Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

# Извештај за изработен проект

**Тема: TransportEase – ride-sharing апликација**

transportease

**Изработил:**

Горги Лазарев

201042

# Содржина

# 1. Вовед во Laravel

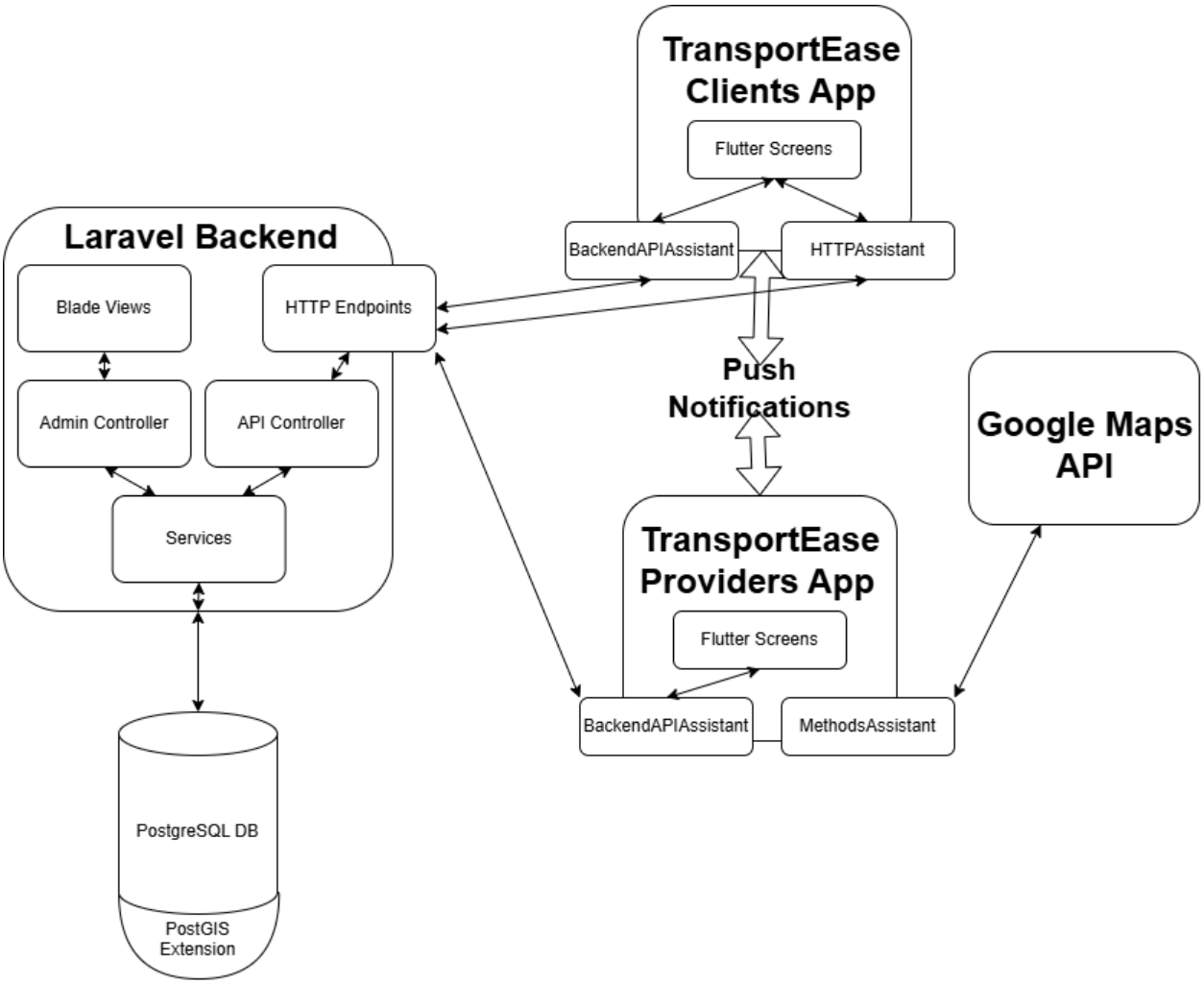
Laravel претставува open-source PHP framework за изработка на веб апликации и сервиси кој што го следи MVC (Model-View-Controller) патернот и реискористува компоненти од различни frameworks и библиотеки.

Laravel има многу достапни библиотеки и многу елегантна и експресивна синтакса што им овозможува на девелоперите пишување на читлив код и брз развој на апликации.

Покрај ова, има многу клучни функционалности кои го прават еден од најдобрите frameworks за развој на веб апликации:

- Artisan CLI – Laravel вклучува многу моќен command-line tool наречен Artisan со кој на едноставен начин се генерираат миграции, контролери, модели, менаџирање со шемата на базата на податоци, стартување на локален сервер и многу други функции
- Eloquent ORM – ORM кој што нуди интерфејс за едноставна интеракција со ентитети од базата на податоци и самите табели во базата на податоци. Овозможува дефинирање на шемата на базата на податоци преку код, испраќање на queries и пристап до ентитети од базата директно преку нивните модели. Исто така овомзможува подесување на голем број на достапни провајдери на бази на податоци
- Интеграција со пакети и библиотеки – Преку Composer – package manager-от на Laravel лесно се интегрираат библиотеките во Laravel и се прошируваат функционалностите
- Автентикација и авторизација – преку Laravel Sanctum се нуди built-in автентикација и авторизација која што лесно се подесува преку Middlewares, пермисии и екстензија на шемата на корисникот
- Blade Templating Engine – овозможува реискористување на погледите со динамичен приказ на содржина со услови, циклуси и layouts.
- Екстензивна документација и заедница – има голема заедница и доста екстензивна документација со која што програмерите може да се консултираат при наидување на проблем

## 2. Архитектура на апликацијата



# 3. Вовед во крос-платформски апликации и Flutter и Dart

Во денешен ден, апликациите се распространети насекаде и се дел од нашиот секојдневен живот. Најразлични апликации се користат за најразлични цели, од апликации за навигација и фитнес, до апликации за научно истражување и е-комерц, секој може да најде апликација која одговара на своите потреби и интереси. Апликациите најчесто ги подобруваат аспектите на нашиот живот и ја зголемуваат продуктивноста и можноста за комуникација, а тоа се должи на нивната распространетост на најразлични уреди, од компјутери и лаптопи до таблети, мобилни телефони и паметни часовници.

Крос-платформски апликации претставуваат одличен начин за развој на апликации кои можат да работат на повеќе оперативни системи, како Android, iOS, Windows, Linux, MacOS и Web-от со користење на ист код за сите платформи. Токму ова всушност го претставува и најголемиот бенефит од развивањето на крос-платформски апликации што тие се создаваат со еден ист код, со што се заштедува на време и ресурси и нудат ефикасен начин на развој на апликации. Покрај тоа, крос-платформските апликации овозможуваат поедноставен процес на одржување и надградба, бидејќи измените и поправките можат да се воведуваат на едно место и да се ажурираат на сите платформи автоматски.

Еден од иновативните начини за креирање на крос-платформски мобилни апликации е користењето на Flutter и Dart. Оваа рамка за развој на крос-платформски апликации стануваат се пошироко застапени и претставуваат врвен инструмент во сферата на мобилните апликации и нудат многу можности за креирање на привлечни, брзи и функционални апликации.


Flutter е отворена рамка за развој на крос-платформски апликации, која е создадена од страна на Google. Таа овозможува креирање на крос-платформски апликации кои можат да работат на различни оперативни системи како што се Android и iOS, Windows, Linux и MacOS и Web-от со идентичен изглед и

функционалност. Она што го прави Flutter особено атрактивен е неговата брзина и перформанси, што резултира во извонредно респонзивни апликации.

Со секоја нова верзија, Flutter се подобрува и збогатува со нови функционалности и можност за развој на апликации. Тоа го прави Flutter една од најзначајните технологии во областа на мобилниот развој и остава долготрајно влијание во индустријата.


Dart, пак, е програмскиот јазик врз кој што се базира Flutter и е исто така развиен и поддржуван од страна на Google. Овој јазик е специјално дизајниран за развој на мобилни апликации со Flutter. Dart се одликува со својата ефикасност, брзина и лесна читливост, што го прави идеален за програмерите кои сакаат да создаваат квалитетни мобилни апликации.

Една од главните карактеристики на Dart е тоа дека е објектно-ориентиран јазик, што значи дека објектите и класите се основни компоненти во кодот. Ова им дава на програмерите можност да организираат и управуваат со кодот на подобар начин и да го направат повеќе структуриран и читлив.

Dart се истакнува и со својата ефикасност и брзина во извршувањето на кодот. Овој јазик користи just-in-time компајлер за развој и дебагирање на апликациите и автоматско преведување во машински код при извршување на апликацијата. Ова овозможува брзина и лесно дебагирање при развојот на апликации.

Со сите овие карактеристики и предности, Dart е јазик кој продолжува да расте во популарност и да се користи за развој на иновативни апликации во современите информатички индустрии.

Големиот број на достапни библиотеки и готови widget-и и компоненти за градење на кориснички интерфејси се главна причина, покрај можноста за развивање на крос-платформски апликации, поради која оваа рамка за развој на апликации, Flutter и програмскиот јазик Dart секојдневно добиваат нови корисници и поддршка.

Исто така, заедницата на корисници на Flutter и Dart е доста активна и нуди голема поддршка за секој што ја користи оваа рамка преку споделување на своите знаења и искуства.

Flutter и Dart, како и сите библиотеки и готови компоненти се многу добро документирани и содржат богати документации прикрепени со примери за лесна адаптација и користење во нашите апликации без да мора да бараме на милион места за да решиме некој проблем или имплементираме некое решение.

Како крај на воведот можеме да потенцираме дека Flutter и Dart се многу моќна алатка за развивање на крос-платформски апликации, а со револуцијата и се поголемата застапеност на паметните мобилни и мали уреди, оваа алатка е многу привлечна и за оние кои прв пат се среќаваат со ваков тип на апликации и заедницата допрва ќе расте.

