Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РОЗРОБКА КРОСПЛАТФОРМЕНОГО ДОДАТКУ ДЛЯ БРОНЮВАННЯ РЕСТОРАНІВ ВИКОРИСТОВУЮЧИ FLUTTER SDK

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
старший викладач
Борозенний С.О.
“____” _____ 2021 р.

Виконав студент ФІ-4
Ніверовський М.М.
“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ
Завідувач кафедри мультимедійних систем
канд. фіз-мат. наук, доц. _____ Жежерун О.П.
(підпис)
«___» _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу
студенту Ніверовському Микиті Миколайовичу
Факультету інформатики 4 р.н. бакалаврської програми
ТЕМА: Розробка кросплатформеного додатку для бронювання ресторанів використовуючи Flutter SDK

Зміст текстової частини до курсової роботи:

- Індивідуальне завдання
- Вступ
- Огляд теоретичного матеріалу і здійснення дослідження
- Висновки
- Список літератури
- Додатки (за необхідністю)

Дата видачі «___» _____ 2020 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: **Розробка кросплатформеного додатку для бронювання ресторанів використовуючи Flutter SDK**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	15.10.2020	
2.	Огляд теоретичних матеріалів за темою роботи	21.01.2021	
3.	Написання першого розділу	22.02.2021	
4.	Написання другого розділу	05.03.2021	
5.	Написання практичної частини	12.03.201	
7.	Написання третьої частини	02.04.2021	
8.	Написання висновків та корегування роботи	04.04.2021	
9.	Створення презентації та написання доповіді.	05.04.2021	
10.	Здача курсової	12.04.2021	

ЗМІСТ

Анотація

Під час виконання даної роботи була досліджена платформа для кросплатформної розробки Flutter, мова програмування Dart. Була розглянута декларативна парадигма розробки. Було розроблено мобільний застосунок під Android та IOS для бронювання столиків у ресторані та Web-версія для адміністрування та керування цими ресторанами та бронюваннями. Крім того, були розглянути додаткові бібліотеки для спрощення та вдосконалення розробки додатків за допомогою Flutter. Також для збереження даних було використано хмарні послуги на платформі Firebase.

Використані скорочення

Кросплатформений додаток - додаток для декількох операційних систем з єдиним програмним кодом.

Flutter - платформа для розробки застосувань для Android, IOS, MacOS, Windows, Linux, Web та інших систем з єдиним програмним кодом

Dart - багатопарадигмальна мова програмування розроблена компанією Google.

MacOS - операційна система розроблена компанією Apple, яку вона використовує у своїх комп'ютерах Macintosh.

SDK - (software development kit) - набір із засобів для розробки, таких, як утиліти, документації.

Фреймворк - програмна платформа, яка визначає каркас програмного застосунку, та додає багато необхідних програмних елементів до застосунку.

AOT - (ahead-of-time) компіляція високорівневої мови програмування у низькорівневу до початку виконання роботи. Це дозволяє підвищити швидкість під час роботи.

JIT - (just-in-time) компіляція, яка виконується прямо під час роботи застосунку.

CLI - (command line interface) інтерфейс для керування застосунком через командний рядок.

ВСТУП

Постановка задачі.

Дослідити мову програмування Dart, платформу для кросплатформеної розробки Flutter. Реалізувати систему для бронювання столиків у ресторанів за допомогою Flutter SDK з єдиним програмним кодом.

Актуальність дослідження.

Згідно із даними німецької компанії statista, яка спеціалізується на зборі дани, кількість смартфонів, які використовуються, сягає 3.6 мільярдів штук у 2020 році. Згідно з графіку(див. Додаток А) у 2023 кількість девайсів повинна бути біля 4.3 мільярдів штук[1]. І це зрозуміло, зараз майже у кожної людини є власний мобільний девайс, у деяких людей - навіть декілька.

Тому зараз розробка мобільних додатків, актуальна, як ніколи раніше, і з часом, ця актуальність тільки буде зростати. Але розробка нативних додатків, тобто окремо під IOS та Android доволі витратна. Тому що компанії необхідно мати дві окремі команди, як мінімум одну з IOS, а іншу з Android технологій. Для маленьких компаній або стартапів це - непід’ємні витрати.

Для таких компаній або, щоб просто прискорити розробку мобільних додатків, є вихід. Це розробка кросплатформеного програмного застосунку. Одним з таких як раз і є Flutter. Про який і піде мова у цій курсовій роботі.

Структура роботи.

Робота складається зі вступу, трьох розділів, списку літератури та доповнень.

У першому розділі подана загальна характеристика про мову програмування Dart. Її історія, особливості та інше.

У другому розділі детально розглянути деякі аспекти фреймворку Flutter: його історія, декларативний стиль програмування, який в ньому використовується, деякі віджети, керування станами, дерева віджетів та інші, переваги та недоліки, а також, чому саме Dart використовується у Flutter.

У третьому розділі поданий опис практичної роботи, який був виконаний, а саме: які саме засоби були обрані, яким середовищем користувались, яка була обрана архітектура проекту та процес виконання роботи.

РОЗДІЛ 1. Мова програмування Dart.

1.1 Історія

У жовтні 2011 року на презентації GOTO від Google була представлена мова програмування Dart.

Автори мови: Lars Bak та Kasper Lund стверджували, що створили її на заміну JavaScript [2].

Розробники прийняли Dart зі змішаними почуттями, тому що на їх думку мережа б розбилась на декілька таборів. Було відомо, що компанія Google збирається інтегрувати Dart VM у Google Chrome, але потім у березні 2015 року Google заявила, що Dart буде компілюватися у JavaScript[3].

У серпні 2018 року було випущено Dart 2, у якій з'явилась надійна система типізації[4]. Приклад:

```
printListDouble(List<double> list) => print(list);

main() {
 final doubles = [];
 doubles.add('2.1');
 doubles.add(0.2);
 printListDouble(doubles);
}
```

Тут з'являється помилка у статичному аналізаторі при виклику функції `printListDouble`:

```
The argument type 'List<dynamic>' can't be assigned to the parameter type 'List<double>'
```

Тому що змінна `doubles` має тип `'List<dynamic>`. А її намагаються привезти до `'List<double>`, що неможливо.

Раніше така помилка з'являлась вже на етапі роботи.

У листопаді 2019 році представляють Dart 2.6, у якому з'являється dart2native[5]. Це розширення дозволяє компілювати додаток на Dart у

виконуваний файл для Linux, macOS, Windows. Це дозволило не встановлювати Dart SDK до системи.

У березні 2021 року вийшов Dart 2.12, у якому додали null safety технологію[6]. Це дозволяє уникати багато помилок.

Раніше:

```
int n;

addOneAndPrint(int n){
 print(n+1);
}

main(){
 addOneAndPrint(n);
}
```

Ми отримаємо помилку, тому що змінна `n` не ініціалізована, а у прикладі додається 1 до `null`.

Стало:

```
int n = 0;
addOneAndPrint(int n){
 print(n+1);
}
main(){
 addOneAndPrint(n);
}
// або
int? n;
addOneAndPrint(int n){
 print(n+1);
}
main(){
 if (n != null) {
   addOneAndPrint(n);
 }
}
```

Це оновлення зменшить кількість помилок у рази.

1.2 Особливості мови

Dart - “С подібна” мова, дуже схожа на TypeScript та JavaScript, тому багатьом розробникам не буде дуже складно зрозуміти її.

Приклад “Hello World” програми:

```
main() {
 print('Hello, World!');
}
```

Розглянемо декілька цікавих особливостей:

- Mixins

Міксини - це додаткові елементи у мові Dart, які дозволяють використовувати один і тий самий код повторно у ієрархії класів.

```
mixin Driver on Person {
 bool get canDriveMoto => false;

 void drive(){
   print("vroom...");
 }
}

class Student extends Person with Driver {
 @override
 bool get canDriveMoto => true;
}

class Teacher extends Person with Driver {}
```

- Extensions

За допомогою цієї особливості ми можемо розширяти класи, які вже готові для використання не змінюючи їх.

```
extension DateFormat on DateTime {
 String toCustomFormat() {
   return "$day.$month.$year";
```

```
};
}
```

- Конструктори класів

1. Ключове слово new не обов’язкове для виклику конструктора.

```
var date = new Date(2021, 1, 1);
var date = Date(2021, 1, 1);
```

2. У Dart є дуже цікавий синтаксичний цукор для призначення аргументів конструктора:

```
class Date {
 int year;
 int month;
 int day;

 Date(this.year, this.month, this.day);
}
```

3. Іменні конструктори

```
 Date.origin(): year = 0, month = 0, day = 0;
```

4. Конструктор фабрика, який реалізує паттерн фабрика

```
factory Date.fromJson(Map<String, Object> json) {
  return Date(json['year']!, json['month']!, json['day']!);
}
```

- Каскади

Зручна особливість, яка дозволяє викликати функції класу послідовно.

```
extension DateOperations on Date {
 void addYear(int v){
  year = year + v;
 }
}
var date1 = Date(2021, 1, 1);
var date = date1..addYear(2)..addYear(5);
```

- Асинхронність

Для асинхронного програмування у Dart є класс Future та два ключових слова: async та await.

```
Future<void> makeOrder() async {
 await Future.delayed(Duration(seconds: 10),
  () {
    // make Order
  }
);
}

Future<void> main() async {
 print("Make order");
 await makeOrder();
}
```

За допомогою async та await слів асинхронний код стає зрозумілим, ніби він є синхронним.

