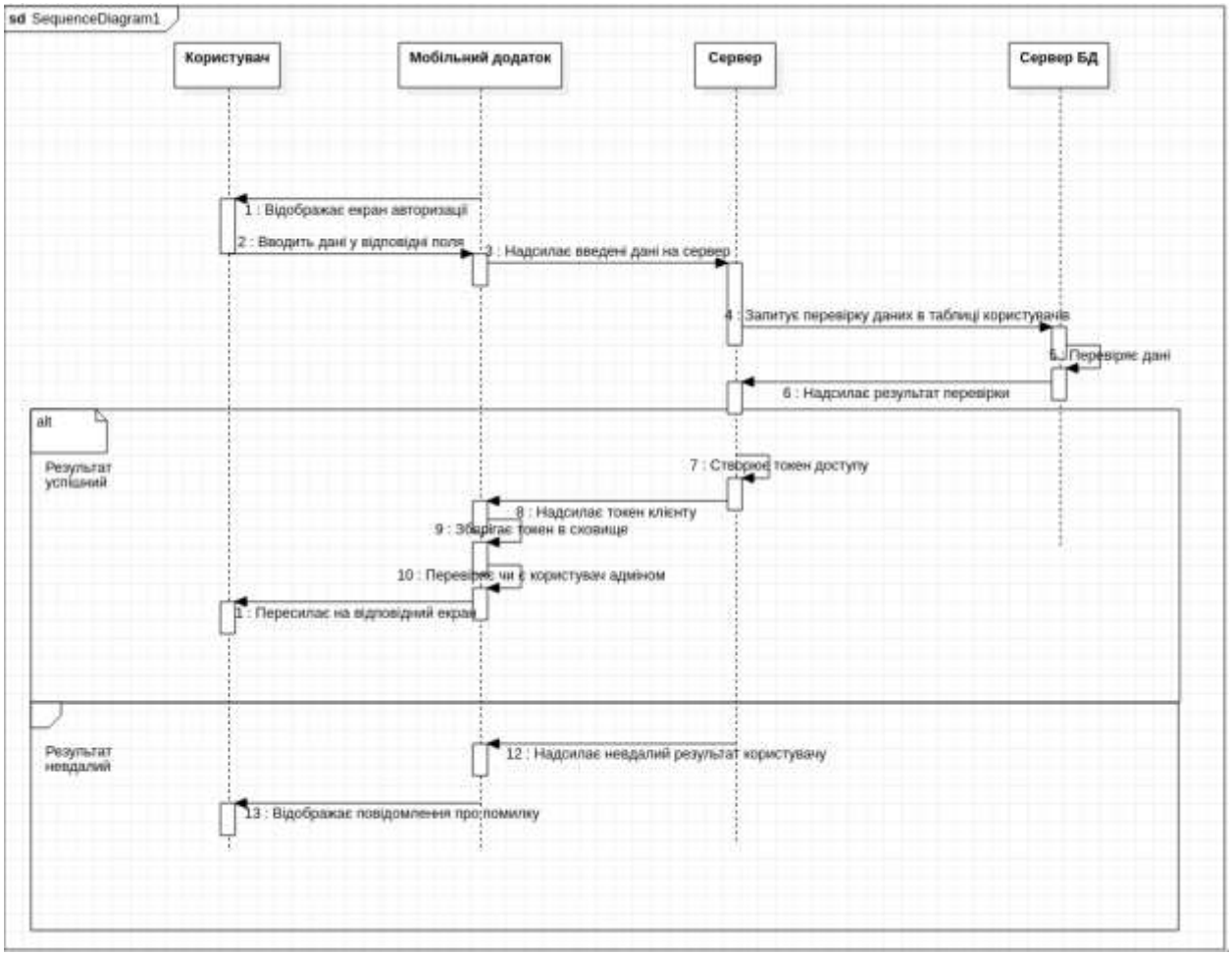

Рисунок 2.2 – Діаграма послідовності «Авторизація користувача»

Як можна побачити з діаграми послідовності, основою процесу авторизації, а після успішного завершення даного процесу, і взаємодії з системою буде процес обміну токенами між застосунком і сервером системи.

2.2.2 Етап логічного проєктування

На етапі логічного проєктування розроблено діаграму діяльності.

На ній (див. рис. 2.3) як і у випадку з попередньою діаграмою показано процес авторизації користувачем від моменту входу в систему до отримання повідомлення від серверу, або з токеном для подальшої взаємодії, або з помилкою під час процесу авторизації.