# 4. Архитектура на една Flutter и Dart апликација

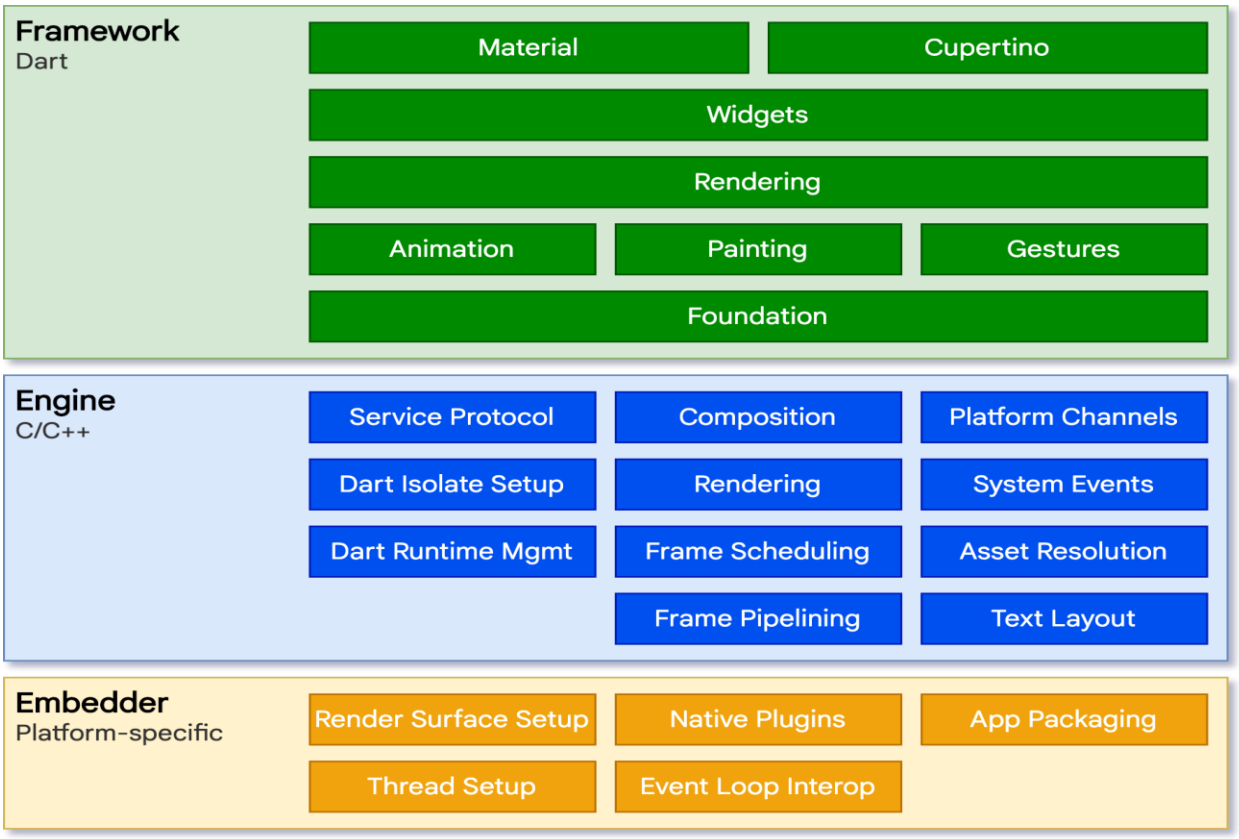
Flutter е изграден како слоевит систем од независни библиотеки кои што зависат од слојот под себе и секој слој од framework-от е дизајниран да биде опционален и заменлив.

Јадрото на Flutter го претставува Flutter Engine-от во кој се имплементирани сите имплементации на ниско ниво, од кои најголем дел се напишани во C++, и е задолжен за растеризацијата на сцените по вчитување на нов frame кој треба да се исцрта на екранот.

Engine-от преку dart:ui се поврзува со Framework-от на тој начин што C++ кодот се wrap-нува во Dart класи кои се задолжени за влез/излез, графика и рендерирање на текст.

Embedder-от претставува компонента која зависи од платформата на која што Flutter се извршува кој што нуди влез во самата програма и ја координира комуникацијата со оперативниот систем врз кој што се извршува со цел да ги овозможи сервисите на оперативниот систем кон Flutter Framework-от. Embedder-ите се напишани во различни јазици специфични за платформата, пр. Java и C++ за Android, Objective-C и Objective-C++ за iOS и macOS, C++ за Windows и Linux.

За разлика од ова, кога Flutter се извршува на Web, Framework-от, односно Dart кодот директно се компајлира во JavaScript кој го разбираат прелистувачите.




*Слика 1: Слоевита архитектура на Flutter.*

*Извор:https://docs.flutter.dev/resources/architectural-overview*

### a. Layout на Flutter апликација

За корисничките интерфејси Flutter ја минимизира апстракцијата, односно не ги користи директно widget-ите и UI библиотеките на оперативниот систем на кој што се извршува, туку си има развиено свое множество на widget-и кое се извршува преку Dart код кој понатаму се претвара во нативен код за платформата врз која што се извршува.

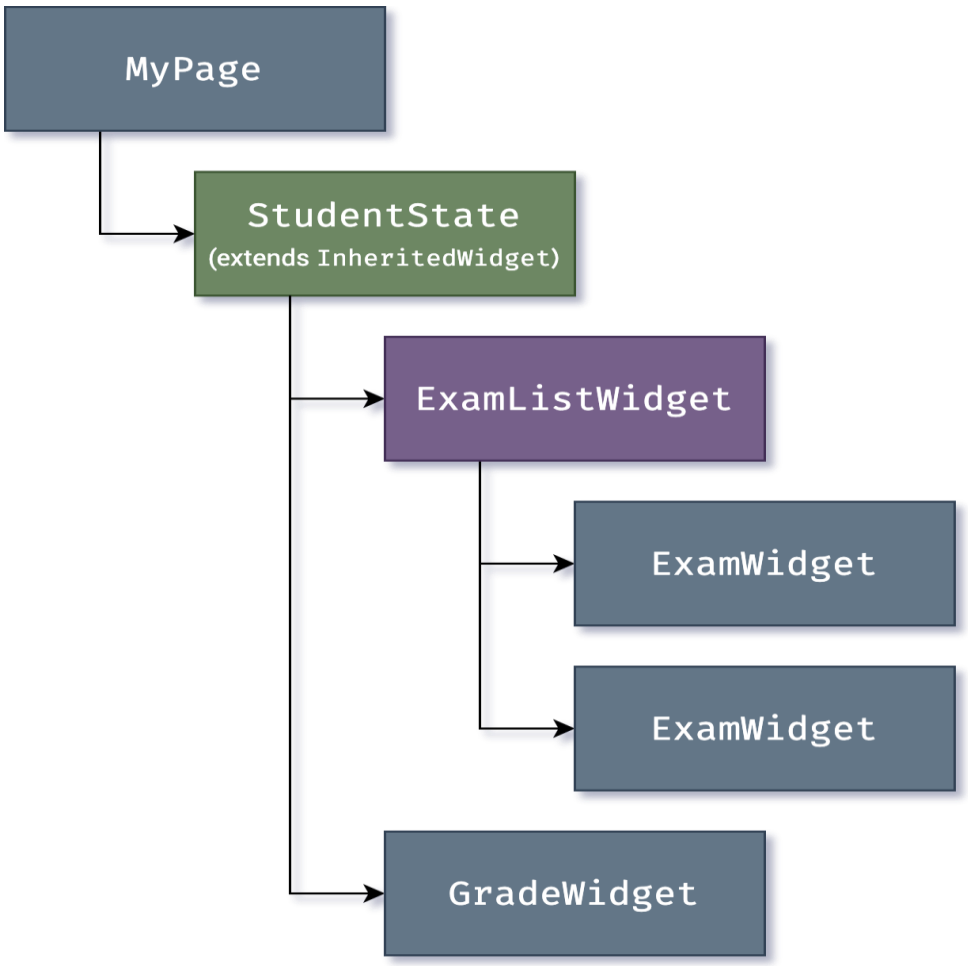
Flutter Widget-ите се организирани како дрво од елементи, при што секој елемент, односно widget се гради преку повик на неговиот build метод.

```dart
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My Home Page'),
        ),
        body: Center(
          child: Builder(
            builder: (context) {
              return Column(
                children: [
                  const Text('Hello World'),
                  const SizedBox(height: 20),
                  ElevatedButton(
                    onPressed: () {
                      print('Click!');
                    },
                    child: const Text('A button'),
                  ),
                ],
              );
            },
          ),
        ),
      ),
    );
  }
}
```

*Слика 2: Код на еден stateless widget.*

*Извор:https://docs.flutter.dev/resources/architectural-overview*

Widget-ите во Flutter, како што се гледа и на кодот од Слика 2 и на дијаграмот од Слика 3 се вгнездуваат еден во друг за да го формираат дрвото од елементи, пришто постојат елементи во кои може да се вгнезди само еден widget, пр. Center, Align и Padding и елементи во кои што може да се вгнездат повеќе widget-и, пр. Container, Row и Column.



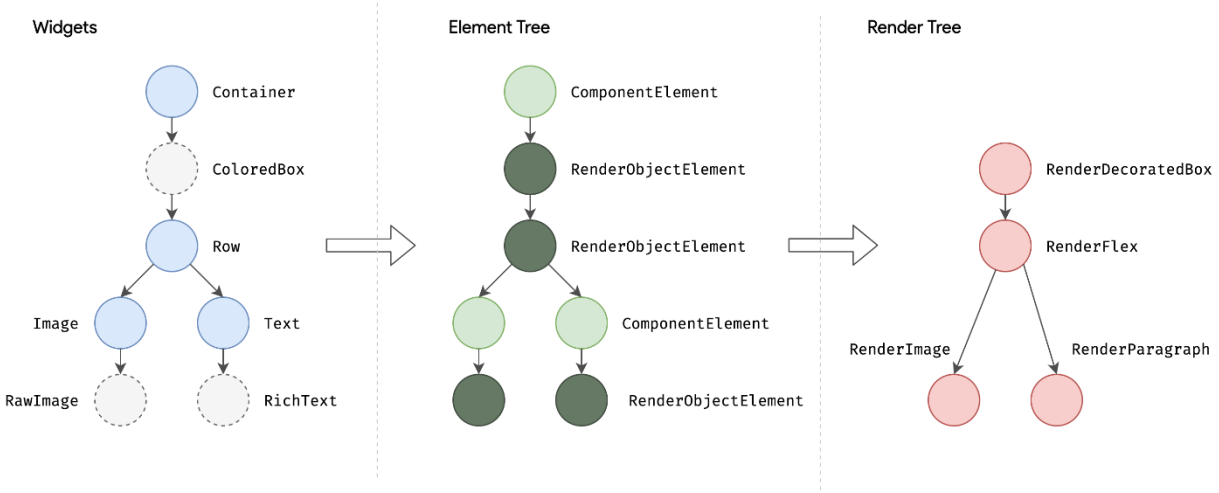

*Слика 3: Структура на еден widget во кој се вгнездени повеќе widget-и.*

*Извор:https://docs.flutter.dev/resources/architectural-overview*

Во Flutter постојат два типа на widget-и, односно елементи и тоа:

- **ComponentElement** – елементи кои преставуваат вид на контејнери, односно хостови (обвивки) за други елементи
- **RenderObjectElement** – елементи кои се вклучени во layout-от и рендерирањето на апликацијата

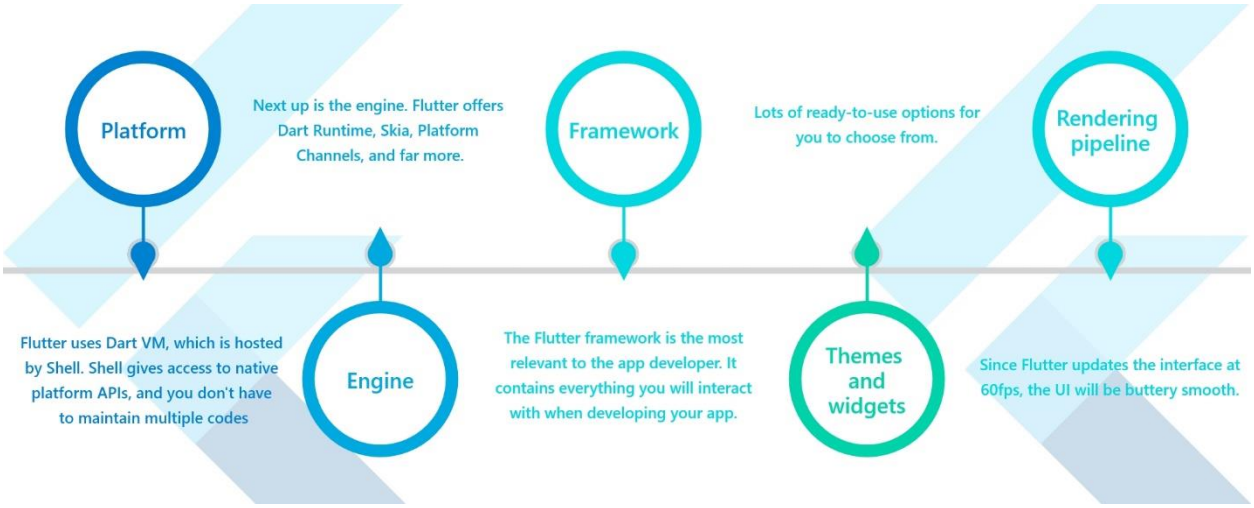
При рендерирањето на widget-ите тие најпрво се претвоаааат во елементи од соодветниот тип и потоа тоа дрво од елементи при фазата на градење на layout-от се претвора во рендер дрво во кое учествуваат само елементите кои се од тип RenderObjectElement.