- Isolates

Ізоляти - це альтернатива потокам у інших мовах програмування. Завдяки ізолятам на Dart можна побудувати багатопоточний додаток.

- Ahead-Of-Time/Just-In-Time

Dart VM пропонує just-in-time (JIT) компілятор з hot-reload можливістю, тобто, дозволяє перезавантажувати додаток залишаючи поточний стан.

Якщо опублікувати додаток у App Store або розгорнути серверну частину, то компілятор DART AOT дає можливість Ahead-Of-Time компіляції на процессорах ARM або X64. Це значить, що AOT-скомпільований додаток запускається з постійною швидкістю та за дуже короткий проміжок часу.

1.3 Робота з бібліотеками та пакетами

У Dart є багато вбудованих бібліотек для роботи з колекціями, асинхронністю, математичними функціями, ізолятами, файлами, сокетами, html та багато інших.

Спільнота Dart використовує сайт pub.dev для поширення бібліотек та утиліт, та pub manager для керування бібліотеками у проектах.

Для приєднання бібліотеки до проекта необхідно створити файл pubspec.yaml у папці з проектом.

```
name: my_app_name
dependencies:
 package_name: ^1.0.0
```

Усі бібліотеки будуть завантажені з сайту pub.dev.

РОЗДІЛ 2. Flutter.

2.1 Історія

У 2015 році Google анонсувала Flutter SDK на мові програмування Dart. Тоді вона не була у відкритому доступі та не була кросплатформеною, розробка велась тільки під Android [8].

У 2017 році компанія випустила альфа версію, яка вже була публічною.

На конференції Google I/O 2017 було анонсовано вже кросплатформену версію SDK, і компанія вже рекомендувала всім використовувати Flutter для мобільної розробки.

У 2018 році з'явився Flutter 1.0. Це було дуже значуще оновлення, тому що Flutter перейшов у stable стан, тобто це був сигнал, що Flutter готовий для розробки у продакшн. Також була додана підтримка Google Maps, WebView на мобільні пристрої, Flare для створення векторних складних анімацій та інше. Й напевне найбільш значуще та несподіване для спільноти оновлення - це анонсування Flutter for Web. Тим самими було заявлено, що Flutter - це не тільки про мобільну розробку[9].

Вже у 2019 було додано у публічний альфа доступ можливість розробки на Flutter під Web та Desktop: Windows, Linux та MacOS.

Наступне велике оновлення для Flutter було вже у березні 2021 року на презентації Flutter Engage, вийшов Flutter 2. Можливість розробки веб додатків перейшло у stable стан, вийшов Dart 2.12 з null safety технологією, бета - версія для розробки десктопних застосувань, додані нові віджети, додан Google Mobile Ads та багато інших цікавинок. Ще цікавим моментом, є те, що компанія Canonical оголосила у своєму Twitter аккаунті, що Flutter став інструментом за замовчуванням для розробки застосунків на операційну систему Ubuntu[7][10].

2.2 Декларативный стиль

Існує дві альтернативні парадигми програмування імперативна та декларативна.

Імперативне програмування - це парадигма, яка заснована на складанні алгоритму дій, які змінюють стан програми, тобто вона ніби відповідає на питання: “Як саме це зробити?”. Приклади: C ++, Java.

Декларативне програмування - це парадигма, в якій описується бажаний результат і не описується алгоритм дій, тобто ця парадигма ніби відповідає на питання: “Що саме зробити?”. Приклад: HTML, SQL.

Фреймворки для розробки користувацького інтерфейсу використовують імперативну парадигму програмування, будь-то для Win32, web, Android, iOS та інші. І взагалі, імперативний стиль програмування, напевне найбільше усім знайомий.

Приклад на псевдокоді:

Імперативний стиль:

```
a.setSize(25)
a.clearList()
var b = new ViewB(...)
a.add(b)
```

Декларативний стиль:

```
return ViewA(
 size: 25,
 list: ViewB(...),
)
```

Тобто у декларативному стилі більшість переходів та іншу роботу робить за розробника фреймворк.

Flutter - це декларативний фреймворк. Це значить, що Flutter будує користувацький інтерфейс самостійно, розробник керує тільки станами.

2.3 Віджети

Усі елементи користувацького інтерфейсу у Flutter - це віджети. Розробники Flutter кажуть, що під час розробки надихались сучасним веб/мобільним фреймворком React[11]. Головна ідея у тому, що ви будуєте користувацький інтерфейс тільки за допомогою віджетів. Віджети описують увесь користувацький інтерфейс, як має виглядати кожний елемент, як повинна рухатись анімація. Коли віджет змінює свій стан, викликається функція build() цього віджета.

У Flutter існує два типи віджетів:

- незмінні віджети(Stateless widget)
- віджети, які можуть змінюватися(Stateful widget).

Головна робота віджета - це виконати функцію build(), яка описує віджет за допомогою інших віджетів. Фреймворк створює ці віджети доти, поки не дійде до об’єктів, які вже відповідають за рендеринг(малювання) на екрані.

Основні види віджетів:

- для розмітки

Row - дозволяє розмістити віджети по горизонталі

Column - дозволяє розмістити віджети по вертикалі

Center - розташовує віджет по центру

і тощо

- для керування виглядом

Padding - можна додати відстань з кожної сторони

Theme - для налаштування теми для усіх дочірніх віджетів

і тощо

- для анімації

AnimatedContainer - можна анімовано змінити будь які властивості контейнера

AnimatedSize - анімована зміна розміру

і тощо

Ще є безліч інших видів віджетів.

2.4 Керування станами

Stateless віджети незмінні і всі їх властивості теж незмінні. Такі віджети складаються із одного класу: StatelessWidget.

А ось Stateful віджети мають можливість змінювати свої властивості і складаються вже з двох класів: StatefulWidget, який створює екземпляр класу State. І що цікаво, що сам StatefulWidget клас незмінний, як і StatelessWidget, а зміни відбуваються у класі State протягом усього життєвого циклу віджета.

Розберемо на прикладі, як відбуваються зміни:

```
class IncrementorPage extends StatefulWidget {
 IncrementorPage({Key? key}) : super(key: key);

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

class _IncrementorPageState extends State<IncrementorPage> {
 int _increment = 0;
 @override
 Widget build(BuildContext context) {
  return Scaffold(
   body: Center(
    child: Text("$_incrementor"),
   ),
   floatingActionButton: FloatingActionButton(
```

```
      onPressed: () {
        setState(() {
          _increment += 1;
        });
      },
    ),
  );
}
}
```

Для початку розберемо структуру класів. Клас **IncrementorPage** наслідується від класу **StatefulWidget** та визначає функцію **createState**(), в якому створюється екземпляр класу **_IncrementorPageState**, який наслідується від класу **State**.

У класі **_IncrementorPageState** визначений цілочисельний атрибут _increment та перевизначена функція **build**.

Функція **build** повертає дерево віджетів, у якому спочатку йде віджет Scaffold. За допомогою цього віджета можна створити стандартну сторінку у стилі Material Design.

У Scaffold є параметр body, у якому ми створюємо віджет Center, який розташовує свій дочірній віджет у центрі. У цього віджета у параметрі child (це стандартна назва параметру, яка приймає дочірній віджет) передається віджет Text, який потрібен для відображення тексту, де й відображається атрибут _increment класу **_IncrementorPageState**.

У Scaffold також є другий параметр floatingActionButton, який приймає віджет. Зазвичай туди передають віджет FloatingActionButton, але можна й будь-який інший. Віджет FloatingActionButton створює кнопку. У цього віджета є параметр onPressed, у який передається функція, яка виконується, коли натискається кнопка.