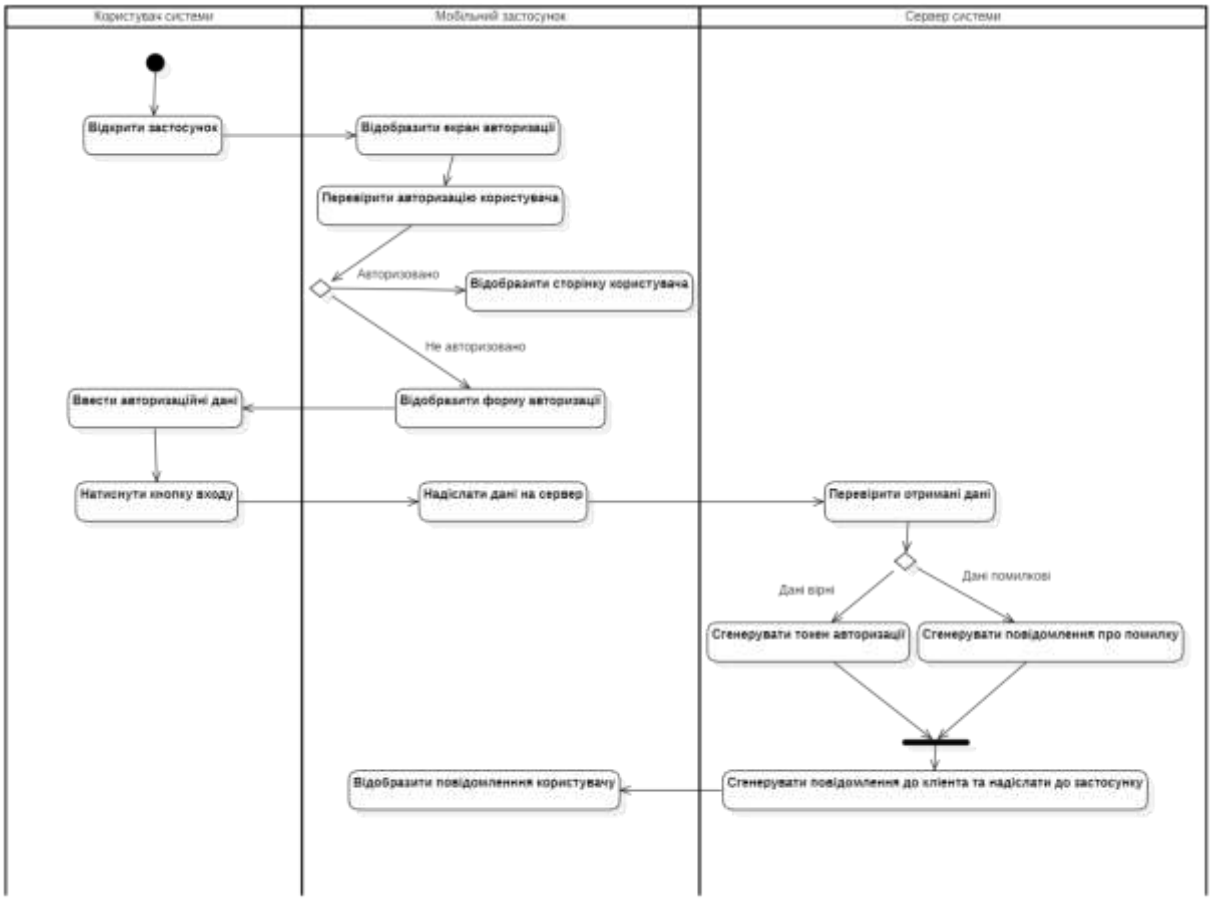

Рисунок 2.3 – Діаграма діяльності

2.2.3 Етап фізичного проєктування

Етап фізичного проєктування полягає в створенні діаграми розгортання, на якій зображені вузли, які є фізично існуючими елементами системи – сервер, сервер бази даних, персональний комп’ютер (див. рис. 2.4).

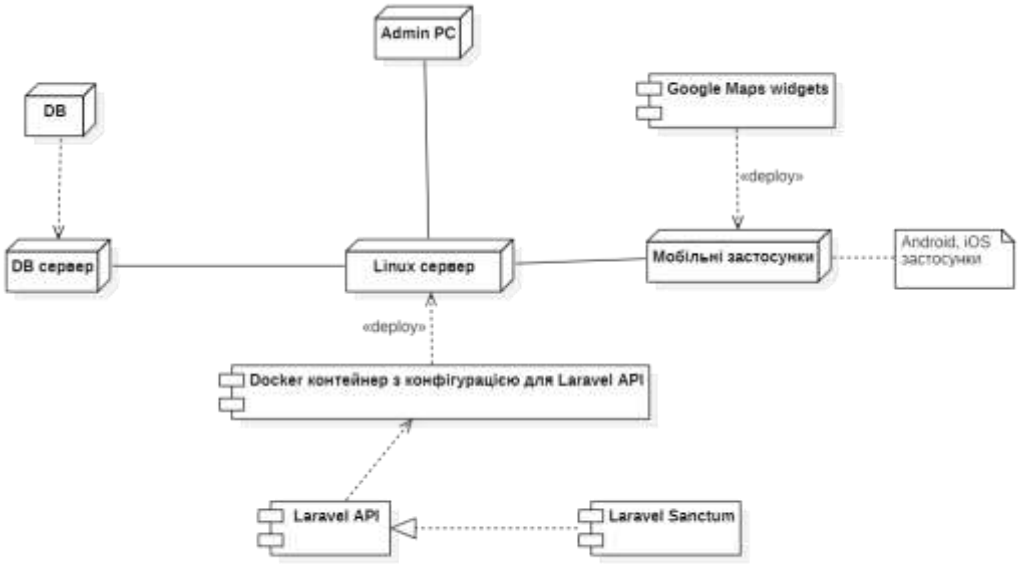

Рисунок 2.4 – Діаграма розгортання системи

Під час подальшого масштабування системи існує варіант збільшення ресурсів Linux серверу, або додавання, ще одного серверу з налаштуванням, балансування навантаження.

2.2.4 Проєктування архітектури за допомогою класів

На даному етапі було спроєктовано структуру класів, та було представлено за допомогою UML діаграми класів.

Діаграма класів – логічна модель базової структури системи, відображає статичну структуру системи та зв'язки між її елементами.

З огляду на те, що кожен віджет у фреймворку Flutter є окремим об’єктом класів, що наслідують клас Widget. Було б цікаво поглянути на кросплатформову

частину цієї частини системи. Зробимо це за допомогою діаграми класів, що відображає імплементацію шаблону проектування абстрактна фабрика Abstract Factory (див. рис. 2.5).

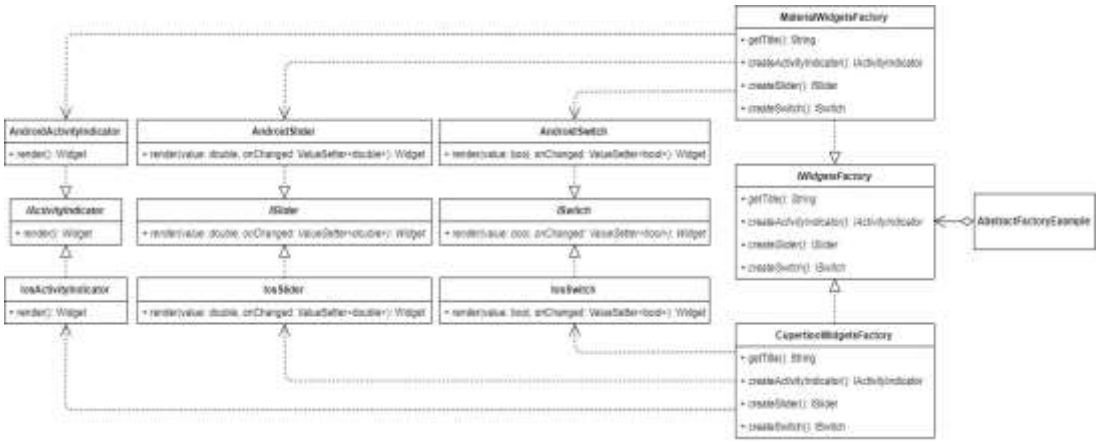

Рисунок 2.5 – Абстрактна фабрика для кросплатформового відображення компонентів

Дана діаграма надає вичерпне уявлення про процес реалізації кросплатформового рендеру для двох мобільних систем. Шаблон абстрактна фабрика було обрано саме через його пристосованість до створення сімейств об’єктів класів. Що є значною перевагою під час створення компонентів в одному стилі для різних систем, відповідно до їх стильових настановлень.