*Слика 4: Процес на градење на layout-от во Flutter.*

*Извор:https://docs.flutter.dev/resources/architectural-overview*

## b. Workflow на Flutter апликација




*Слика 5: Workflow на една Flutter апликација*

*Извор: https://www.inheritx.com/blog/what-is-flutter-and-how-flutter-works*

Секоја Flutter апликација го користи Dart VM, кој е хостиран од Shell и нуди пристап до нативните API кои ги нуди платформата на која што се извршува, па тоа ни овозможува да пишуваме и одржуваме само еден код, а нашата апликација да работи на повеќе платформи.

Потоа се користи Flutter Engine-от за извршување на имплементациите од ниско ниво, како што е објаснето на почетокот во втората секција.

Она со што програмерите интерактираат е Flutter Framework-от кој содржи се што им е потребно на програмерите за развој на Flutter апликации, а дополнително се достапни и голем број на библиотеки со теми и widget-и кои се претвараат во дрво од елементи, како што е објаснето во секција 2.а.

На крај ова дрво од елементи се рендерира и на корисниците им се прикажува интерфејсот кој што се ажурира во 60 frame-ови во секунда.

# 5. Опис на развиената апликација

Апликацијата која што ја развив како дел од оваа проектна задача преставува апликација за наоѓање на превозници во рамките на Република Македонија. Платформата која што е развиена, всушност претставува две посебни апликации, една за клиентите, односно корисниците на кои им е потребен превоз и една за превозниците, односно провајдерите на услугата.

Апликацијата е инспирирана од Uber и Lyft, кои претставуваат најпопуларните апликации за барање на превоз и споделување на превоз во светот.

Апликацијата на клиентите им нуди платформа преку која можат да побараат превоз до било која дестинација во рамките на Македонија и во моментот на пишување можат да изберат од два типа на превозници, такси возило и приватно возило, при што при барањето и селекцијата на дестинацијата, автоматски се лоцира моменталната локација на корисникот и се пресметува трошокот за различните типови на превоз, како и оддалеченоста од дестинацијата и на корисникот му се дава можност за избор на тип на превозник и му се прикажуваат податоци како што се оддалеченоста, трошокот и му се исцртува целата рута која во тој момент е одберана како најкратка и најдобра на мапата во апликацијата. Откога корисникот ќе го избере посакуваниот тип на превоз, тој може да го поднесе барањето кое што ќе се пропагира до сите превозници од одбераниот тип кои се наоѓаат во радиус од 5 километри од моменталната локација на корисникот и при што се приоритизираат превозниците кои се најблиску до корисникот. Доколку има достапен превозник од посакуваниот тип на корисникот кој го прифатил барањето за превоз, тој се насочува кон корисникот и на корисникот му се прикажува во реално време статусот на патувањето, најпрво, додека да стигне превозникот до

местото на клиентот му се прикажува колку минути е оддалечен според актуелната состојба на превозникот и сообраќајот каде што се движи, потоа кога превозникот ќе пристигне до корисникот и корисникот ќе го започне патувањето, на корисникот му е прикажана рутата по која би требало да се движи превозникот и која е најдобра и најбрза во тој момент како и преостанатото време до пристигањето на дестинацијата. Откако корисникот ќе пристигне на дестинацијата и превозникот ќе го заврши патувањето, на корисникот му се прикажува приказ за износот кој треба да го плати за патувањето со платежната метода која ја избрал, во кеш или преку платежна картичка. По плаќањето, на корисникот му се прикажува екран за оценување на превозникот според искуството кое корисникот го поминал во текот на тоа патување. Ова е клучно за задржување само на добрите превозници и нивно приоритизирање при доделувањето на патувањата. Исто така на корисникот му е достапна можноста да го повика превозникот преку бројот кој што е запишан во системот и можност за откажување на патувањето пред да започне истото. Доколку пак во моментот на барање на превоз нема достапни превозници од бараниот тип, на корисникот му се прикажува порака дека може да побара друг тип на превозник или пак да почека одредено време додека да стане достапен превозник од бараниот тип, или пак доколку нема ниту еден достапен превозник, на корисникот му се прикажува порака да почека одредено време додека да се појави достапен превозник.

Од друга страна пак, апликацијата им овозможува на превозниците од различни типови да се регистрираат на платформата преку посебната апликација за превозници и да го зголемат обемот на работа а со тоа и можностите за профит со тоа што ќе бидат достапни кон сите корисници на апликацијата. Во моментот како превозници може да се регистрираат обични граѓани, кои имаат свој автомобил и сакаат да вршат превоз на патници со што ќе може дополнително да заработат или пак да ги споделат трошоците со некој доколку се движат на иста дестинација; возачи на такси возила, вработени во такси компании; локални и интернационални превозници на патници како и превозници на пакети – карго превозници. Иако сите овие типови на превозници може да се регистрираат во апликацијата, во моментот апликацијата им е достапна само на граѓаните кои сакаат да вршат превоз со сопствени возила и на возачите на такси возила, вработени во такси компании.

За секој тип на превозник се пополнуваат соодветните детали потребни за регистрација на тој конкретен тип на превозник.

Откако превозник ќе се регистрира и ќе ги пополни потребните податоци и ќе се најави, тој има можност да го промени својот статус како достапен или пак да го исклучи и да биде недостапен. При премин во состојба достапен, локацијата на превозникот се следи во реално време и се ажурира во системот и кај корисниците. Достапен превозник може да добие барање за превоз, доколку тој бил најблизок до одреден корисник кој побарал превоз на кое што му се прикажани локацијата на поаѓање, односно локацијата на корисникот кој бара

превоз и посакуваната дестинација, при што може или да го одбие барањето или да го прифати. По прифаќањето на барањето, на превозникот му се исцртува рутата на мапата по која што тој треба да се движи за да најпрво пристигне до корисникот и му се прикажува предвиденото време до пристигање. Кога превозникот ќе стигне на местото за поаѓање, треба да го ажурира статусот на патувањето преку клик на копче кое е достапно кај превозникот. Откога превозникот и корисникот ќе го започнат патувањето, превозникот соодветно го ажурира статусот на патувањето и му се исцртува нова рута на мапата, која сега е од местото на поаѓање на корисникот до посакуваната дестинација. По пристигањето на посакуваната дестинација превозникот го завршува патувањето и му се прикажува износот на средства кој ќе го добие за тоа патување. По завршувањето на патувањето превозникот може да остане достапен за превоз или пак да го промени својот статус во недостапен.

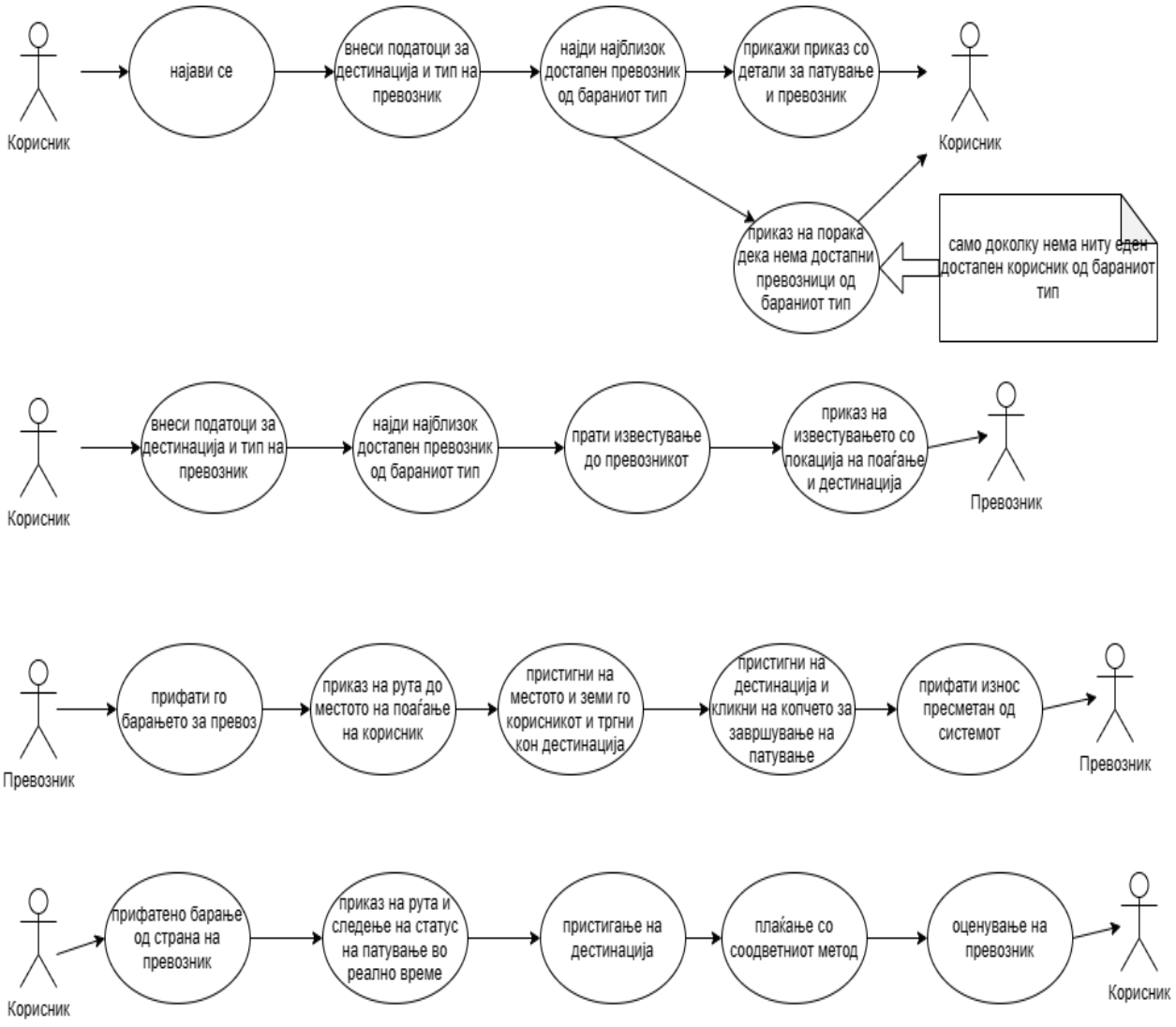
На превозниците им се достапни неколку погледи преку пристап во менито и тоа: главниот поглед, каде што го ажурира својот статус на достапност и ја гледа мапата и му се исцртуваат рутите и појавуваат известувањата, потоа поглед каде што може да ги гледа дотогашните заработки и историја на секое патување кое го има направено преку апликацијата, со приказ на местото на поаѓање, дестинацијата, датумот на кој е завршено патувањето како и износот на средства кој го добил за тоа патување; поглед во кој може да ги гледа своите оценки, на скала од 1 до 5 при што се прикажува текстуален опис на оценките, од Многу лошо до Одлично и на крај поглед на кој се прикажани самите податоци за превозникот, како што се неговото име, телефонски број, емаил и модел на автомобилот со кој врши превоз на патници.