У цій функції виконується функція setState, яка визначена у класі **State**. Ця функція приймає єдиний параметр, а саме - іншу функцію. Функцію setState викликають тоді, коли потрібно сповістити фреймворк, що

потрібно перемалювати цей віджет. А функція, яка передається у setState, спочатку виконується, а вже потім перемальовується віджет.

У нашому випадку передається функція, яка просто збільшує атрибут _increment на одиницю.

У “Додатку Б” можна подивитись, як буде виглядати цей віджет на мобільному пристрої. А у “Додатку В” вже перерисований віджет, після натискання кнопки.

Для більш просунутого керування станами існують додаткові бібліотеки: Provider, BLoC, Redux, MobX, GetX та інші.

2.5 Дерева віджетів, елементів та відтворення

З приклада, який було розглянуто у попередньому підрозділі можно побачити, що розробник створює дерево віджетів. У Flutter фреймворка, окрім дерева віджетів, існує також ще два дерева: дерево елементів (Element tree) та дерево відтворення(Render tree).

Розробник створює та контролює дерево віджетів, а дерева елементів та відтворення керує фреймворк Flutter у середині, але це робиться на основі дерева віджетів. Дерево віджетів, яке створюється на основі кода та яке будується фреймворком після виклика функції build, - це просто набір налаштувань, які Flutter враховує для відображення. Але це не є пряме відображення, це лише каже Flutter, що потрібно відобразити на екрані. Дерево елементів - дуже важлива річ, воно будується на основі дерева віджетів та зв’язує його з деревом відтворення.

Дерево віджетів постійно перебудовується, воно перебудовується кожен раз, коли викликається функція build. Дерево елементів працює по-іншому, воно не перебудовується кожний раз, коли викликається build. Воно з’єднує віджети, тобто ті конфігурації, які пише розробник, з елементами відтворення, які є частиною дерева відтворення. А дерево відтворення - це

кінцева точка і це вже те, що бачить кінцевий користувач на екрані. Дерево відтворення теж перебудовується не дуже часто.

Кожному віджету у дереві віджетів відповідає елемент у дереві елементів. Елемент врешті-реши - це просто об’єкт який знаходиться під управлінням фреймворку та має посилання на віджет. Елемент сам по собі не має внутрішніх налаштувань, це просто вказівник на віджет, який має конфігурації. Наприклад, якщо ви маєте віджет Container з певним кольором, елемент Container буде відносно пустий елемент з вказівником на віджет, який вже маю інформацію про колір.

Також є цікава особливість, яка стосується Stateful віджетів. У прикладі, який було розглянуто раніше, було два класи: StatefulWidget та State. У дереві елементів створюється тільки один елемент з віджету, а вже у цього елемента буде посилання на об’єкт класу State, так само і у віджета буде посилання на цей же об’єкт. Важливо зазначити, що об’єкт класу State є незалежним, і не належить ніякому дереву. Здається, що елементи це просто об’єкти які мають посилання на віджет. Але це не зовсім так, вони також мають посилання на Rendered Box, об’єкт, який відповідає за відображення віджету на екрані.

2.6 Переваги та недоліки

- Переваги
    - Написавши код на Flutter один раз, його можна переносити на різні системи, вносячи лише невелички правки. Авжеж кросплатформені фреймворки так і повинні працювати, але на Flutter це робити легше, де б не було.
    - Гаряче перезавантаження(Hot Reload). Ця особливість дозволяє вносити зміни при розробці, не перезбираючи проект, та бачити зміни на емуляторі або підключеному девайсі у реальному часі. Це економить дійсно багато часу при розробці.

- Швидка розробка. В конкурентній среді, коли швидкий вихід на ринок може дати можливість заробити більше популярності та грошей, швидкість розробки є одним із вирішальних моментів. На Flutter велика швидкість не тільки через, те що він кросплатформенний, але й тому що на ньому дійсно можна писати швидше за інші фреймворки та нейтів розробку[12].
- Велика кількість готових віджетів. У Flutter дуже багато віджетів, з якими можна легко зробити щось потрібне. Також можна окремо виділити віджети для роботи з анімацією. Напевне, робота саме з анімацією є найлегшою саме у Flutter, порівняно з іншими фреймворками.
- Кожен сезон команда розробки Flutter проводить опитування серед розробників, де питає відгуків та порад. Це дає можливість простим розробникам впливати на майбутнє фреймворку.
- Доступний для веб та десктоп розробки. Більшість інших конкурентів не мають такої можливості.
- Елементи користувацького інтерфейсу будуть однакові на усіх платформах, тому що Flutter відмальовує все самостійно, а не використовує візуальні елементи кожної системи.

- Недоліки
  - Головним недоліком зараз є те, що Flutter досить молодий фреймворк і він знаходиться у стадії бурхливої розробки. Через це в ньому досить багато багів.
  - Мова програмування. Flutter використовує мову програмування Dart і вона не є популярною, через це її потрібно вчити. Хоча вона є досить простою та схожею на більшість сучасних мов.

- Застосунки на Flutter важать більше, ніж аналоги та нативні.
- Через те, що Flutter постійно змінюється, іноді потрібно досить багато змінювати у застосунках, які довго не оновлювали.

2.7 Чому Dart?

Багато розробників, які знайомляться з Flutter запитують, чому компанію Google використала саме мову програмування Dart для розробки на Flutter. Для цього є декілька причин:

1. Комбінація Ahead-Of-Time та Just-In-Time компіляцій.

JIT компіляція використовується під час розробки, що дозволяє дуже сильно прискорити час розробки. А AOT компіляція використовується вже під час запуску додатків на пристроях у продакшені.

2. Непотрібні окремі файли для верстки користувацьких інтерфейсів.

Dart має декларативну та програмовану верстку, яку легко читати та візуалізувати.

3. Відсутність додаткових шарів для обробки, таких як JavaScript bridge.

Програма без проблем працює на пристрої користувача, тому що Dart компілює в рідний код пристрою.

РОЗДІЛ 3. Реалізація додатку для бронювання столиків

3.1 Технічне завдання

Технічне завдання полягає у створенні мобільного застосунку з можливістю бронювання столиків та веб застосунку для створення ресторанів та бронювання столиків для них. Все це потрібно виконати використовуючи Flutter SDK.

3.2 Використані засоби

Під час виконання практичної роботи був розроблен мобільний додаток для бронювання столиків та веб додаток для адміністрацій ресторанів, де вони можуть створити сторінку ресторану, а потім - слідкувати за бронюваннями від клієнтів. Для реалізації цієї задачі, окрім стандартних бібліотек, які доступні у Flutter SDK, були використані наступні:

- GetX - легка бібліотека, яка вирішує дуже багато проблем у розробці мобільних застосунків, такі як керування станами, ін’єкція залежностей та навігація у застосунку. Також у цій бібліотеці присутній механізм перекладу на інші мови, який також був використаний. Для керування станами використовується реактивний стиль[13].
- flutter_neumorphic - набір віджетів у стилі неоморфізму, який знаходиться у трендах 2020-2021 роках[14].
- flutter_svg - бібліотека для відображення векторних зображень[20].
- equatable - бібліотека, яка перевизначає оператор дорівнювання та хеш-код для класа даних.
- json_serializable - бібліотека для генерацій функцій для класа даних для обробки JSON

Окрім Flutter фреймворку були використані хмарні технології у вигляді Firebase.

- Firebase Authentication[15] - хмарний сервіс, який дозволяє зручно слідкувати за вашими користувачами: авторизовувати, реєструвати, востановлювати пароль тощо. Також він має зручні SDK для багатьох популярних мов програмування. Окрім того є можливість легкого під'єднання авторизацій за допомогою соціальних мереж. Firebase Auth має зручний SDK, який інтегрований з іншими сервісами Firebase. Також його можна під'єднати до власного серверного застосунка, тому що він підтримує OAuth 2.0 та OpenID Connect[15].
- Firestore Database[16] - хмарна NoSQL база даних, у якій дані зберігаються у колекціях, в яких зберігаються документи з полями. Ці поля можуть бути різних типів даних: числа, булеві значення, масиви, об'єкти, строки та посилання. Доступ до даних з бази можна отримати у реальному часі, тобто якщо хтось вносить зміни у базу, то інші користувачи можуть одразу побачити зміни. Також на мобільних пристроях є підтримка збереження даних без мережі, які синхронізуються з базою одразу, як з'явиться зв'язок. У Firestore дуже зручний SDK, за допомогою якого можна писати гнучкі запроси до бази, сортувати, фільтрувати дані та обробляти їх. Також є зручний інтерфейс для управління доступом до бази, в якому є інтеграція Firebase Authentication.
- Firebase Storage[17] - сервіс для збереження файлів. Він дозволяє створювати різні корзини, в яких можна створювати директорії та завантажувати в них та з них файли. У цьому сервісі також є зручний інтерфейс для управління доступа до бази, в якому є інтеграція з Firebase Authentication. Окрім того у Firebase Storage є інтеграція з Google Cloud Storage APIs, за допомогою якого можна робити різні операції з файлами, наприклад, фільтрація зображень або перекодування відео.