2.2.5 Проєктування структури даних

На даному етапі було спроєктовано загальну структуру інформаційної моделі даних системи обліку обладнання. Структуру моделі даних було представлено у вигляді ER-діаграми (див. рис. 2.6), яка містить в собі опис таблиць в базі даних, полів в таблицях та їх властивості.

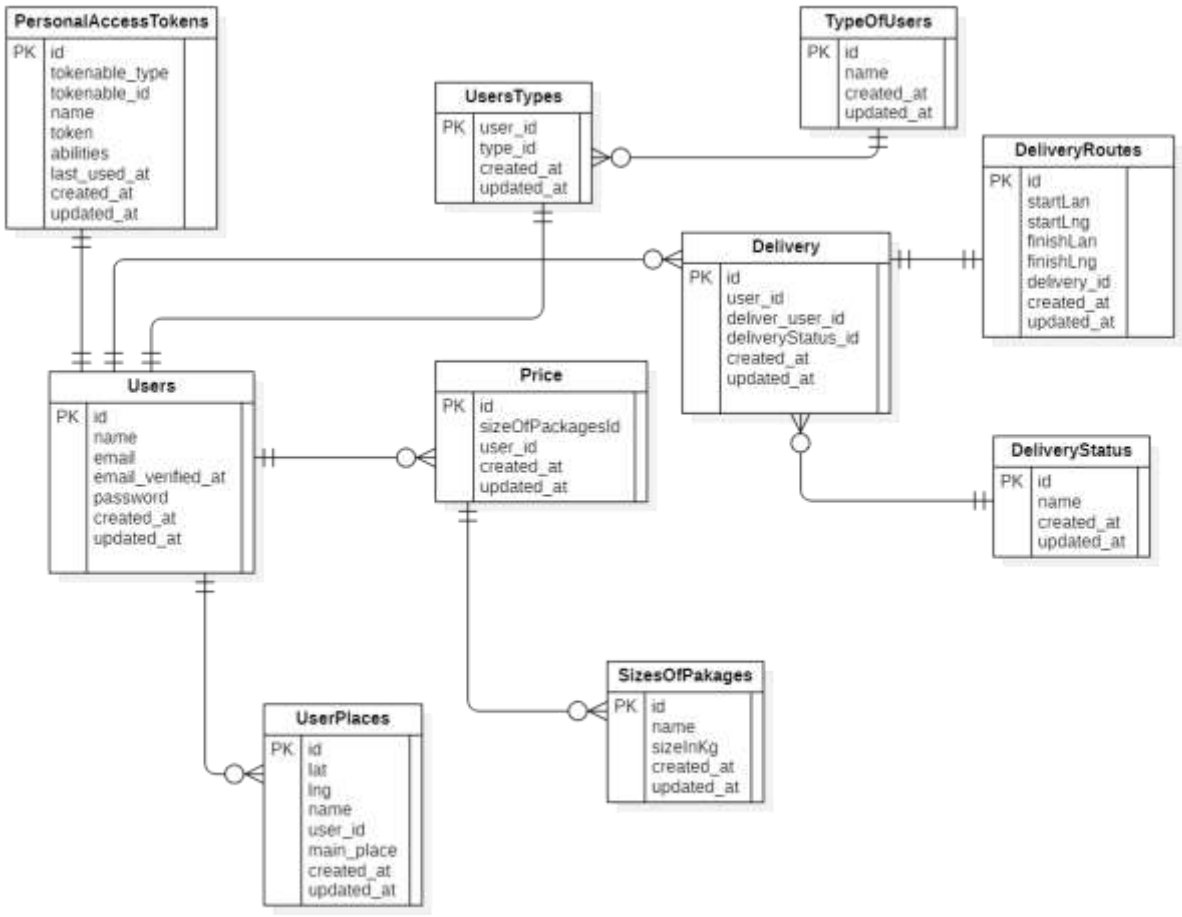

Рисунок 2.6 − ER-модель даних системи

Розглянемо детальніше основні таблиці інформаційної моделі системи.

У таблиці «users» («Користувачі») зберігаються дані про всіх користувачів, що існують в системі (див. табл. 2.1).

Таблиця 2.1 – Користувачі

Назва поля	Тип даних	Властивість	Опис
id	BIGINT(20) unsigned	Primary key (PK), AutoIncrement	Ідентифікатор користувача
name	text		Ім’я користувача
e-mail	text		E-mail користувача
email_verified_at	timestamp		Дата підтвердження паролю

password	text		Hash-пароль користувача
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Таблиця «TypeOfUsers» («Типи користувачі») містить дані про те, яку назву мають типу користувачів в даній системі(див. табл. 2.2).

Таблиця 2.2 – Типи користувачів

Назва поля	Тип даних	Властивість	Опис
id	Integer	Primary key (PK), AutoIncrement	Ідентифікатор обладнання
name	Text		Назва типу
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Таблиця «usersTypes» («Користувачі з типами») містить дані про користувачів яким вказано тип, та який саме цей тип.(див. табл. 2.3).

Таблиця 2.3 – Користувачі з типами

Назва поля	Тип даних	Властивість	Опис
user_id	integer	Primary key (PK), AutoIncrement	Ідентифікатор користувача
type_id	integer		Ідентифікатор типу
created_at	timestamp		Дата створення запису
updated_at	timestamp		Дата останнього оновлення

Таблиця «PersonalAccessToken» («Персональний токен доступу») містить дані про токени доступу, які генеруються при авторизації та використовуються під час роботи з системою (див. табл. 2.4).

Таблиця 2.4 – Персональний токен доступу

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор
tokenable_type	text		Модель Laravel для якої створено токен
tokenable_id	bigInt(20)		Ідентифікатор для моделі
name	text		Назва
token	text		Хешований токен користувача
abilities	text		Перелік можливостей які отримує користувач, з цим токеном
last_used_at	timestamp		Дата та час останнього використання токену
created_at	timestamp		Дата створення запису
updated_at	timestamp		Дата останнього оновлення запису

Таблиця «UserPlaces» («Місця користувачів») містить дані про місця які користувачі додали у власний список точок на мапі, а також про те, які з них вказано як головні у користувача(див. табл. 2.5).