Апликацијата дополнително нуди веб-базиран администраторски панел преку кој што може да се менаџира со корисниците, барањата за превоз и превозниците на апликацијата и дополнително има панел на кој што се прикажани статистики за тоа како се одвиваат функционалностите на платформата како што се број на регистрирани корисници (вкупно и график на кој што се прикажани бројот на регистрирани корисници во последните 30 дена, по денови), бројот на превозници и pie chart за нивна дистрибуција по тип на превозник и број на направени барања за превоз и график на кој се прикажани колку има во последните 30 дена.

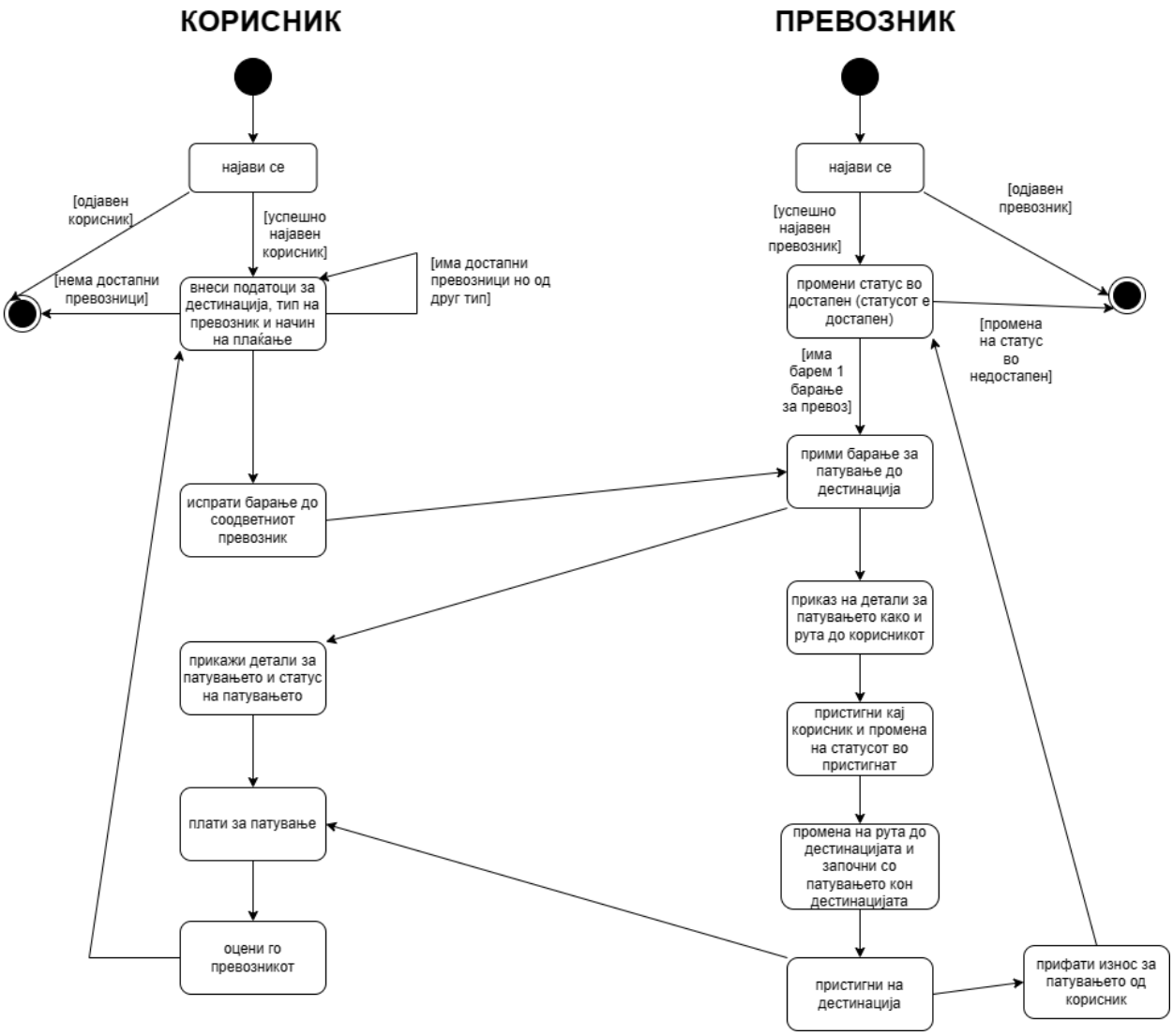
Исто така веб апликацијата за менаџмент им нуди и на останатите корисници менаџирање на нивниот профил преку панелот за менаџмент кој што може да се пристапи преку најава на веб апликацијата за менаџмент која што е хостирана.

# 6. Дијаграми за основните функционалности на апликацијата

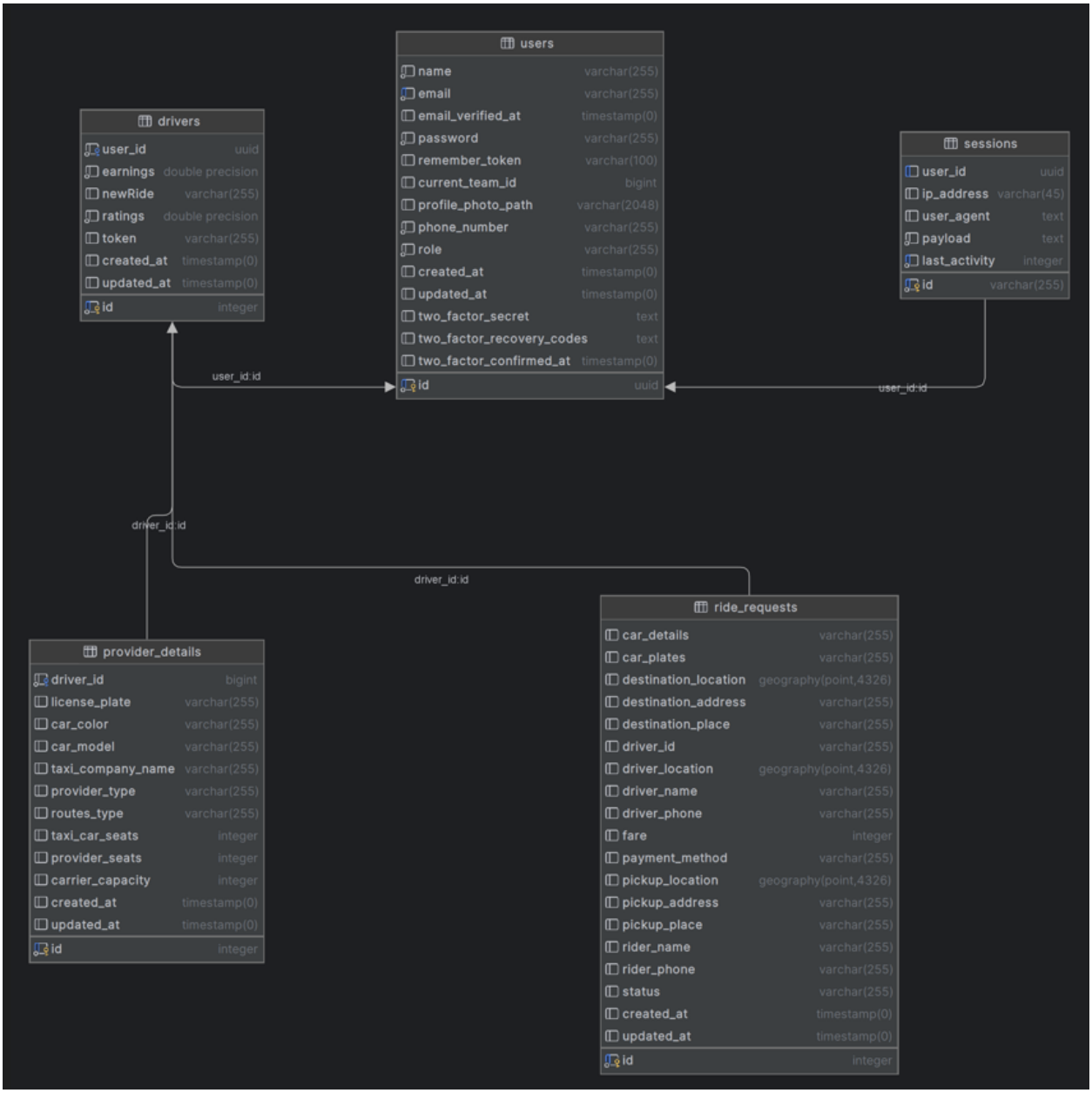
## c. Use Case дијаграми



## d. Дијаграм на активности



## е. Дијаграм на базата на податоци



# 7. Развој на апликацијата

## a. Технологии користени при развојот на апликацијата

Останати технологии кои беа користени при развојот на апликацијата, покрај Flutter и Dart беа:

- Laravel – беше користен за изработка на API за custom backend-от за комуникација на двете апликации преку нудење на endpoint-и на кои што праќаа барања двете апликации и комуницираа со базата на податоци и дополнително за изработка на администраторскиот панел
- PostgreSQL со PostGIS екстензијата – како база на податоци беше користена PostgreSQL база поради тоа што за неа постои PostGIS екстензија која што овозможува работа со географски типови на податоци, како што во случајот беа локациите на превозникот и клиентите, местото на поаѓање и дестинацијата
- Firebase Geofire – библиотека која овозможува зачувување и ажурирање на локацијата во реално време за да може да се следи рутата и локацијата на превозникот
- Firebase Cloud Messaging - провајдер за праќање на нотификации и пораки до достапните превозници

- FlutterFlow – алатка за визуелен развој на апликации која го олеснува креирањето на погледите преку овозможување на drag-and-drop едитори и готови компоненти. Оваа алатка служеше да изработка на некои од погледите во апликацијата и беше од големо олеснување во градењето на погледите преку нудење на готови имплементации на библиотеки за стилизирање на компонентите и widget-ите и utilities.

- Google Maps API – API од страна на Google кое овозможува лесна интеграција на Google Maps во апликации и прилагодување на потребите на апликацијата. Голем дел од Google Maps API множеството беше искористен во развојот на апликацијата, пред се Maps API за приказ на самата мапа, Directions API за одредување и прикажување на рутите, Places API, за претварање на адреси во геометриски локации и обратно, како и за autocomplete при барање на дестинации.

- Chart.js – за прикажување на графиците во администраторскиот панел

- Railway – платформа која нуди инфраструктура и едноставен deployment на апликации со автоматски CI/CD преку линкување на git репозиториум

- Tailwindcss – библиотека користена за изгледот на администраторскиот панел

Дизајнот на делови од апликацијата како и некои програмски сегменти беа инспирирани од курсевите за Flutter & Dart од The Coding Café: Muhammad Ali's Coding Cafe - YouTube

### b. Процес на развој на апликацијата

Платформата TransportEase беше првично развиена за потребите на предметот Напреден Веб Дизајн и во првичната верзија front-end-от беше развиен, но користеше Firebase како backend-as-a-service.

За потребите на овој предмет и за апликацијата да имаме поголема контрола над се што се случува потребно беше да се изработи custom backend. Поради таа причина, го избрав Laravel како технологија за изработка на backend-от.

Најпрво беше потребно да се избере база на податоци која што ќе поддржува работа со географски податоци и поради тоа беше избрана PostgreSQL која што ја има PostGIS екстензијата за работа со географски податоци.

На почетокот на развојот беше направена шемата за базата на податоци, односно, со помош на Laravel Eloquent директно преку код беа поставени ентитетите, нивните атрибути, типовите на атрибутите и релациите кои што ги имаат.

По изработката на шемата, започнав со изработка на сервисите кои што ќе овозможуваат CRUD функционалности за основните ентитети.

Исто така започнав и со изработка на API контролерите кои што праќаа информации до сервисите и ги добиваа промените за да можат да испраќаат JSON објекти како одговор на барањата.