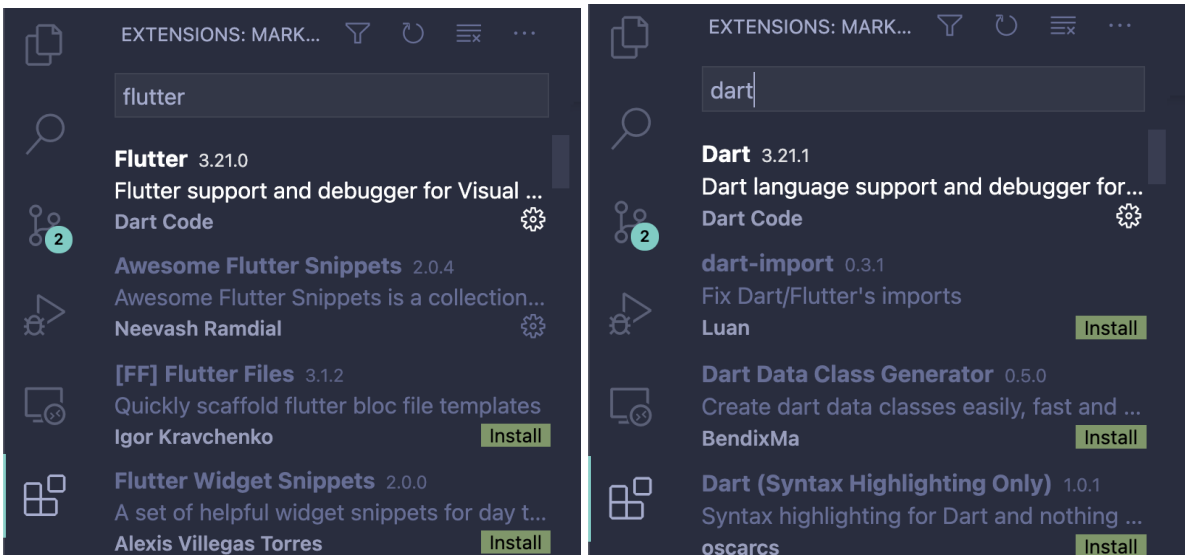
- Firebase Analytics[18] - зручний сервіс для відстеження дій користувачів. Дуже потрібний та зручний для комерційних цілей.
- Firebase Crashlytics[19] - дуже класний сервіс, який дозволяє слідкувати за аварійним завершенням застосунків на Android та iOS.

3.3 Середовище розробки

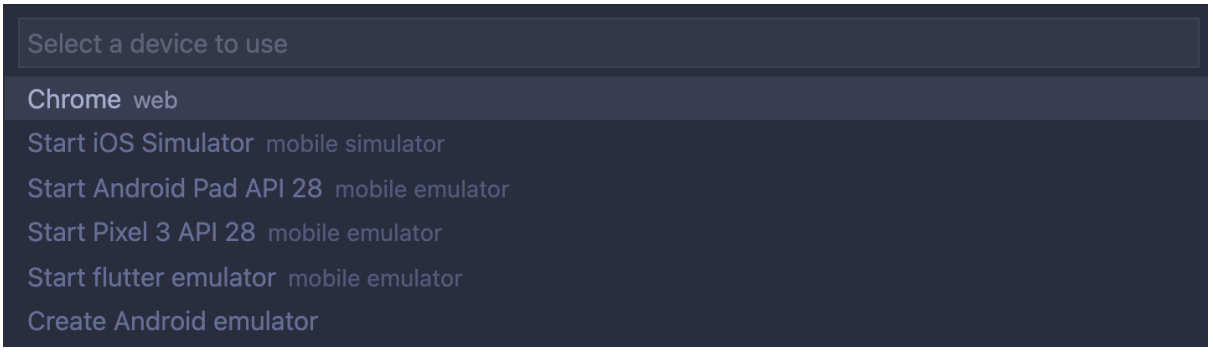
Для того, щоб розробляти застосунки на Flutter, пропонується два середовища: Android Studio і Visual Studio Code. Для зручної роботи потрібно в обох встановити плагіни для Dart та Flutter.

Android Studio вважається більш потужним засобом, проте він потребує більших обчислювальних потужностей на комп'ютері. Тому був обран Visual Studio Code, тому що він більш легкий і задовольняє усі потреби.

Visual Studio Code - це легке, але водночас потужне середовище розробки від компанії Microsoft. У VS Code є зручний інструментарій для відладки програмного застосунку, для роботи з системою керування версій Git та багато іншого. Одним із переваг у цій IDE є можливість встановлення розширень під різні задачі, зокрема для того, щоб зручно працювати з Flutter потрібно встановити розширення з бібліотеки розширень прямо у середовищі.



Ці розширення дозволяють створювати проекти, запускати їх на різних пристроях для тестування, обираючи їх з меню.



Є додаткова можливість запуску Dart DevTools, це додаткова утиліта, яка допомагає дебажити застосунки на Dart та зокрема во Flutter. За її допомогою можна зробити наступне[23]:

- Слідкувати за розміткою користувацького інтерфейсу та за станами у Flutter
- Діагностувати проблеми у продуктивності
- Дивитись використання пам’яті, процесорних та мережевих потужностей.
- Аналізувати код та розмір застосунка.
- Слідкувати за деревом віджетів у реальному часі.

Скріншот DevTools дивитись у Додатку Г.

Додатково було використано середовище розробки Xcode. Для того, щоб налаштувати розробку під iOS, потрібно використовувати тільки Xcode. І взагалі, якщо необхідно розробити застосунок під iOS, то без macOS, не обійтись.

3.4 Архітектура проекту

При розробці була використана класична трирівнева архітектура:

- перший рівень - доступ до даних
- другий рівень - бізнес-логіки

- третій рівень - представлення.

На рівні доступу до даних був використаний популярний паттерн “Репозіторій”. Він інкапсулює великі запити до бази даних. До того ж можна легко змінити джерело даних, перевизначивши методи репозиторію у іншому класі.

На рівні бізнес логіки використовувались класи наслідуванні від класу GetxController з бібліотеки GetX для контролю за станами та обробки даних.

Рівень представлення був поділений на три частини.

Одна для відображення у браузері, яка використовується для адміністрування над бронюваннями та додавання ресторанів у базу. Друга для мобільних пристроїв, де користувач може подивитись заклади та забронювати там стіл.

Третя частина - спільна. Це авторизація та реєстрація, яка присутня у обох частинах. Також є віджети, які використовуються у обох частинах.

Діаграму рівнів наявна у Додатку Ґ.

3.5 Опис розробки та застосунку

Для розробки практичної частини використовувався Flutter останньої версії 2.0.4, який вийшов якраз під час розробки.

Спочатку був створения проект за допомогою Flutter CLI, який був назван GoRes:

flutter create gores

Також був додан контроль версій Git та підключена можливість запуску у Web. Тому що на момент створення проекту Flutter for Web ще не знаходився у стандартному SDK.