Таблиця 2.5 – Місця користувачів

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор місця
lng	text		Географічна довжина
lat	text		Географічна широта
name	text		Назва місця
main_place	bool		Прапорець, що вказує чи є цей запис головним для користувача
created_at	timestamp		Дата створення запису
updated_at	timestamp		Дата оновлення запису

Таблиця «SizesOfPackages» («Розміри об’єктів доставки») містить типи можливих розмірів об’єктів для доставки(див. табл. 2.6).

Таблиця 2.6 – Розміри об’єктів доставки

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор розміру

name	text		Назва розміру
sizeInKg	integer		Кількість кілограм
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Таблиця «Price» («Ціна») містить дані про ціну за одиницю доставки вказану для різним типів розмірів користувачами – виконавцями(див. табл. 2.7).

Таблиця 2.7 – Ціна

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор
sizeOfPackages_id	integer		Ідентифікатор розміру об’єкта доставки
user_id	integer		Ідентифікатор користувача
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Таблиця «Delivery» («Доставка») містить дані про доставку, замовника, виконавця та поточний статус (див. табл. 2.8).

Таблиця 2.8 – Доставка

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор доставки

user_id	integer		Ідентифікатор користувача – замовника
delivery_user_id	text		Ідентифікатор користувача – виконавця
delivery_status_id	text		Ідентифікатор статусу доставки
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Таблиця «DeliveryRoutes» («Маршрути доставки») містить дані про точку отримання об’єкта доставки, та точку куди слід доставити його (див. табл. 2.9).

Таблиця 2.9 – Маршрути доставки

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор доставки
startLan	text		Географічна широта точки отримання
startLng	text		Географічна довжина точки отримання
finishLan	text		Географічна широта точки призначення
finishLng	text		Географічна довжина точки призначення

delivery_id	text		Ідентифікатор доставки
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

Таблиця «DeliveryStatus» («Статус замовлення») містить дані про типи запчастин які встановлені в обладнання або знаходяться на складі (див. табл. 2.10).

Таблиця 2.10 – Статус замовлення

Назва поля	Тип даних	Властивість	Опис
id	integer	Primary key (PK), AutoIncrement	Ідентифікатор статусу
name	text		Назва статусу
created_at	timestamp		Дата створення
updated_at	timestamp		Дата оновлення

3 РЕАЛІЗАЦІЯ

3.1 Опис технологій

3.1.1 Мобільний застосунок

В моїй системі мобільний додаток відповідає за зовнішній вигляд системи та взаємодію з користувачем. Дизайн повинен бути досить простим та зрозумілим, але в той же час сучасним та привабливим. Крім того мобільний застосунок повинен мати широкі можливості по керуванню різними частинами системи.

Зі стандартного набору віджетів, які надаж фреймворк Flutter, було створено більшу частину екранів застосунку. Але на деяких екранах було використано специфічні віджети. Наприклад для відображення карти будо використано віджети Google Maps.

Також зі сторонніх бібліотек які використовувались слід відзначити бібліотеки для керування станом застосунку, та бібліотеки для мережевої взаємодії.

Почнемо з бібліотек для керування станом. Фреймворк вводить два основних класи віджетів: віджети зі станом та віджети без стану.

Багато віджетів не мають змінюваного стану: вони не мають властивостей, які змінюються з часом (наприклад, іконка або мітка). Такі віджети підкласу *StatelessWidget*.

Однак, якщо унікальні характеристики віджета повинні змінюватися на основі взаємодії з користувачем або інших факторів, цей віджет є віджетом зі змінним станом. Наприклад, якщо віджет має лічильник, який збільшується, коли користувач натискає кнопку, то значення лічильника є станом для цього віджета. Коли це значення змінюється, віджет потрібно перебудувати, щоб оновити свою частину інтерфейсу. Ці віджети є підкласом *StatefulWidget*, і

(оскільки сам віджет є незмінним) вони зберігають змінюваний стан в окремому класі, який є підкласом *State*. *StatefulWidgets* не мають методу побудови; замість цього, їх користувальницький інтерфейс будується через їх об'єкт *State*.

Кожного разу, коли ви змінюєте об'єкт *State* (наприклад, збільшуючи лічильник), ви повинні викликати setState(), щоб повідомити фреймворк про необхідність оновити користувацький інтерфейс шляхом повторного виклику методу побудови *State*.

Наявність окремих об'єктів стану та віджетів дозволяє іншим віджетам поводитися як з віджетами без стану, так і з віджетами зі станом однаково, не турбуючись про втрату стану. Замість того, щоб утримувати дитину для збереження її стану, батьки можуть створити новий екземпляр дитини у будь-який час без втрати постійного стану дитини. Фреймворк виконує всю роботу з пошуку та повторного використання існуючих об'єктів стану, коли це доречно.

Але це підхід який можна доповнити та загорнути в обгортку з так званого ситаксичного цукру, для збільшення читабельності коду та зниження складності розуміння (хоча це суб’єктивно). Вирішити це питання покликана велика кількість бібліотек[18]. Оглянемо кілька з них:

- Provider – обгортка навколо InheritedWidget, щоб зробити їх простішими у використанні та більш придатними для повторного використання. Використовуючи провайдера замість ручного написання InheritedWidget, ви отримуєте спрощене виділення/розпорядження ресурсами, ліниве завантаження;
- Riverpod – ще один хороший вибір, схожий на Provider і є безпечним для компіляції та тестування. Riverpod не має залежності від Flutter SDK;
- setState – низькорівневий підхід для використання для специфічних віджетів, ефемерних станів;

- Redux – набір утиліт, що дозволяють легко використовувати Redux Store для побудови Flutter віджетів.
- BLoC – це система паттернів потоків управління станом для Flutter, рекомендована розробниками Google. Допомагає керувати станом та надавати доступ до даних з центрального місця у вашому проекті.

У даному проекті було вирішено використати бібліотеку Provider, через схожий принцип створення моделей даних до Laravel, а отже наскрізна зрозумілість коду зросте.

Для мережевої взаємодії було використано бібліотеку Dio.

Dio – потужний Http клієнт для мови Dart. Що підтримує перехоплення, глобальні конфігурації, запити форм, відміну запитів, завантаження файлів, таймаути, та багато чого іншого.