Се обидов API ендпоинтите да следат некаква REST нотација односно со спецификација на типот на барање да се знае за каков тип на операција станува збор.

GET – земање на податоци

POST – креирање на нови податоци

PUT – модифицирање на постоечки податоци

DELETE – за бришење на податоци

Исто така се обидов секој ендпоинт да биде читлив и да може да се заклучи за што станува збор по самото url на ендпоинтот. пр. /api/drivers/updateRating/{id} –

дека станува збор за ендпоинт каде што се ажурира рејтингот на даден возач со некој конкретен id или пак /api/ride-requests/updateStatus/{id} – за ажурирање на статусот на барањето за превоз со некој конкретен id

Како што ги мигрирав функционалностите од Firebase во custom backend-от и додавав функционалности во сервисите на backend-от, правев промени и во front-end-от за да може да се комуницира со бекендот наместо со firebase.

Во front-end-от за да не ги пребришувам веќе постоечките функции одлучив само да креирам wrapper функции со исто име со наставка From/In Backend за да може овие функции да функционираат со било кој бекенд.

Поради тоа си креирав BackendAPIAssistant класа со статички методи кои ги повикувам а во кои што само се испраќаат и примаат податоци и се трансформираат во соодветен формат од некаков ендпоинт. Со ова, со замена на урл-то функциите би функционирале со било кој бекенд сервис.

За изработка на админ панелот, креирав посебен контролер кој комуницира директно со сервисите и влече информации од таму. За проверка на администраторските пермисии додадов middleware кој се повикува само на урлто за пристап на админ панелот /admin.

Најпрво ги развив погледите со Blade и Tailwindcss за стиловите и им испраќам податоци преку контролерот според урлто како што се листата на корисници, барања за превоз...

На крај додадов raw queries во сервисите со кои влечев информации за пополнување на графиците.

Графиците ги додадов преку Chart.js

На крај, за да може да комуницираат и емулаторот и вистински андроид уреди со бекендот, потребно беше тој да се хостира.

Бекендот го хостирав со помош на Railway, кој што нуди платформа за едноставен deployment на сервиси и бази на податоци преку поврзување со git репозиториум.

Во моментот на пишување, бекендот е достапен на следниот урл: TransportEase (railway.app)

### i. Податочни модели

За потребите на апликацијата дефинирани се повеќе податочни модели кои ја олеснуваат работата со податоци и ни претставуваат репрезентација на ентитети од реалниот свет. Некои од податочните модели се специфични само за клиентската апликација, други само за апликацијата за превозници, но има и такви модели кои што се користат и од двете апликации. Исто така дефинирана е класа MethodsAssistants кои се методи дефинирани од мене кои се користат почесто и за подобра видливост се дефинирани како статички методи во оваа класа.

Достапната локација како и сите променливи податоци кои треба да бидат перзистирани беа зачувани преку посебен DataHolder – AppData која претставува custom класа со потребните податоци за апликацијата и се перзистираа тие податоци преку употреба на Provider, кој ни овозможува промена и читање на податоци низ контекстот на целата апликација. За секоја променлива постои метод за ажурирање на вредноста, како што е updateTitle во овој случај и притоа при секоја промена на вредност се известува апликацијата за да може да се рендерира одново layout-от и да биде со променетите вредности.

```
class AppData extends ChangeNotifier {
  Address? pickUpLocation, dropOffLocation;
  DirectionDetails? tripDetails;
  User? loggedInUser;
  AppUser? loggedInUserProfile;
  StreamSubscription<Position>? mainPageStreamSub;
  StreamSubscription<Position>? rideStreamSub;
  AssetsAudioPlayer audioPlayer = AssetsAudioPlayer.newPlayer();
  Position? currentPosition;
  Driver? driverInformation;
  String earnings = "0";
  int numTrips = 0;
  List<String> tripHistoryKeys = [];
  List<TripHistory> tripHistoryData = [];
  String title = "";
  double starCount = 0.0;
  String rideType = "regular";

  void updateTitle(String e) {
    title = e;
    notifyListeners();
  }
```

*Слика 6: DataHolder AppData за глобалните информации кои се користат во двете апликации*

Во прилог се прикажани сите податочни модели кои што ги користи клиентската апликација и се потребни за репрезентација на адреси, места, корисници, насоки, достапни превозници и предвидувања за пребарани локации.

```dart
class Address {
  String placeFormattedAddress;
  String placeName;
  String placeId;
  double latitude;
  double longitude;

  Address(
      {required this.placeFormattedAddress,
      required this.placeName,
      required this.placeId,
      required this.latitude,
      required this.longitude});

  factory Address.fromJson(Map<String, dynamic> json) {
    return Address(
        placeFormattedAddress: json['results'][0]['formatted_address'],
        placeName: json['plus_code']['compound_code'],
        placeId: json['results'][0]['place_id'],
        latitude: json['results'][0]['geometry']['location']['lat'],
        longitude: json['results'][0]['geometry']['location']['lng']);
  }

  @override
  String toString() {
    StringBuffer stringBuffer = new StringBuffer();
    stringBuffer.writeln("Formatted Address: " + placeFormattedAddress);
    stringBuffer.writeln("PlaceID: " + placeId);
    stringBuffer.writeln("PlaceName: " + placeName);
    stringBuffer.writeln("Latitude: " + latitude.toString());
    stringBuffer.writeln("Longitude: " + longitude.toString());
    return stringBuffer.toString();
  }
}
```

```dart
class AppUser {
  String id;
  String email;
  String name;
  String phone;
  String role;

  AppUser(
      {required this.id,
      required this.email,
      required this.name,
      required this.phone,
      required this.role});

  factory AppUser.fromSnapshot(DataSnapshot dataSnapshot) {
    if (dataSnapshot != null) {
      var data = dataSnapshot.value as Map;
      if (data != null) {
        return AppUser(
            id: dataSnapshot.key!,
            email: data["email"],
            name: data["name"],
            phone: data["phone"],
            role: data["role"]);
      }
    }
    throw Exception("AppUser not found");
  }
}
```

*Слики 7 и 8: Модели за Адреса и Корисник на апликацијата*

```dart
class PlacePrediction {
  String secondaryText;
  String mainText;
  String placeId;

  PlacePrediction(
      {required this.secondaryText,
      required this.mainText,
      required this.placeId});

  factory PlacePrediction.fromJson(Map<String, dynamic> json) {
    return PlacePrediction(
        secondaryText: json["structured_formatting"]["secondary_text"],
        mainText: json["structured_formatting"]["main_text"],
        placeId: json["place_id"]);
  }
}
```

```dart
class NearbyAvailableDriver {
  String key;
  double latitude;
  double longitude;
  NearbyAvailableDriver(
      {required this.key, required this.latitude, required this.longitude});
}
```

*Слики 9 и 10: Модели за Пребарана локација и Најблизок превозник*

Податочните модели кои се користат од страна на апликацијата за превозници се прикажани на сликите под овој текст:

```dart
class Driver {
  String name;
  String phone;
  String email;
  String id;
  String car_color;
  String car_model;
  String license_plate;

  Driver(
      {required this.name,
      required this.phone,
      required this.email,
      required this.id,
      required this.car_color,
      required this.car_model,
      required this.license_plate});

  factory Driver.fromSnapshot(DataSnapshot snapshot, String rideType) {
    if (snapshot.value != null) {
      var data = snapshot.value as Map;
      if (rideType.toLowerCase() == "regular") {
        return Driver(
            car_color: data["provider_details"]["car_color"],
            car_model: data["provider_details"]["car_model"],
            license_plate: data["provider_details"]["license_plate"],
            email: data["email"],
            phone: data["phone"],
            id: snapshot.key.toString(),
            name: data["name"]);
      } else if (rideType.toLowerCase() == "taxi") {
        return Driver(
            car_color: data["provider_details"]["taxi_car_seats"],
            car_model: data["provider_details"]["taxi_company_name"],
            license_plate: data["provider_details"]["license_plate"],
            email: data["email"],
            phone: data["phone"],
            id: snapshot.key.toString(),
            name: data["name"]);
      } else {
        return Driver(
            car_color: data["provider_details"]["car_color"],
            car_model: data["provider_details"]["car_model"],
            license_plate: data["provider_details"]["license_plate"],
            email: data["email"],
            phone: data["phone"],
            id: snapshot.key.toString(),
            name: data["name"]);
      }
      throw Exception("Driver not found");
    }
  }
```

```dart
class RideDetails {
  String pickUpAddress;

  String destinationAddress;
  LatLng destinationLocation;
  LatLng pickUpLocation;
  String paymentMethod;
  String rideRequestId;
  String riderName;
  String riderPhone;

  RideDetails(
      {required this.pickUpAddress,
      required this.destinationAddress,
      required this.destinationLocation,
      required this.pickUpLocation,
      required this.paymentMethod,
      required this.rideRequestId,
      required this.riderName,
      required this.riderPhone});
}
```

*Слики 11 и 12: Модели за Возач и Детали за патувањето*

```dart
class TripHistory {
  String paymentMethod;
  String createdOn;
  String status;
  String fare;
  String destination;
  String pickup;

  TripHistory(
      {required this.paymentMethod,
      required this.createdOn,
      required this.status,
      required this.fare,
      required this.destination,
      required this.pickup});

  factory TripHistory.fromSnapshot(DataSnapshot snapshot) {
    if (snapshot.value != null) {
      var data = snapshot.value as Map;
      return TripHistory(
          createdOn: data["created_at"],
          status: data["status"],
          fare: data["fare"] ?? "0",
          paymentMethod: data["payment_method"],
          destination: data["destination_address"],
          pickup: data["pickUp_address"]);
    }
    throw Exception("This trip was not recorded in our database.");
```

*Слика 13: Модел за Конкретно патување*

Потоа беше развиван корисничкиот интерфејс за корисниците, каде што некои од widget-ите беа креирани со помош на FlutterFlow.

API клучевите беа скриени со помош на environment варијабли соодветно и заштитени, односно ограничени за користење со цел да се спречи можна злоупотреба доколку дојде до несакаен приказ на тие клучеви.