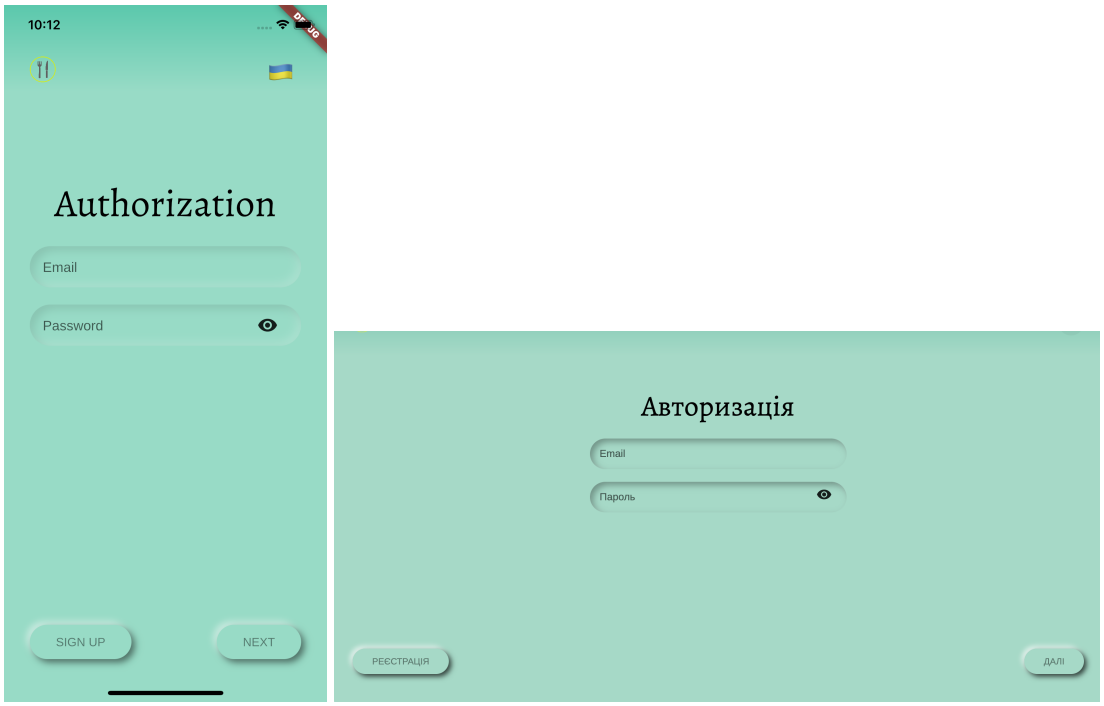
Був створений проект у Firebase та підключені усе необхідні бібліотеки для роботи з ним. Після цього потрібно було додати додаткові налаштування окремо у android та ios директорії.

Після додавання усіх необхідних бібліотек було створено скелет проекту. Був налаштований сервіс для керування мовою застосунку (Додаток Д). Додана можливість ін’єкції залежностей за допомогою бібліотеки GetX (Додаток Е).

Далі почався етап розробки авторизації та реєстрації. Були створені відповідні класи: AuthRepository, LoginController, SignUpController, LoginPage та SignUpPage(Додаток Є). Для збереження користувачів був використан Firebase Authentication.

Розглянемо на прикладі авторизації, як відбувалась розробка кожного рівня:

- LoginPage



У верхній частині сторінки розмістили віджет DefaultAppBar з логотипом та кнопкою перемикання мови. Для цього створили окремий віджет.

Нижче йде пропуск, за ним - текст та поля вводу для email(DefaultTextField) та пароль(DefaultPassword). У полі вводу з правого боку є кнопка для перегляду пароля.

У нижній частині присутні дві кнопки(DefaultButton): “Реєстрація” та “Далі”. Які відповідно переходять на екран реєстрації та намагаються увійти у систему.

- LoginController

У цьому класі є поля для збереження інформації з LoginPage: email та password. Після натискання кнопки “Далі” на екрані, відбувається перевірка значень полів на валідність, чи дійсно користувач ввів email, чи не пусті поля тощо. Після того дані передаються у функцію login класу AuthRepository, яка повертає бульове значення. Якщо повертається true, то користувач авторизувався і його перенаправляють на домашню сторінку. На веб версії вона одна, а на мобільній - інша. Якщо ж повертається false, то користувач отримує повідомлення про помилку.

- AuthRepository

Цей клас відповідає за авторизацію, реєстрацію та слідкує за поточним станом користувача. Через нього відбувається спілкування з Firebase Authentication сервісом.

Розглянемо функцію login.

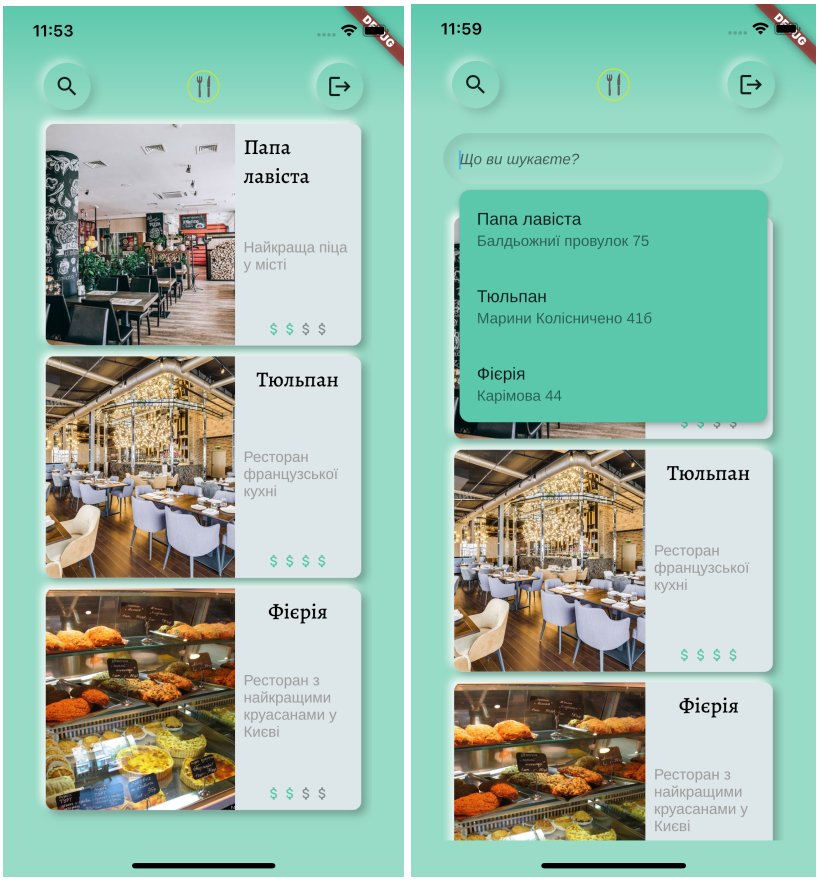
```
Future<bool> login(String email, String password) async {
  try {
    final userCred = await _firebaseAuth.signInWithEmailAndPassword(
      email: email,
      password: password,
    );
    if (userCred.user != null) {
      return await initProfile(userCred.user!.uid);
    }
    return false;
  } on FirebaseAuthException catch (e) {
    _proccessException(e);
    return false;
  } catch (e) {
    snackbarError(errorStr.tr, unknownError.tr);
    return false;
  }
}
```

Викликається функція signInWithEmailAndPassword класу FirebaseAuth, в яку передається email та пароль. Якщо користувач існує, то викликається функція initProfile, в якій збирається інформація про користувача з бази даних Firestore. Якщо ж користувача не існує, то повертається відповідне повідомлення.

Після етапу авторизації та реєстрації користувача перенаправляє на домашню сторінку, в залежності від платформи.