Але мабуть найбільшою перевагою чому було обрано саме цю бібліотеку, для цього проекту, це значно полегшене у порівнянні зі стандартними бібліотеками мережевої взаємодії, відловлювання помилок.

Також приємною особливістю було те, що Flutter надає велику бібліотеку векторних іконок, кожна з яких також реалізована через віджет Icon.

В якості середовища розробки використовувалась Android Studio, з плагіном для розробки на мові Dart та плагіном для розробки з використанням фреймворку Flutter.

3.1.2 Back-end частина системи

У якості backend – складової системи був обраний PHP фреймворк Laravel.

Даний фреймворк було обрано за його простоту у використанні. Вже після інсталяції отримаємо все необхідне для запуску простого backend серверу та веб сайту на ньому. Згодом можемо дуже просто реалізувати будь-який програмний інтерфейс, та шаблон проектування. Велика кількість модулів, що поставляються в комплекті, дозволяють нам:

- підключити базу данних;
- використовувати вбудовану бібліотеку для взвємодії базою даних через об’єктні моделі;
- створити веб сторінки за допомогою вбудованого шаблонізатора;
- створити маршрутизацію веб застосунку;
- повністю контролювати API запити;
- і т.д.

Зупинимось більш детально на API, що буде відповідати за передачу даних до клієнта. Після ініціалізації проекту ми вже маємо створену структуру файлів, та конфігів застосунку. Звісно для програмного інтерфейсу ми також маємо заготовку. Дуже просто додати власний шлях до маршрутизації API. А завдяки моделі MVC дуже просто додати логіку як відповідь на запит шляху від клієнта (див. рис. 3.1).

Рисунок 3.1 – Зразок API шляхів Laravel

Хоча Laravel може обслуговувати авторизацію користувача самостійно, але є пакет в екосистемі даного фреймворку який значно полегшує цю задачу.

Це Laravel Sanctum. Саме він буде відповідати за генерацію токену для взаємодії клієнта з користувачем.

JSON Web Token (JWT) – це відкритий стандарт для створення токенів доступу, заснований на форматі JSON, використовується при передачі даних для автентифікації в клієнт-серверних додатках [19] (див . рис. 3.2).

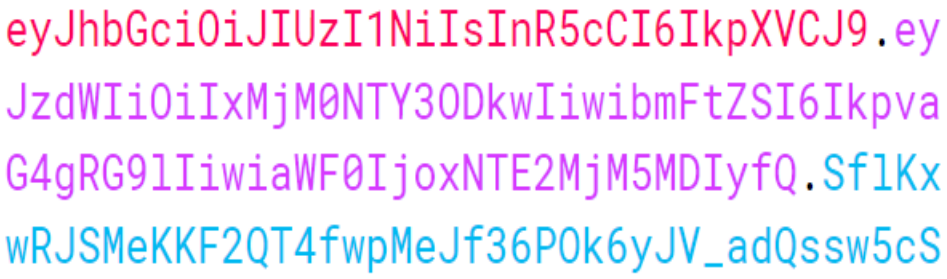

Рисунок 3.2 – Зразок JWT токену

Токени створюються серверами, що підписують секретні ключі та передають клієнтам, які в подальшому використовують даний токен для підтвердження своєї особистості.

Переваги JWT:

- оскільки вони не зберігають дані про сесію, операції з базою даних не потрібні для інформації користувачів;
- управління сесіями може бути виконано без внутрішніх файлів даних браузера (англ. cookie);
- один комутатор (англ. switch) може працювати на декількох серверах;
- база даних працює швидше, тому що ніякі операції з нею не виконуються на постійній основі.

Недоліки JWT:

- якщо ваш секретний ключ недостатньо сильний, їм можна легко маніпулювати;
- немає способу перевизначити сторону сервера, тому що JWT не зберігає сесію.

В парі з Laravel працює СКБД MySQL. Дану систему керування базами даних було обрано за такі переваги:

- підтримка баз даних необмеженого розміру;
- легку здатність до масштабування;
- наслідування;
- потужні та надійні механізми транзакцій та реплікації даних.

3.2 Реалізація системи

3.2.1 Реалізація процесу авторизації користувача

Інтерфейс користувача є однією з найважливіших складових мобільного додатку, адже саме від зручності та функціональності цієї частини системи залежить, чи буде користувач використовувати його.

Найперше, що було розроблено – система авторизації користувачів у системі (див. рис. 3.3).

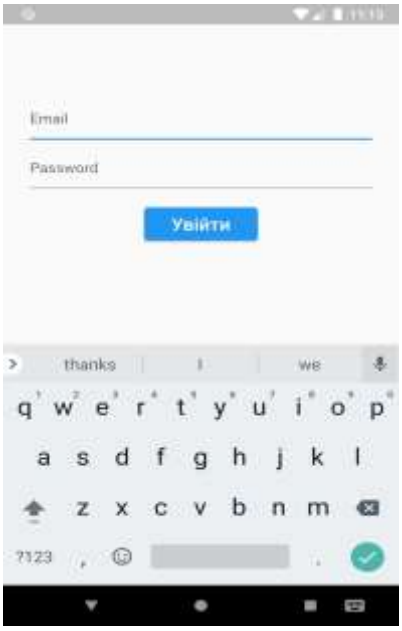

Рисунок 3.3 – Відображення екрану авторизації

Взаємодія з базою даних системи реалізована за допомогою моделей та контролерів у бекенд частині та моделей з сервісами у клієнтській частині (див. рис. 3.4 – 3.5).