### ii. Одредување на тековна локација

За одредување на тековната локација беше користена библиотеката Geolocator. Најпрво се зема логираниот корисник, па се бара пермисија за користење на локацијата на уредот, доколку локацијата е одбиена се обидува уште еднаш да испрати барање и доколку пак е одбиена се испишува порака за грешка. Доколку е прифатена, се зема моменталната локација со висока прецизност и се креираат координати кои се корисат понатаму за земање на тековната локација а и за поместување на камерата на мапата на моменталната локација. Функцијата initGeoFireListener се користи за да се пребаруваат достапните превозници во регион од 5 километри и да се прикажат на мапата.

```
void locatePosition() async {
  MethodsAssistants.getLoggedInUser(context);
  LocationPermission permission;
  permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.deniedForever) {
      return Future.error('Location Not Available');
    }
  }
  permissionGranted();
  Position position = await Geolocator.getCurrentPosition(
      desiredAccuracy: LocationAccuracy.high,
      forceAndroidLocationManager: true);
  currentPosition = position;

  MapsLocation.LatLng latLngPosition =
      MapsLocation.LatLng(position.latitude, position.longitude);

  CameraPosition cameraPosition =
      new CameraPosition(target: latLngPosition, zoom: 14);

  googleMapController
      .animateCamera(CameraUpdate.newCameraPosition(cameraPosition));

  Address address = await MethodsAssistants.getAddressFromCoordinates(
      currentPosition, context);
  initGeoFireListener();
}
```

*Слика 14: Функција за лоцирање на моментална локација*

### iii. Поднесување на барање за превоз до дестинација

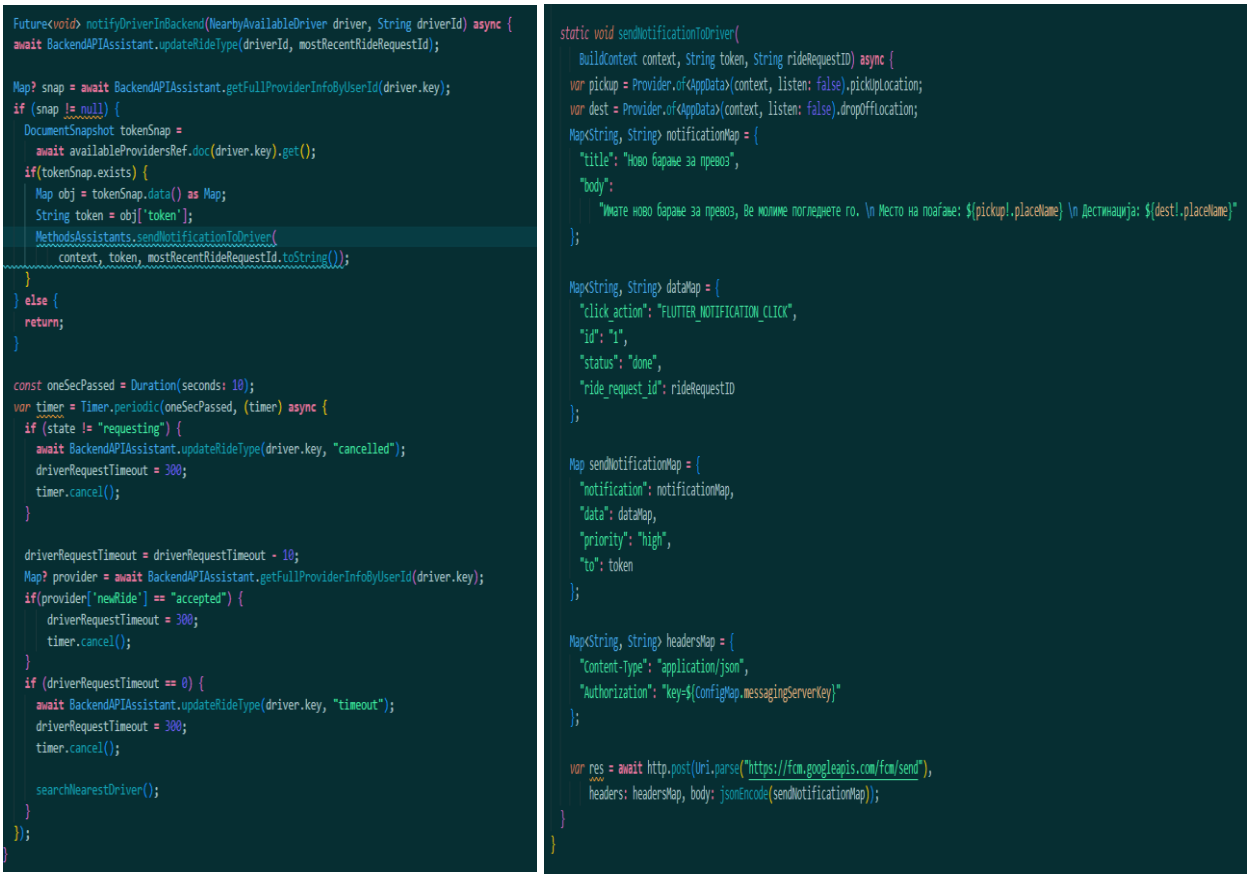
Кога корисникот ќе побара превоз до некоја дестинација, најпрво се одбира најблискиот превозник од соодветниот тип и потоа му се испраќа барање за превоз.

Одбирањето на најблискиот превозник се прави така што се зема првиот превозник од листата на достапни превозници, бидејќи превозниците се додаваат во листата од најблискиот прво и се проверува дали соодветствува на бараниот тип на превозник – дали е такси или обичен превозник. Доколку не е се зема следниот достапен и се така додека да се најде соодветен или доколку нема ниту еден достапен се прикажува порака за грешка.

```
Future<void> searchNearestDriver() async {
  if (availableDrivers.length == 0) {
    showDialog(
        context: context,
        builder: ((context) => AlertDialog(
              content: Text(
                  "Во моментот нема достапни превозници, Ве молиме почекајте."), // Text
            ))); // AlertDialog
    cancelRideRequest();
    resetTrip();
    return;
  } else {
    var nearestDriver = availableDrivers[0];
    var nearestDriverRideType =
        await providersRef.child(nearestDriver.key).child("role").get();
    if (nearestDriverRideType.value != null) {
      String driverType = nearestDriverRideType.value.toString();
      String compareType = rideType + "_driver";
      if (driverType == compareType) {
        notifyDriver(nearestDriver);
        availableDrivers.removeAt(0);
      } else {
        showDialog(
            context: context,
            builder: ((context) => AlertDialog(
                  content: Text(
                      "Во моментот нема достапни превозници од бараниот тип. Ве молиме обидете се повторно или селектирајте друг тип на превозник."), // Text
                ))); // AlertDialog
        availableDrivers.removeAt(0);
        searchNearestDriver();
      }
    } else {
      showDialog(
          context: context,
          builder: ((context) => AlertDialog(
                content: Text(
                    "Во моментот нема достапни превозници. Ве молиме обидете се повторно за неколку минути."), // Text
              ))); // AlertDialog
    }
  }
}
```

*Слика 15: Функција за одбирање на најблизок превозник*

Испраќањето на самото барање за превоз беше имплементирано преку Firebase Cloud Messaging, преку праќање на push нотификација до токенот на соодветниот превозник добиен од firebase и преку слушање на апликацијата за превозници на нотификации на тој конкретен токен. Испраќањето на нотификацијата се прави преку праќање на post барање со соодветната порака и клуч на проектот на Firebase и се испраќа на достапниот сервер на Firebase Cloud Messaging.



```
Future<void> notifyDriverInBackend(NearbyAvailableDriver driver, String driverId) async {
await BackendAPIAssistant.updateRideType(driverId, mostRecentRideRequestId);

Map? snap = await BackendAPIAssistant.getFullProviderInfoByUserId(driver.key);
if (snap != null) {
  DocumentSnapshot tokenSnap =
    await availableProvidersRef.doc(driver.key).get();
  if(tokenSnap.exists) {
    Map obj = tokenSnap.data() as Map;
    String token = obj['token'];
    MethodsAssistants.sendNotificationToDriver(
        context, token, mostRecentRideRequestId.toString());
  }
} else {
  return;
}

const oneSecPassed = Duration(seconds: 10);
var timer = Timer.periodic(oneSecPassed, (timer) async {
  if (state != "requesting") {
    await BackendAPIAssistant.updateRideType(driver.key, "cancelled");
    driverRequestTimeout = 300;
    timer.cancel();
  }

  driverRequestTimeout = driverRequestTimeout - 10;
  Map? provider = await BackendAPIAssistant.getFullProviderInfoByUserId(driver.key);
  if(provider['newRide'] == "accepted") {
      driverRequestTimeout = 300;
      timer.cancel();
  }
  if (driverRequestTimeout == 0) {
    await BackendAPIAssistant.updateRideType(driver.key, "timeout");
    driverRequestTimeout = 300;
    timer.cancel();

    searchNearestDriver();
  }
});
}
```

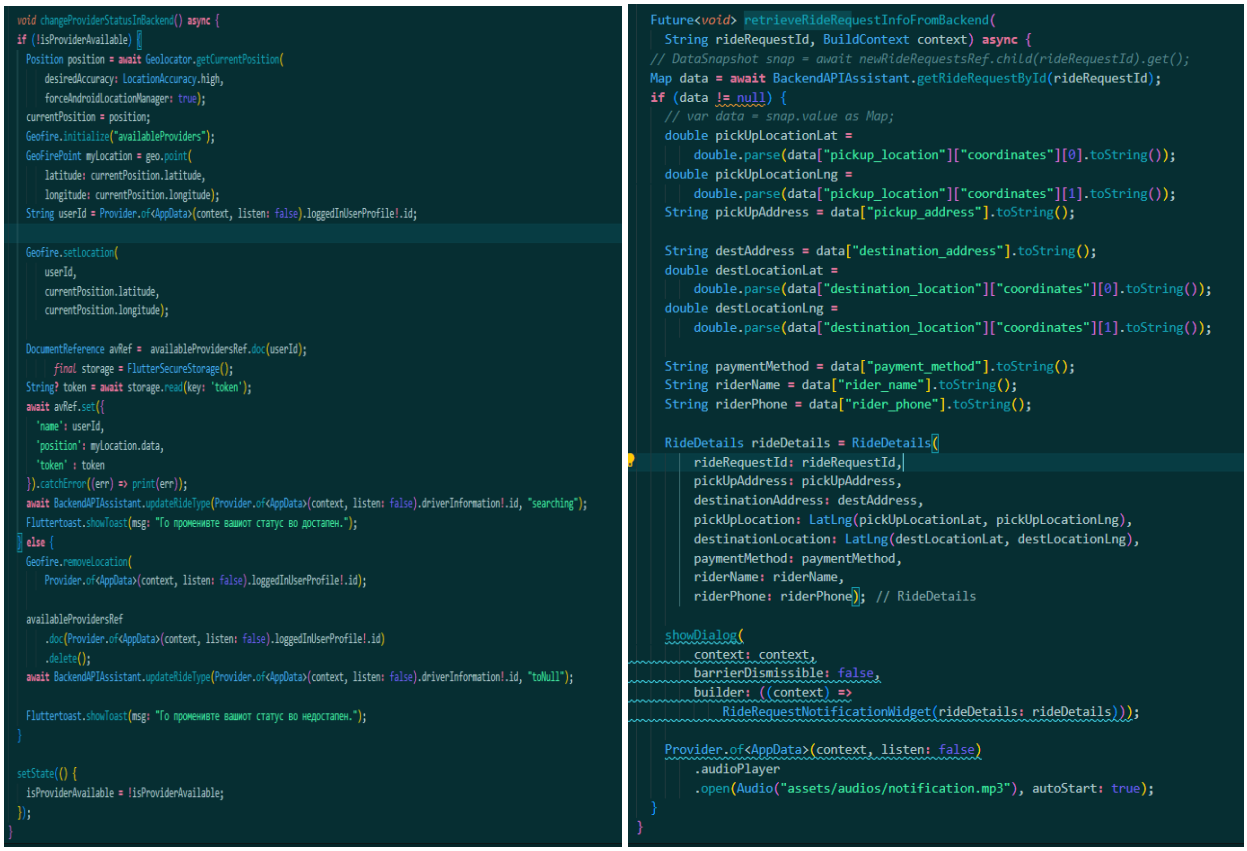
```
static void sendNotificationToDriver(
    BuildContext context, String token, String rideRequestID) async {
  var pickup = Provider.of<AppData>(context, listen: false).pickUpLocation;
  var dest = Provider.of<AppData>(context, listen: false).dropOffLocation;
  Map<String, String> notificationMap = {
    "title": "Ново барање за превоз",
    "body":
        "Имате ново барање за превоз, Ве молиме погледнете го. \n Место на поаѓање: ${pickup!.placeName} \n Дестинација: ${dest!.placeName}"
  };

  Map<String, String> dataMap = {
    "click_action": "FLUTTER_NOTIFICATION_CLICK",
    "id": "1",
    "status": "done",
    "ride_request_id": rideRequestID
  };

  Map sendNotificationMap = {
    "notification": notificationMap,
    "data": dataMap,
    "priority": "high",
    "to": token
  };

  Map<String, String> headersMap = {
    "Content-Type": "application/json",
    "Authorization": "key=${ConfigMap.messagingServerKey}"
  };

  var res = await http.post(Uri.parse("https://fcm.googleapis.com/fcm/send"),
      headers: headersMap, body: jsonEncode(sendNotificationMap));
}
}
```

*Слики 16 и 17: Функции за известување на одбран превозник*

Потоа беше имплементирано барањето на превоз преку зачувување на потребните податоци за патувањето во база на податоци и потоа читање на тие податоци за испраќање на порака до достапните превозници.