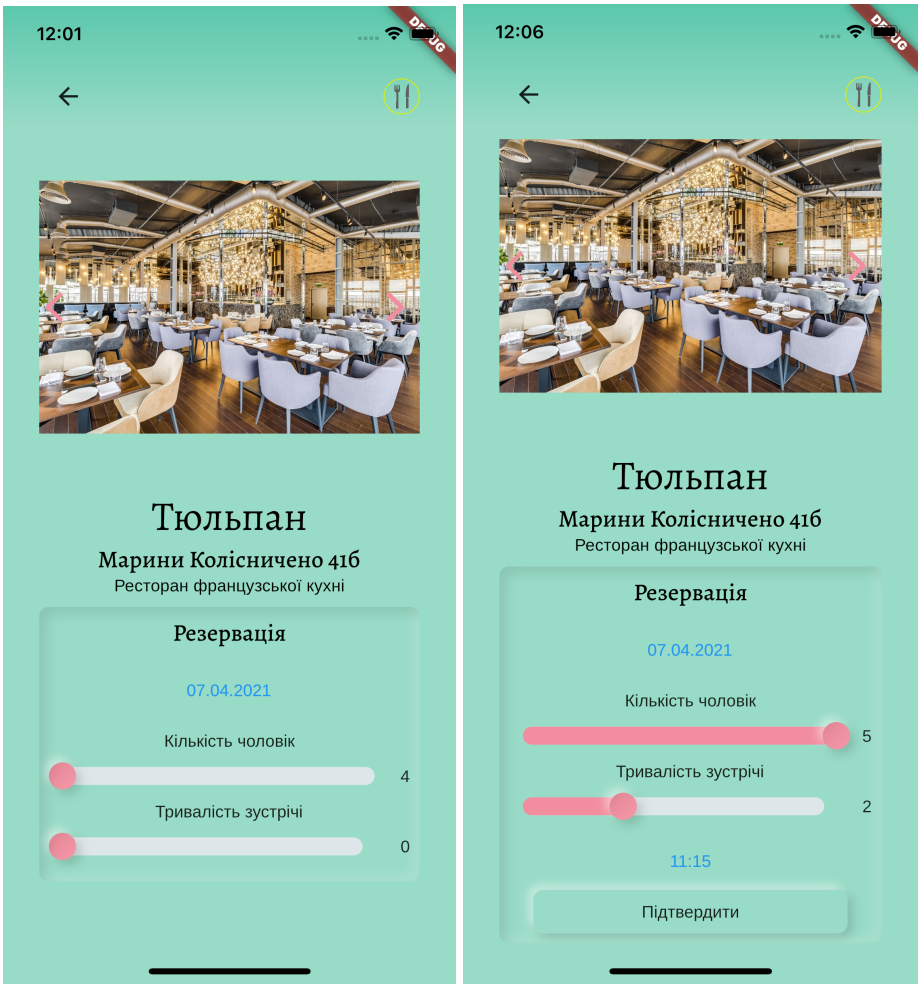
- Мобільний застосунок

Користувач бачить список ресторанів, кнопку пошука та виходу. На картці ресторану є зображення, назва, опис, короткий опис та ціновий порядок.
У пошуку можна обрати ресторан, і він одразу перейде на сторінку.



На сторінці ресторану можна побачити зображення, подивитись їх. Також можемо зробити бронювання, для цього потрібно обрати: дату, кількість

чоловік, тривалість, час та підтвердити. У разі якщо місць на даний час не має, з’явиться повідомлення.



- Браузер

На головній сторінці є список ресторанів, які належать цьому користувачу-адміністратору. Якщо відкрити його, то можна подивитись бронювання столиків на конкретну дату.

Одразу після списка ресторанів, є кнопка для реєстрації нового ресторана у системі. На цій сторінці є поля для вводу імен, опису, адреси, вартості. Можна обрати титульне зображення та додаткові. Є можливість додавання кількості столів з кількістю місць для персон у ресторані. Скріншоти дивитись у Додатку Ж.

ВИСНОВОК

В результаті виконання роботи відбулося ознайомлення с фреймворком Flutter та мовою програмування Dart. Переглянуті їх особливості, історія створення та багато іншого.

Практичним результатом став кросплатформений застосунок для Android, iOS та Web з єдиним програмним кодом, який дає можливість бронювати столики у ресторанах з боку мобільного додатку та додавати ресторани у систему з боку веб додатку.

Для керування станами була обрана бібліотека GetX, тому що вона легка та потужна водночас, та має багато сервісів, які полегшують розробку.

Для збереження даних та у якості серверної частини був обран хмарний сервіс Firebase, в першу чергу, тому що він надає безкоштовний доступ до багатьох своїх сервісів. Також він має гарну документацію та SDK під велику кількість фреймворків та мов програмування.

СПИСОК ДЖЕРЕЛ

1. **Statista.com** - Number of smartphone users worldwide from 2016 to 2023
   https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

2. **gotocon -** Presentation: "Opening Keynote: Dart, a new programming language for structured web programming"
   http://gotocon.com/aarhus-2011/presentation/Opening

3. **news.dartlang.org -** Dart for the Entire Web
   https://news.dartlang.org/2015/03/dart-for-entire-web.html

4. **medium.com -** Announcing Dart 2 Stable and the Dart Web Platform
   https://medium.com/dartlang/dart-2-stable-and-the-dart-web-platform-3775d5f8eac7

5. **sdtimes.com -** Dart 2.6 released with dart2native
   https://sdtimes.com/goog/dart-2-6-released-with-dart2native/

6. **medium.com -** Announcing Dart 2.12
   https://medium.com/dartlang/announcing-dart-2-12-499a6e689c87

7. **twitter.com -** Flutter is the default choice for future Ubuntu apps
   https://twitter.com/ubuntu/status/1367063203600031746

8. **medium.com -** A Brief History of Flutter
https://medium.com/pragmatic-programmers/a-brief-history-of-flutter-939645f93255

9. **medium.com -** Flutter 1.0 : An Introduction
https://medium.com/swlh/flutter-1-0-an-introduction-cb9540b2e1a

10. **medium.com -** What's New in Flutter 2
https://medium.com/flutter/whats-new-in-flutter-2-0-fe8e95ecc65

11. **flutter.dev -** Introduction to widgets
https://flutter.dev/docs/development/ui/widgets-intro

12. **codemagic.io -** Flutter Versus Other Mobile Development Frameworks: A UI And Performance Experiment
https://blog.codemagic.io/flutter-vs-ios-android-reactnative-xamarin/

13. **github.com -** getx
https://github.com/jonataslaw/getx

14. **pub.dev -** flutter_neumorphic
https://pub.dev/packages/flutter_neumorphic

15. **firebase.google.com -** Firebase Authentication
https://firebase.google.com/docs/auth

16. **firebase.google.com -** Cloud Firestore
https://firebase.google.com/docs/firestore

17. **firebase.google.com -** Cloud Storage
https://firebase.google.com/docs/storage

18. **firebase.google.com -** Google Analytics
https://firebase.google.com/docs/analytics

19. **firebase.google.com -** Firebase Crashlytics
https://firebase.google.com/docs/crashlytics

20. **pub.dev -** flutter_svg
https://pub.dev/packages/flutter_svg

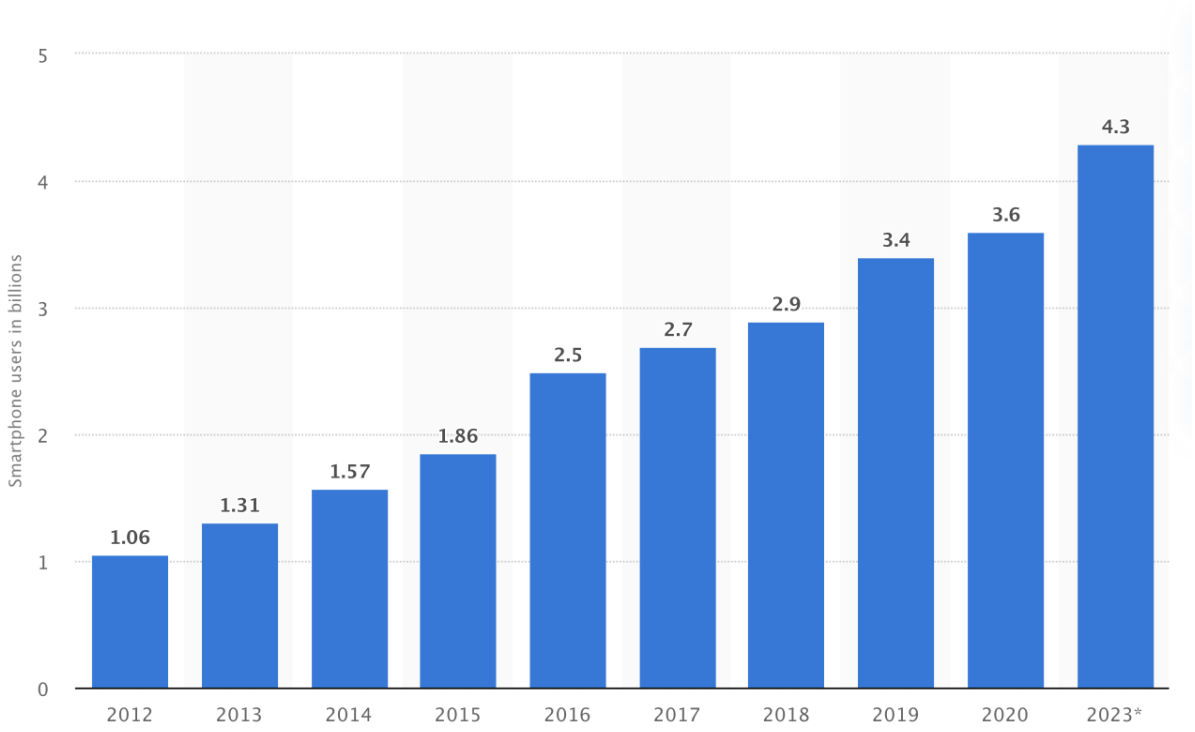
21. **pub.dev -** equatable
https://pub.dev/packages/equatable