Якщо доторкнулися сервісів та моделей. То доцільно буде сказати декілька слів про архітектуру. Взагалі Flutter не вимогливий до архітерктури. І кожен може роботи таку, яка подобається. В нашому випадку було обрано MVVM.

```
final storage = const FlutterSecureStorage();

bool get authenticated => _isLoggedIn;

User get user => _user;

void login({Map? creds}) async {
  var response = await dio.post('/sanctum/token', data: creds);
  String token = response.data.toString();
  tryToken(token: token);
}

void tryToken({required String token}) async {
  if (token == null) {
    return;
  } else {
    try {
      var response = await dio.get(
          '/user',
          options: Dio.Options(
              headers: {'Authorization': 'Bearer $token'}
          )
      );
      _isLoggedIn = true;
      _user = User.fromJson(response.data);
      _token = token;
      storeToken(token: token);
      print(_user);
      notifyListeners();
    } catch (e) {
      print(e);
    }
  }
}
```

Рисунок 3.4 – Сервіс до таблиці «Користувачі» в застосунку користувача

У бекенд частині контролер напряму взаємодіє з відповідною моделлю у якій вказано, до якої саме таблиці звертатися та які таблиці з нею пов’язані. Також модель надає нам доступ до функцій отримання пов’язаних з нею даних з інших таблиць. У клієнтському застосунку модель виконує такі ж самі функції (див. рис. 3.5).

```
class User {
  String name;
  String email;
  String avatar;

  User({required this.name, required this.email, required this.avatar});

  User.fromJson(Map<String, dynamic> json)
  : name = json['name'],
    email = json['email'],
    avatar = json['avatar'];

}
```

Рисунок 3.5 – Модель таблиці «Користувачі» у застосунку

Зв’язування таблиць між собою виконуємо під час створення таблиць БД за допомогою міграцій Laravel.

Структуру бази даних було сформовано за допомогою міграцій, одного з інструментів Laravel, а заповнюємо таблиці первинними даними було з використанням сідерів бази даних. Такий підхід є нативним у фреймворку та значно спрощує процес створення бази даних. А окрім цього збільшує рівень контролю за версіями структури, та за допомогою простих команд дозволяє їх змінювати, роблячи відкат на декілька кроків назад чи навпаки оновлюючи її.

3.2.2 Реалізація карти

Враховуючи функціональні вимоги до системи було розроблено інтерфейс для двох ролей користувачів: для користувача – виконавця та користувача – замовника. Основним компонентом, для кожного з користувачів,

який найчастіше зустрічається на екранах застосунку стала мапа з маркерами, та маршрутом доставки.

Як ми з'ясували раніше кожен з елементів користувацького інтерфейсу є віджетом одного з двох типів, то й віджет карти не став виключенням. В якості провайдера карт було обрано сервіс Google Maps. А саме:

- Maps SDK for Android – основна бібліотека / сервіс яку використовують під час інтеграції карт у застосунок на платформу Android;
- Places API – надає доступ до отримання деталей обраного на мапі місця;
- Directions API – надає доступ до маршрутів згенерованих між двома, або більше точками на мапі.

Враховуючи що Google надає нам доступ до сервісу карт та є власником мови Dart, а разом із нею фреймворку Flutter це надає величезні переваги. По-перше у нас є гарна підтримка як наборів для інтеграції так і вже створених бібліотек для цього. По-друге у нас великий обсяг документації, і відповіді на всі питання, що виникають в процесі розробки, можна отримати звідти. Дивлячись на це можна подумати, що процес інтеграції карт в застосунок дуже швидкий та простий. І це дійсно так. Все, що треба це отримати ключ, підключити бібліотеку до застосунку, і в код екрану вставити шаблонний код. І все. Простий застосунок, що відображає карту створено. Але в процесі доповнення функціоналу мапи є певні нюанси, на них і звернемо увагу.

Перш за все слід звернути увагу на те, що подальша взаємодія з Google картами відбувається за рахунок мережевих запитів, та отримання даних зі сторонніх серверів. Для спрощення цих запитів було написано два сервіси один відповідає за отримання даних про локацію (див. рис. 3.6), а другий виступає обгорткою для бібліотеки Dio (див. рис. 3.7).

```
import 'dio.dart';

class LocationService {

  final String key = '';

  Future<Map <String, dynamic>> getPlace(String input) async {
    var response = await dio.get('https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=$input&inputtype=textquery&fields

    var placeId = response.data['candidates'][0];

    return placeId;
  }
}
```

Рисунок 3.6 – Сервіс для отримання даних про локацію

```
import 'package:dio/dio.dart';

// or new Dio with a BaseOptions instance.
var options = BaseOptions(
  baseUrl: 'https://1f6c-212-111-202-6.eu.ngrok.io/api',
  headers: {
    Headers.acceptHeader: 'Application/Json'
  },
  connectTimeout: 5000,
  receiveTimeout: 3000,
);

Dio dio = Dio(options);
```

Рисунок 3.7 – Сервіс Dio

По – друге варто звернути увагу на процес генерації маршруту. Хоча на перший погляд здається, що це просто лінія. Але у Flutter все віджет, тому і лінія маршруту складається з багатьох маленьких віджетів (див. рис. 3.8).

```
void getPolyPoints() async {
  PolylinePoints polylinePoints = PolylinePoints();

  PolylineResult result =await polylinePoints.getRouteBetweenCoordinates(googleAPiKey,
      PointLatLng(sourceLocation.latitude, sourceLocation.longitude), PointLatLng(destinationLocation.latitude, destinationLocation.longitude));

  if(result.points.isNotEmpty) {
    result.points.forEach((PointLatLng point) {
      polylineCoordinates.add(LatLng(point.latitude, point.longitude));
    });
    setState(() {});
  }
}
```

Рисунок 3.8 – Функція для отримання маршруту

З точки зору графічного інтерфейсу на всі екрани з мапою було додано поле пошуку, що ініціює запит на сторонній ресурс. І для відображення результатів, було розроблено власний віджет.

3.2.3 Розробка власних віджетів

Як ми могли впевнитись все у фреймворку Flutter є об'єктами якихось класів. І наш віджет не став винятком, бо для його розробки спершу слід було створити клас, який наслідує один з варіантів. Або StatefullWidget, або StatelessWidget.