Следењето на позицијата на достапните превозници во реално време беше направено со помош на Geofire библиотеката која отвара stream на кој што слуша за било какви промени во локацијата и таа локација ја ажурира во realtime базата на податоци. Бидејќи не постои имплементација на оваа библиотека за Веб, geoflutterfire2 библиотеката се користеше за соодветно овозможување на следење на позицијата на превозниците од страна на клиентите преку зачувување на локацијата во firestore базата и слушање на промените на документите во таа база.

За да може превозник да се појавува во листата на достапни превозници, најпрво треба да го промени својот статус во достапен.



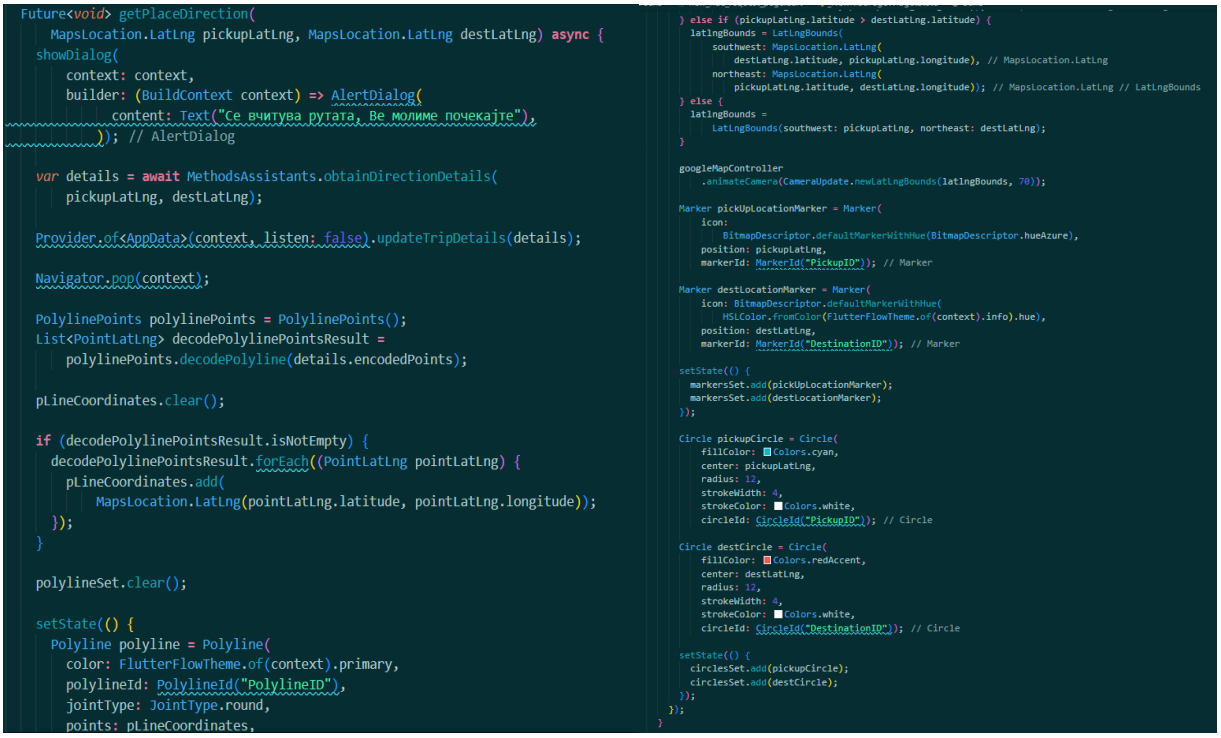
```dart
void changeProviderStatusInBackend() async {
if (!isProviderAvailable) {
  Position position = await Geolocator.getCurrentPosition(
      desiredAccuracy: LocationAccuracy.high,
      forceAndroidLocationManager: true);
  currentPosition = position;
  Geofire.initialize("availableProviders");
  GeoFirePoint myLocation = geo.point(
      latitude: currentPosition.latitude,
      longitude: currentPosition.longitude);
  String userId = Provider.of<AppData>(context, listen: false).loggedInUserProfile!.id;

  Geofire.setLocation(
      userId,
      currentPosition.latitude,
      currentPosition.longitude);

  DocumentReference avRef =  availableProvidersRef.doc(userId);
       final storage = FlutterSecureStorage();
  String? token = await storage.read(key: 'token');
  await avRef.set({
    'name': userId,
    'position': myLocation.data,
    'token' : token
  }).catchError((err) => print(err));
  await BackendAPIAssistant.updateRideType(Provider.of<AppData>(context, listen: false).driverInformation!.id, "searching");
  Fluttertoast.showToast(msg: "Го променивте вашиот статус во достапен.");
} else {
  Geofire.removeLocation(
      Provider.of<AppData>(context, listen: false).loggedInUserProfile!.id);

  availableProvidersRef
      .doc(Provider.of<AppData>(context, listen: false).loggedInUserProfile!.id)
      .delete();
  await BackendAPIAssistant.updateRideType(Provider.of<AppData>(context, listen: false).driverInformation!.id, "toNull");

  Fluttertoast.showToast(msg: "Го променивте вашиот статус во недостапен.");
}

setState(() {
  isProviderAvailable = !isProviderAvailable;
});
}
```

```dart
Future<void> retrieveRideRequestInfoFromBackend(
  String rideRequestId, BuildContext context) async {
// DataSnapshot snap = await newRideRequestsRef.child(rideRequestId).get();
Map data = await BackendAPIAssistant.getRideRequestById(rideRequestId);
if (data != null) {
  // var data = snap.value as Map;
  double pickUpLocationLat =
      double.parse(data["pickup_location"]["coordinates"][0].toString());
  double pickUpLocationLng =
      double.parse(data["pickup_location"]["coordinates"][1].toString());
  String pickUpAddress = data["pickup_address"].toString();

  String destAddress = data["destination_address"].toString();
  double destLocationLat =
      double.parse(data["destination_location"]["coordinates"][0].toString());
  double destLocationLng =
      double.parse(data["destination_location"]["coordinates"][1].toString());

  String paymentMethod = data["payment_method"].toString();
  String riderName = data["rider_name"].toString();
  String riderPhone = data["rider_phone"].toString();

  RideDetails rideDetails = RideDetails(
      rideRequestId: rideRequestId,
      pickUpAddress: pickUpAddress,
      destinationAddress: destAddress,
      pickUpLocation: LatLng(pickUpLocationLat, pickUpLocationLng),
      destinationLocation: LatLng(destLocationLat, destLocationLng),
      paymentMethod: paymentMethod,
      riderName: riderName,
      riderPhone: riderPhone); // RideDetails

  showDialog(
      context: context,
      barrierDismissible: false,
      builder: ((context) =>
          RideRequestNotificationWidget(rideDetails: rideDetails)));

  Provider.of<AppData>(context, listen: false)
      .audioPlayer
      .open(Audio("assets/audios/notification.mp3"), autoStart: true);
}
}
```

*Слики 18 и 19: Функции за промена на статус на достапност и приказ на ново барање кај возач*

По прифаќањето на барањето од страна на превозникот, му се исцртува патека на мапата до локацијата на корисникот од каде што треба да се качи во возилото и да започнат со патувањето, исто така откако ќе започне патувањето кон дестинацијата се ажурира патеката до дестинацијата. Функцијата преку која се имплементира ова е прикажана во слики 20 и 21 и се изведува на тој начин што преку Directions API се земаат сите потребни детали, како и енкодираните точки од рутата кои се претвараат во polyline точки со декодирање и се прикажуваат на мапата преку координатите и додавање на маркери за почетна и крајна дестинација.

Исто така апликацијата за превозници откако превозник ќе стане достапен слуша и чека Firebase Notifications за барање за превоз. При добивање на ваква порака, се преземаат деталите за тоа конкретно барање од база и се прикажуваат. Функцијата преку која е имплементирано ова е прикажано во слика 22.



```
Future<void> getPlaceDirection(
    MapsLocation.LatLng pickupLatLng, MapsLocation.LatLng destLatLng) async {
  showDialog(
      context: context,
      builder: (BuildContext context) => AlertDialog(
            content: Text("Се вчитува рутата, Ве молиме почекајте"),
          )); // AlertDialog

  var details = await MethodsAssistants.obtainDirectionDetails(
      pickupLatLng, destLatLng);

  Provider.of<AppData>(context, listen: false).updateTripDetails(details);

  Navigator.pop(context);

  PolylinePoints polylinePoints = PolylinePoints();
  List<PointLatLng> decodePolylinePointsResult =
      polylinePoints.decodePolyline(details.encodedPoints);

  pLineCoordinates.clear();

  if (decodePolylinePointsResult.isNotEmpty) {
    decodePolylinePointsResult.forEach((PointLatLng pointLatLng) {
      pLineCoordinates.add(
          MapsLocation.LatLng(pointLatLng.latitude, pointLatLng.longitude));
    });
  }

  polylineSet.clear();

  setState(() {
    Polyline polyline = Polyline(
      color: FlutterFlowTheme.of(context).primary,
      polylineId: PolylineId("PolylineID"),
      jointType: JointType.round,
      points: pLineCoordinates,
```

```
  } else if (pickupLatLng.latitude > destLatLng.latitude) {
    latlngBounds = LatLngBounds(
        southwest: MapsLocation.LatLng(
            destLatLng.latitude, pickupLatLng.longitude), // MapsLocation.LatLng
        northeast: MapsLocation.LatLng(
            pickupLatLng.latitude, destLatLng.longitude)); // MapsLocation.LatLng // LatLngBounds
  } else {
    latlngBounds =
        LatLngBounds(southwest: pickupLatLng, northeast: destLatLng);
  }

  googleMapController
      .animateCamera(CameraUpdate.newLatLngBounds(latlngBounds, 70));

  Marker pickUpLocationMarker = Marker(
      icon:
          BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueAzure),
      position: pickupLatLng,
      markerId: MarkerId("PickupID")); // Marker

  Marker destLocationMarker = Marker(
      icon: BitmapDescriptor.defaultMarkerWithHue(
          HSLColor.fromColor(FlutterFlowTheme.of(context).info).hue),
      position: destLatLng,
      markerId: MarkerId("DestinationID")); // Marker

  setState(() {
    markersSet.add(pickUpLocationMarker);
    markersSet.add(destLocationMarker);
  });

  Circle pickupCircle = Circle(
      fillColor: Colors.cyan,
      center: pickupLatLng,
      radius: 12,
      strokeWidth: 4,
      strokeColor: Colors.white,
      circleId: CircleId("PickupID")); // Circle

  Circle destCircle = Circle(
      fillColor: Colors.redAccent,
      center: destLatLng,
      radius: 12,
      strokeWidth: 4,
      strokeColor: Colors.white,
      circleId: CircleId("DestinationID")); // Circle

  setState(() {
    circlesSet.add(pickupCircle);
    circlesSet.add(destCircle);
  });
});
}
```