22. **pub.dev -** json_serializable
https://pub.dev/packages/json_serializable

23. **flutter.dev -** DevTools
https://flutter.dev/docs/development/tools/devtools/overview

Додатки

Додаток А



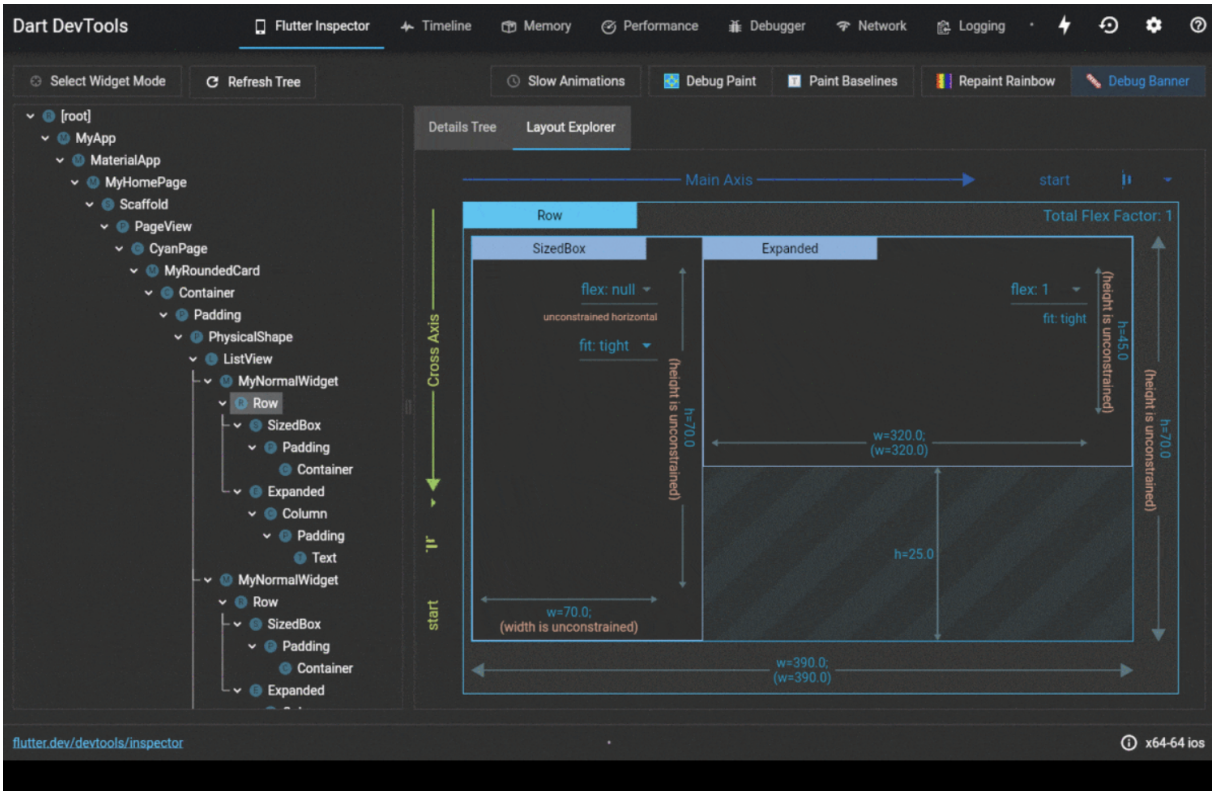
Додаток Б

0

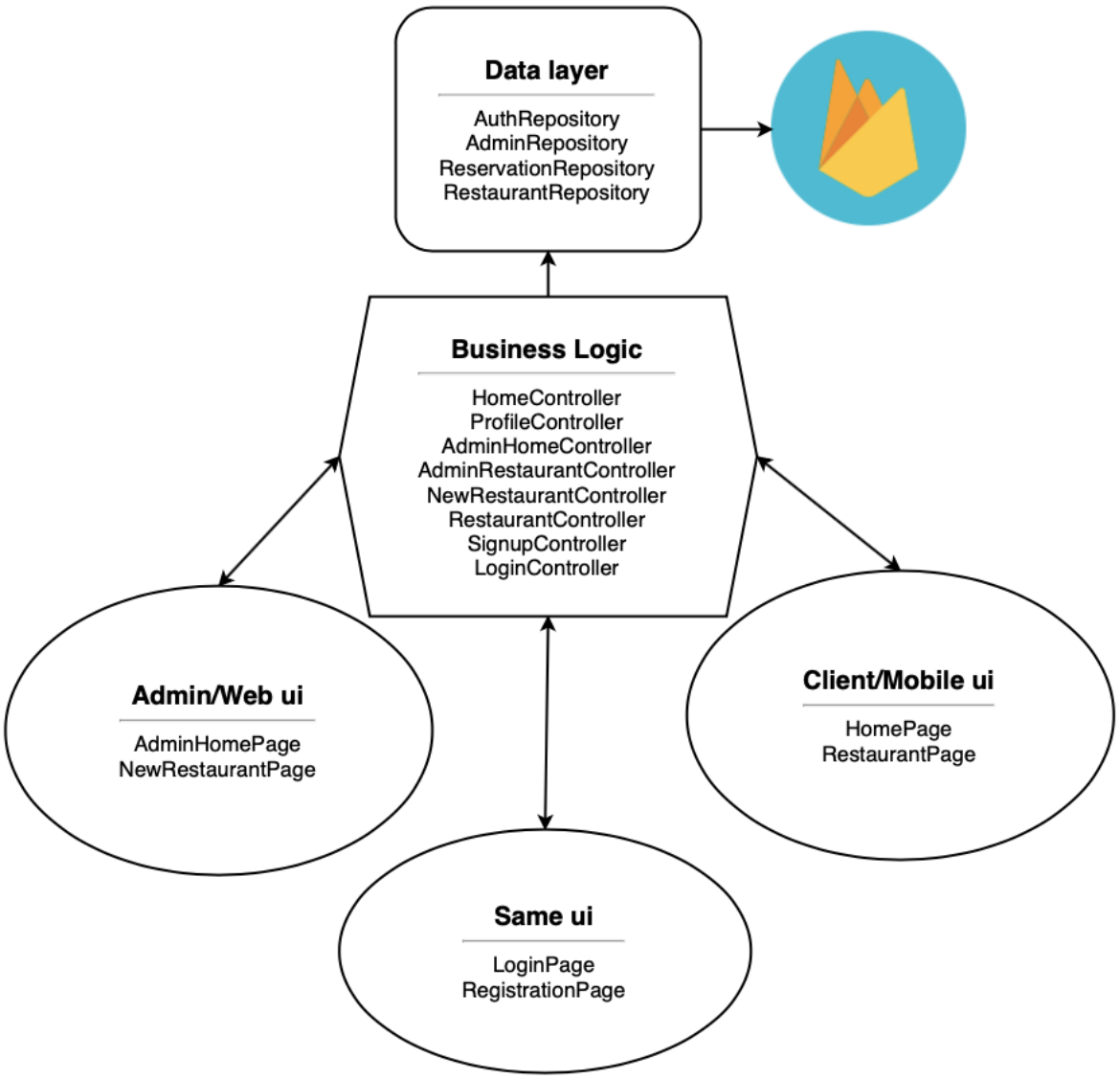
Додаток В

1

Додаток Г


Dart DevTools
Flutter Inspector
Timeline
Memory
Performance
Debugger
Network
Logging
Select Widget Mode
Refresh Tree
Slow Animations
Debug Paint
Paint Baselines
Repaint Rainbow
Debug Banner
[root]
MyApp
MaterialApp
MyHomePage
Scaffold
PageView
CyanPage
MyRoundedCard
Container
Padding
PhysicalShape
ListView
MyNormalWidget
Row
SizedBox
Padding
Container
Expanded
Column
Padding
Text
MyNormalWidget
Row
SizedBox
Padding
Container
Expanded
Details Tree
Layout Explorer
Main Axis
start
Cross Axis
start
Row
Total Flex Factor: 1
SizedBox
Expanded
flex: null
unconstrained horizontal
fit: tight
h=70.0
(height is unconstrained)
flex: 1
fit: tight
h=45.0
(height is unconstrained)
w=320.0;
(w=320.0)
h=25.0
h=70.0
(height is unconstrained)
w=70.0;
(width is unconstrained)
w=390.0;
(w=390.0)
flutter.dev/devtools/inspector
x64-64 ios