За ідеєю новий віджет має відображатись в нижній частині екрану, і виводити користувачу інформацію про місце яке він за допомогою маркеру вказав, як точка отримання об’єкту доставки, або як точку доставки. Віджет має мати власний дизайн (див. рис. 3.9), але щоб його можна було редагувати з батьківського екрану.

```
class _BottomSheetWidgetState extends State<BottomSheetWidget> {
  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(
        borderRadius: BorderRadius.only(topLeft: Radius.elliptical(15, 15), topRight: Radius.elliptical(15, 15)),
        color: Colors.black12
      ),
      padding: EdgeInsets.all(16),
      child: Column(
        mainAxisSize: MainAxisSize.min,
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          Text(
            widget.title,
            textAlign: TextAlign.center,
            style: const TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
          ),
          const SizedBox(height: 60),
          TextButton(
            child: Padding(
              padding: const EdgeInsets.symmetric(horizontal: 20),
              child: Text(widget.buttonTitle, style: TextStyle(fontSize: 20, color: Colors.white),),
            ),
            onPressed: () {
            },
            style: TextButton.styleFrom(
              backgroundColor: Colors.blue,
              elevation: 2,
            ),
          ),
        ],
      ),
    );
  }
}
```

Рисунок 3.9 – Самостійний дизайн у власному віджеті

Саме через це було обрано StatefullWidget. В полях класу нашого віджету визначено параметри які ми наслідуємо, та в конструкторі ініціалізовано (див. рис. 3.10).

```
class BottomSheetWidget extends StatefulWidget {
  final String title;
  final String buttonTitle;
  final VoidCallback onClickedConfirm;
  final VoidCallback onClickedClose;

  const BottomSheetWidget({
    required this.title,
    required this.buttonTitle,
    required this.onClickedConfirm,
    required this.onClickedClose,
    Key? key
  }) : super(key: key);

  @override
  State<BottomSheetWidget> createState() => _BottomSheetWidgetState();
}
```

Рисунок 3.10 – Конструктор з параметрами у власному віджеті

Таким чином ми отримали власний віджет, який можна використати в різних частинах застосунку викликаючи його в батьківському екрані та передаючи параметри (див. рис. 3.11).

```
bottomSheet: BottomSheetWidget(
  title: address,
  buttonTitle: 'Розпочати',
  onClickedConfirm: () {},
  onClickedClose: () {},
),
```

Рисунок 3.11 – Використання власного віджету та передача параметрів

Звісно це самий простий приклад створеного віджету. І під час проектування на етапі створення UML діаграми класів, уже було описано метод, як можна за допомогою інтерфейсів створити ціле сімейство таких

віджетів. Згодом їх модна буде огорнути в пакет, та розповсюджувати через пакетний менеджер фреймворку Flutter.

3.2.4 Розробка docker контейнеру для системи

Для більш простого запуску системи в роботу, її було обернуто в docker контейнер.

Docker – інструментарій для управління ізольованими Linux-контейнерами. Docker доповнює інструментарій LXC більш високим рівнем API, що дозволяє керувати контейнерами на рівні ізоляції окремих процесів. Зокрема, Docker дозволяє не переймаючись вмістом контейнера запускати довільні процеси в режимі ізоляції і потім переносити і клонувати сформовані для даних процесів контейнери на інші сервери, беручи на себе всю роботу зі створення, обслуговування і підтримки контейнерів.

Для коректної роботи docker були створені 2 файли Dockerfile, що збирає початкові залежності та переміщує код проєкту в директорії контейнеру, та docker-compose.yml (див. рис. 3.12) , що відповідає за роботу контейнеру з іншими контейнерами та хост системою.

Рисунок 3.12 – Вміст Dockerfile системи

3.2.5 Тестування відображення екранів застосунку Flutter

Щоб переконатись, що екрани застосунку відображаються вірно використав стандартні можливості Flutter.

Ми можемо використовувати такі варіанти тестів:

- юніт тест – для тестування бізнес логіки застосунку;
- тести віджетів – для перевірки відображення віджетів на екрані;
- тести інтеграції – призначені для тестування великої частини застосунку.

Для того, щоб протестувати відображення інтерфейсу додав до проєкту файли тестів віджетів, які порівнюють фактичне відображення екрану на запущеному пристрої або емуляторі, зі знімком екрану створеним програмно (див. рис. 3.13).

```
void main() {
  testWidgets('MyWidget has a title and message', (tester) async {
    await tester.pumpWidget(const MyWidget(title: 'T', message: 'M'));
    final titleFinder = find.text('T');
    final messageFinder = find.text('M');

    // Use the `findsOneWidget` matcher provided by flutter_test to verify
    // that the Text widgets appear exactly once in the widget tree.
    expect(titleFinder, findsOneWidget);
    expect(messageFinder, findsOneWidget);
  });
}
```

Рисунок 3.13 – Приклад файлу для перевірки відображення елементів на екрані

Тести у фреймворку Flutter дозволяють не лише визначити помилки у роботі або відображенні екранів застосунку, а й вказують на недоліки у використанні сторонніх бібліотек, так і на успішне проходження тестів (див. рис. 3.14 – 3.15).

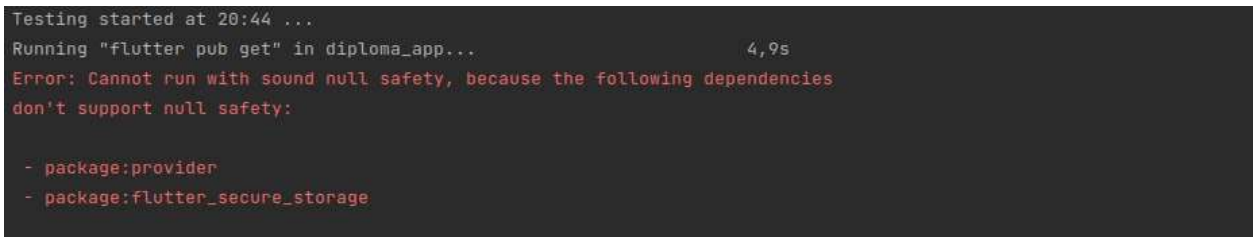

Рисунок 3.14 – Результат тесту екрану з помилкою при використанні сторонніх бібліотек