*Слики 20 и 21: Функции за земање на рута и приказ на рутата на мапа*

```
Future<void> retrieveRideRequestInfoFromBackend(
  String rideRequestId, BuildContext context) async {
// DataSnapshot snap = await newRideRequestsRef.child(rideRequestId).get();
Map data = await BackendAPIAssistant.getRideRequestById(rideRequestId);
if (data != null) {
  // var data = snap.value as Map;
  double pickUpLocationLat =
      double.parse(data["pickup_location"]["coordinates"][0].toString());
  double pickUpLocationLng =
      double.parse(data["pickup_location"]["coordinates"][1].toString());
  String pickUpAddress = data["pickup_address"].toString();

  String destAddress = data["destination_address"].toString();
  double destLocationLat =
      double.parse(data["destination_location"]["coordinates"][0].toString());
  double destLocationLng =
      double.parse(data["destination_location"]["coordinates"][1].toString());

  String paymentMethod = data["payment_method"].toString();
  String riderName = data["rider_name"].toString();
  String riderPhone = data["rider_phone"].toString();

  RideDetails rideDetails = RideDetails(
      rideRequestId: rideRequestId,
      pickUpAddress: pickUpAddress,
      destinationAddress: destAddress,
      pickUpLocation: LatLng(pickUpLocationLat, pickUpLocationLng),
      destinationLocation: LatLng(destLocationLat, destLocationLng),
      paymentMethod: paymentMethod,
      riderName: riderName,
      riderPhone: riderPhone); // RideDetails

  showDialog(
      context: context,
      barrierDismissible: false,
      builder: ((context) =>
          RideRequestNotificationWidget(rideDetails: rideDetails)));

  Provider.of<AppData>(context, listen: false)
      .audioPlayer
      .open(Audio("assets/audios/notification.mp3"), autoStart: true);
}
}
```

*Слика 22: Функција за примање на барање за превоз и земање на детали за тоа барање*

По пристигањето на дестинацијата, превозникот го завршува патувањето со клик на копче во апликацијата и се зачувува ова патување во неговата историја и се пресметува цената според типот на превозник која треба да ја плати корисникот пришто му се префрлаат средствата на превозникот. Средстата се пресметуваат на начин што за такси превозниците е поефтино временски, а поскапо по километража и обратно за обичните превозници.

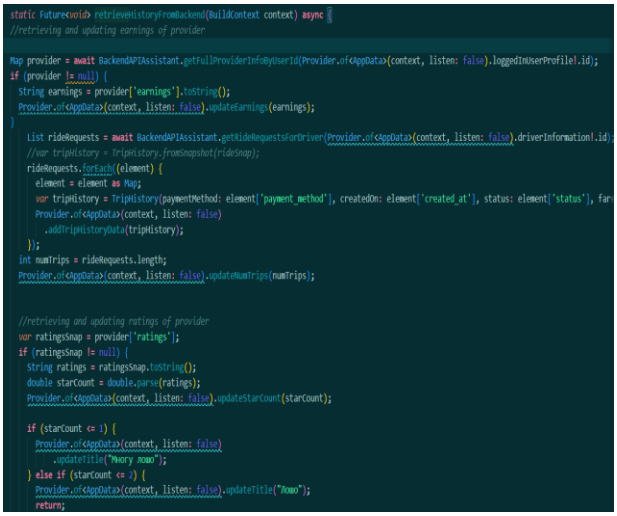
```
Future<void> endTrip() async {
  timer!.cancel();
  showDialog(
      context: context,
      barrierDismissible: false,
      builder: ((context) =>
          AlertDialog(content: Text("Ве молиме почекајте"))));

  var currentLatLng =
      MapsLocation.LatLng(myPosition.latitude, myPosition.longitude);

  var destDetails = await MethodsAssistants.obtainDirectionDetails(
      widget.rideDetails.pickUpLocation, currentLatLng);

  Navigator.pop(context);

  int fareAmount = MethodsAssistants.calculateFare(
      destDetails, Provider.of<AppData>(context, listen: false).rideType);

  status = "finished";

  // newRideRequestsRef
  //     .child(widget.rideDetails.rideRequestId)
  //     .child("fare")
  //     .set(fareAmount.toString());
  await BackendAPIAssistant.updateRideRequestFare(widget.rideDetails.rideRequestId, fareAmount);
  await BackendAPIAssistant.updateRideRequestStatus(widget.rideDetails.rideRequestId, status);

  // newRideRequestsRef
  //     .child(widget.rideDetails.rideRequestId)
  //     .child("status")
  //     .set(status);

  Provider.of<AppData>(context, listen: false).cancelRideSub();

  showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => FareDialogWidget(
          fareAmount: fareAmount,
          paymentMethod: widget.rideDetails.paymentMethod));

  //saveEarnings(fareAmount);
  saveEarningsInBackend(fareAmount);
}
```

*Слика 23: Функција за завршување на патување и зачувување на истото*

```
static int calculateFare(DirectionDetails directionDetails, String rideType) {
  //in USD
  double timeTravelled = 0.0;
  double distanceTravelled = 0.0;
  if (rideType.toLowerCase() == "taxi") {
    timeTravelled = (directionDetails.durationValue / 60) * 0.15;
    distanceTravelled = (directionDetails.distanceValue / 1000) * 0.30;
  } else if (rideType.toLowerCase() == "regular") {
    timeTravelled = (directionDetails.durationValue / 60) * 0.20;
    distanceTravelled = (directionDetails.distanceValue / 1000) * 0.20;
  }
  double fareUsd = timeTravelled + distanceTravelled;

  //in MKD, at time of writing this function 1$ USD = 57 MKD
  double fareMkd = fareUsd * 57;
  return fareMkd.round();
}
```

*Слика 24: Функција за пресметување на износ за патување*

Секој превозник може да ја види историјата на патувања кои ги има направено како и деталите за нив, кога се завршени, од која до која локација е патувањето, кој патник и колкава е цената на патувањето. Имплементирано е на тој начин што се земаат клучевите, односно идентификациските броеви на патувањата од историјата на превозникот и преку нив се пребарува базата за да се преземат сите потребни детали. Во истата функција се презема и оценката на превозникот.



```
static Future<void> retrieveHistoryFromBackend(BuildContext context) async {
//retrieving and updating earnings of provider

Map provider = await BackendAPIAssistant.getFullProviderInfoByUserId(Provider.of<AppData>(context, listen: false).loggedInUserProfile!.id);
if (provider != null) {
  String earnings = provider['earnings'].toString();
  Provider.of<AppData>(context, listen: false).updateEarnings(earnings);
}
    List rideRequests = await BackendAPIAssistant.getRideRequestsForDriver(Provider.of<AppData>(context, listen: false).driverInformation!.id);
    //var tripHistory = TripHistory.fromSnapshot(rideSnap);
    rideRequests.forEach((element) {
      element = element as Map;
      var tripHistory = TripHistory(paymentMethod: element['payment_method'], createdOn: element['created_at'], status: element['status'], far
      Provider.of<AppData>(context, listen: false)
          .addTripHistoryData(tripHistory);
    });
  int numTrips = rideRequests.length;
  Provider.of<AppData>(context, listen: false).updateNumTrips(numTrips);


  //retrieving and updating ratings of provider
  var ratingsSnap = provider['ratings'];
  if (ratingsSnap != null) {
    String ratings = ratingsSnap.toString();
    double starCount = double.parse(ratings);
    Provider.of<AppData>(context, listen: false).updateStarCount(starCount);

    if (starCount <= 1) {
      Provider.of<AppData>(context, listen: false)
          .updateTitle("Многу лошо");
    } else if (starCount <= 2) {
      Provider.of<AppData>(context, listen: false).updateTitle("Лошо");
      return;
```

```
    } else if (starCount <= 2) {
      Provider.of<AppData>(context, listen: false).updateTitle("Лошо");
      return;
    } else if (starCount <= 3.5) {
      Provider.of<AppData>(context, listen: false).updateTitle("Добро");

      return;
    } else if (starCount <= 4.5) {
      Provider.of<AppData>(context, listen: false)
          .updateTitle("Многу добро");

      return;
    } else if (starCount <= 5) {
      Provider.of<AppData>(context, listen: false).updateTitle("Одлично");

      return;
    }
  } else {
    String ratings = "0.0";
    double starCount = double.parse(ratings);
    Provider.of<AppData>(context, listen: false).updateStarCount(starCount);
    Provider.of<AppData>(context, listen: false).updateTitle("Неоценет");
  }
 }
}
```

*Слики 25 и 26: Функција за преземање на историја на патувања и оцена на превозник*

```
static Future<void> obtainTripHistoryData(BuildContext context) async {
  var keys = Provider.of<AppData>(context, listen: false).tripHistoryKeys;

  for (String key in keys) {
    var rideSnap = await newRideRequestsRef.child(key).get();
    if (rideSnap.exists && rideSnap.value != null) {
      var tripHistory = TripHistory.fromSnapshot(rideSnap);
      Provider.of<AppData>(context, listen: false)
          .addTripHistoryData(tripHistory);
    }
  }
}

static void clearTripHistory(context) {
  Provider.of<AppData>(context, listen: false).updateNumTrips(0);
  Provider.of<AppData>(context, listen: false).tripHistoryData.clear();
  Provider.of<AppData>(context, listen: false).tripHistoryKeys.clear();
}

static String formatDateAsString(String date) {
  DateTime dateTime = DateTime.parse(date);
  String formattedDate =
      "${DateFormat.MMMd().format(dateTime)}, ${DateFormat.y().format(dateTime)} - ${DateFormat.jm().format(dateTime)}";
  return formattedDate;
 }
}
```

*Слики 27 и 28: Функција за преземање на детали на патувања и чистење на историја*

Во овој извештај се прикажани само најважните функции и начин на имплементација на најважните функционалности. За повеќе детали околу начинот на имплементација на останатите функционалности достапен е целиот изворен код на линковите на репозиториумите:

Линк до GitLab репозиториум: https://gitlab.finki.ukim.mk/ioss/Transport-Ease

Линк до алтернативен GitLab: https://gitlab.finki.ukim.mk/201042/transportease

Линк до GitHub репозиториум: https://github.com/gorgilazarev3/Transport-Ease/

# 8. Заклучок и можности за дополнување на апликацијата

Flutter и Dart се моќни алатки за развој на апликации кои претставуваат иновација во светот на програмирањето. Со Flutter, програмерите добиваат крос-платформска рамка за развој на апликации кои се одликуваат со одлични перформанси и современ дизајн, а пак Dart, како програмски јазик, им обезбедува ефикасност и лесна читливост на кодот. Заедно, Flutter и Dart им нудат на програмерите моќни алатки за креирање на апликации кои можат да ги задоволат потребите на модерните корисници и кои имаат можност да создаваат иновативни и функционални апликации кои работат на повеќе платформи и им овозможуваат на корисниците да се поврзат и интеракционираат со содржината на еден брз и ефикасен начин. Flutter и Dart продолжуваат да растат во популарност и да оставуваат значителен влијание во областа на мобилниот развој, и се очекуваат да продолжат да иновираат и внесуваат новости во иднината.

Laravel стана вистински избор за развивачите на PHP поради неговата експресивна синтакса, робусните карактеристики и активната заедница. Им овозможува на програмерите ефикасно да градат скалабилни, одржливи веб-апликации. Без разлика дали сте искусен програмер или штотуку почнувате со развој на веб апликации, Laravel може да отвори возбудливи можности за вашите проекти.

Додека пак, платформа развиена како дел од оваа проектна задача може да биде проширена со повеќе функционалности и да биде користена не само како платформа за превоз, туку како и платформа за изведување туристички екскурзии и организирани патувања, да им овозможува на корисниците брза и лесна комуникација со карго компаниите за достава на добра и можности за проширување на апликациите надвор од границите на Република Македонија.