Додаток Ґ


Data layer
AuthRepository
AdminRepository
ReservationRepository
RestaurantRepository
Business Logic
HomeController
ProfileController
AdminHomeController
AdminRestaurantController
NewRestaurantController
RestaurantController
SignupController
LoginController
Admin/Web ui
AdminHomePage
NewRestaurantPage
Client/Mobile ui
HomePage
RestaurantPage
Same ui
LoginPage
RegistrationPage

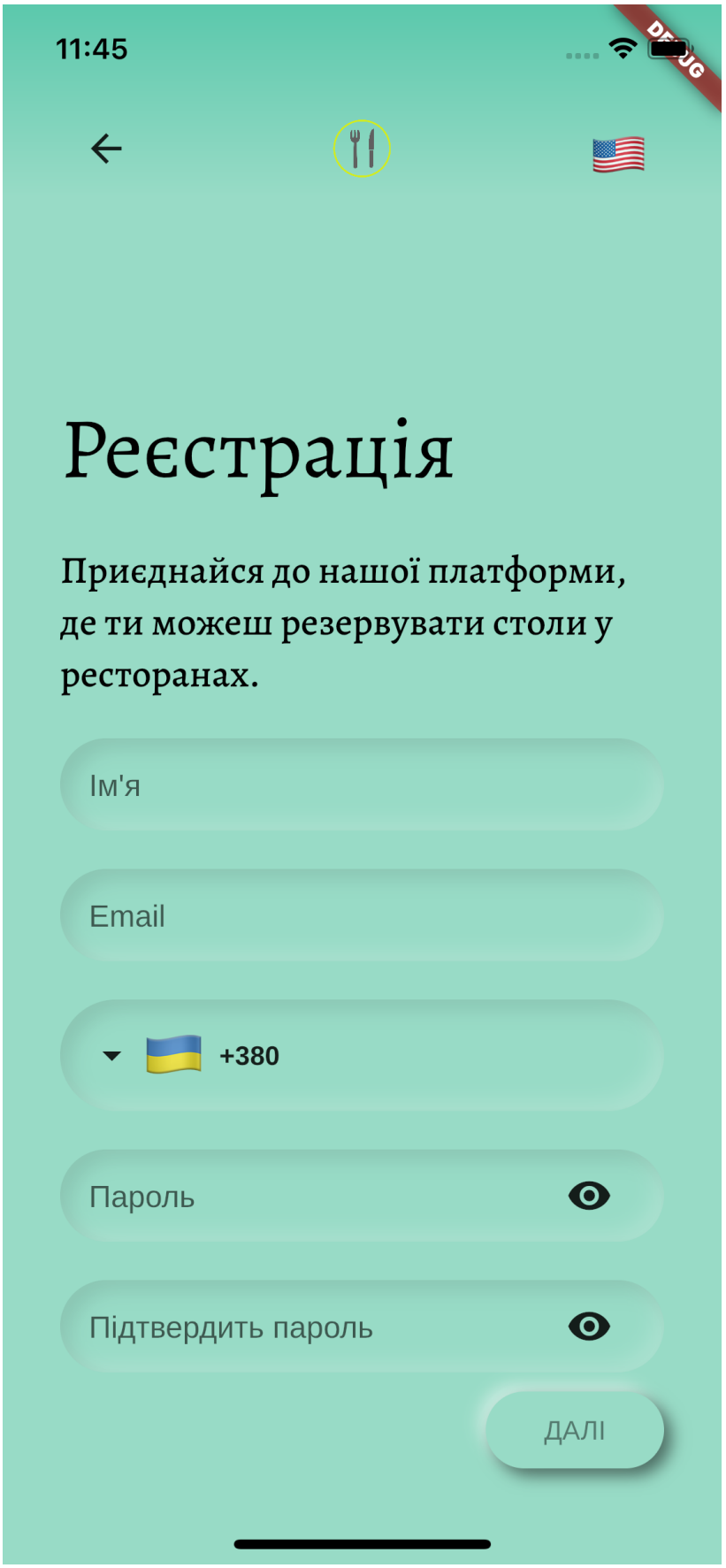
Додаток Д

```
class TranslationService extends Translations {
  static final locale = en;
  static final fallbackLocale = en;
  static const en = Locale('en', 'US');
  static const ua = Locale('ua', 'UA');

  @override
  Map<String, Map<String, String>> get keys => {
        '$en': en_US,
        '$ua': ua_UA,
      };

  Future<void> changeLocale(Locale locale) async {
    final _sharedPreferences = await SharedPreferences.getInstance();

    await _sharedPreferences.setString("languageCode", locale.languageCode);
    await _sharedPreferences.setString("countryCode", locale.countryCode!);

    Get.updateLocale(locale);
  }

  Future<void> restoreLocale() async {
    final _sharedPreferences = await SharedPreferences.getInstance();

    final langCode = _sharedPreferences.getString("languageCode");
    final countCode = _sharedPreferences.getString("countryCode");

    if (langCode != null && countCode != null) {
      Get.updateLocale(Locale(langCode, countCode));
```

Додаток Е

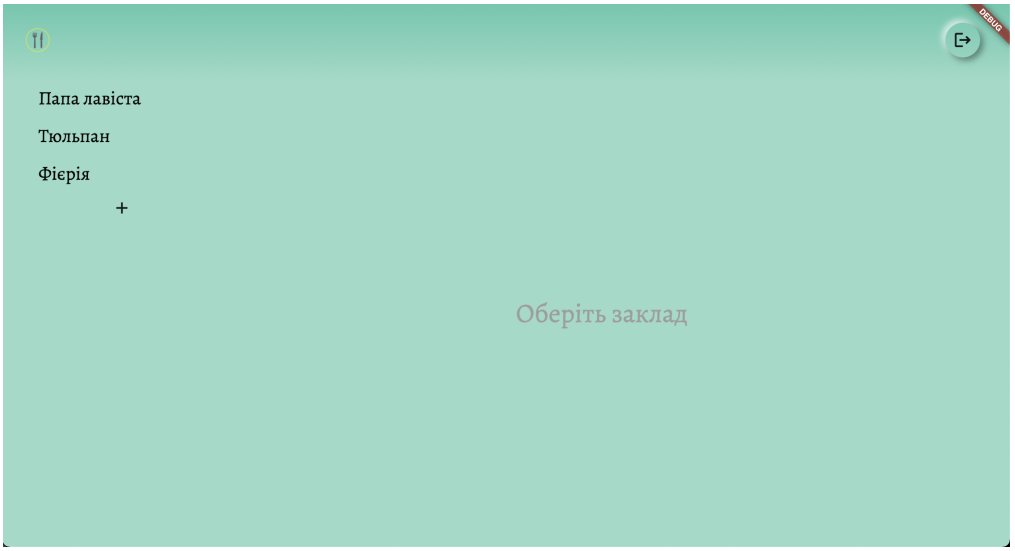
```
class InitialBinding implements Bindings {
  @override
  void dependencies() {
    Get.lazyPut<AuthRepository>(() => AuthRepository());
    Get.lazyPut<FilePickerManager>(() => FilePickerManager());
    Get.lazyPut<FileStorageManager>(() => FileStorageManager());
    Get.lazyPut<AdminRepository>(() => AdminRepository(
          Get.find(),
          Get.find(),
          Get.find(),
        ));
    Get.lazyPut<RestaurantRepository>(() => RestaurantRepository(Get.find()));
    Get.lazyPut<ProfileController>(() => ProfileController());
    Get.lazyPut<LoginController>(
        () => LoginController(authRepository: Get.find()));
    Get.lazyPut<HomeController>(() => HomeController(Get.find()));
    Get.lazyPut<SignUpController>(
        () => SignUpController(authRepository: Get.find()));
    Get.lazyPut<AdminRestaurantController>(
        () => AdminRestaurantController(Get.find(), Get.find()));
    Get.lazyPut<ReservationRepository>(() => ReservationRepository(Get.find()));
    Get.lazyPut<AdminHomeController>(
        () => AdminHomeController(Get.find(), Get.find()));
  }
}
```

Додаток Є


11:45
DEBUG
Реєстрація
Приєднайся до нашої платформи, де ти можеш резервувати столи у ресторанах.
Ім'я
Email
+380
Пароль
Підтвердить пароль
ДАЛІ

Додаток Ж


Папа лавіста
Тюльпан
Фієрія
+
Резервація
06.04.2021
Телефон клієнта
Початок
Кінець
Кількість осіб


Папа лавіста
Тюльпан
Фієрія
+
Оберіть заклад


Ім'я
Опис
Адреса
Вартість
$ $ $ $ $
Оберіть головне зображення
Оберіть зображення
Столи
Предогляд