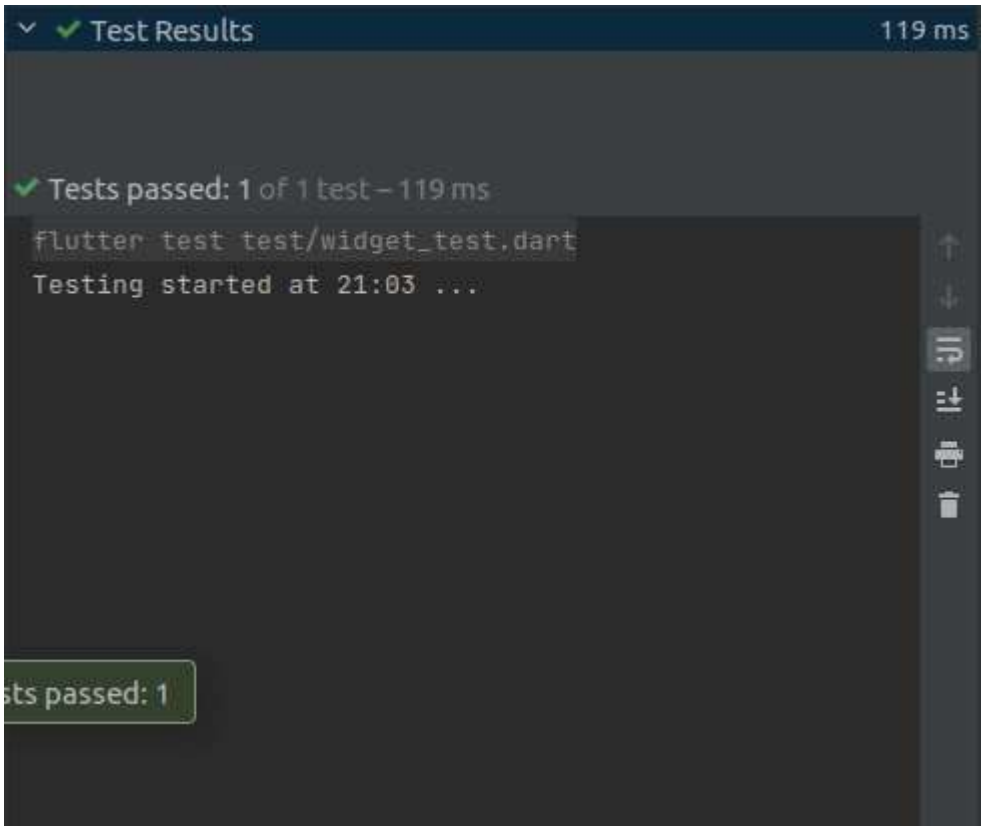

Рисунок 3.15 – Вивід повідомлення про успішне проходження тесту з використанням IDE Android Studio

ВИСНОВОК

В ході написання кваліфікаційної роботи було проведено огляд фреймворку Flutter, оглянуто принцип його роботи та його аналоги, сформульовано вимоги до системи з використанням аналізу вимог, спроєктовано з допомогою діаграм в нотації UML та побудовано архітектуру системи, реалізовано інтерфейс користувача та серверну частину проєкту.

Під час реалізації системи, отримано практичні та теоретичні навички роботи з різними бібліотеками, як бекенд частини, так і клієнтського застосунку. Отримано знання про різні архітектурні рішення в обох частинах систем. В рамках розробки застосунку поглиблено знання про складові фреймворку Flutter, через розробку власного віджету. Використано технології для спрощення підтримки, та розгортання системи на серверах.

В результаті роботи було розроблено сервіс замовлення та доставки товарів із застосуванням наступних технологій:

- СУБД MySQL 8.0;
- Laravel PHP Framework для реалізації back end;
- Flutter для реалізації клієнтського застосунку

Були використані принципи узгодження JSON API для комунікації між клієнтом та сервером.

Розроблена система на базі Laravel PHP Framework та Flutter реалізує наступні базові функції:

- додавання/ перегляд/ оновлення/ видалення замовлень на доставку;
- перегляд маршрутів для доставки та керування ним;
- взаємодії з мапою для додавання власних точок;
- авторизація за допомогою токенів;
- перегляд різних деталей відповідно до інформації в системі;
- захист від несанкціонованого доступу до інформації та системи.

Розроблена інформаційна система має наступні переваги:

- працює на різних платформах;
- використані сучасні технології, що робить розробку актуальною сьогодні, гнучкою в модифікації та легкою в супроводі;
- не потребує потужних клієнтських робочих станцій, але потребує досить потужний сервер;
- не прив’язана до робочого місця;
- підтримує велику кількість користувачів;
- централізація, яка полегшує адміністрування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Flutter for Android developers | Flutter. URL: https://docs.flutter.dev/get-started/flutter-for/android-devs
2. Pragmatic Flutter: Building Cross-Platform Mobile Apps for Android, iOS, Web & Desktop / Tyagi Priyanka / CRC Press. / 2021 / 354c
3. Flutter in Action / Eric Windmill / Manning Publications Co. / 2020 – 5c
4. Flutter in Action / Frank Zammetti /APress Media LLC. / 2019 – 104c
5. Beginning Flutter 3.0 with Dart: A Beginner to Pro. Learn how to build Advanced Flutter 3.0 Apps / Sinha Sanjib / Leanpub / 2022 / 1037c
6. Flutter architectural overview | Flutter. URL: https://docs.flutter.dev/resources/architectural-overview#layout-and-rendering
7. Cross-Platform UIs with Flutter / Edge Ryan, Miola Alberto / Packt Publishing / 2022 / 260c
8. Beginning Flutter with Dart: A Step by Step Guide for Beginners to Build a Basic Android and iOS Mobile Application/ Sinha Sanjib / Independently published / 2021 / 382c.
9. Flutter Cookbook: Over 100 proven techniques and solutions for app development with Flutter 2.2 and Dart / Alessandria S., Kayfitz B / Packt Publishing / 2021 / 639c.
10. React Native · Learn once, write anywhere URL: https://reactnative.dev/
11. Ionic Framework - The Cross-Platform App Development Leader URL: https://ionicframework.com/
12. What is Xamarin? – Xamarin URL: https://learn.microsoft.com/uk-ua/xamarin/get-started/what-is-xamarin

13. Flutter vs React Native vs Native: Deep Performance Comparison | inVerita URL: https://inveritasoft.com/blog/flutter-vs-react-native-vs-native-deep-performance-comparison
14. Laravel Application Development Blueprints / Kiliçdagi A., Yilmaz H.I. / Packt Publishing / 2013 / 260c.
15. Laravel Up & Running / Stauffer Matt / 2nd Edition. — O'Reilly Media / 2019 / 544c.
16. Інформаційні технології та моделювання бізнес-процесів / Томашевський О.М., Цегелик Г.Г., Вітер М.Б., Дубук В.І./ Навчальний посібник. - К.: Видавництво «Центр учбової літератури» / 2012 / 296с.
17. Safety, Reliability and Risk Analysis: Theory, Methods and Applications / Martorell S., Guedes Soares C., Barnett J. (eds.) / Proceedings of the European Safety and Reliability Conference, ESREL / 2008 / 3512c.
18. State in Flutter: A Guide to State Management in Flutter / Sinha Sanjib / Leanpub. / 2021 / 231c
19. JSON Web Tokens URL: https://auth0.com/docs/tokens/json-web-tokens