Primjena Flutter: Darta i izrada mobilne aplikacije

Dragić, Daniel

VELEUČILIŠTE
s pravom javnosti
BALTAZAR ZAPREŠIĆ
Zaprešić

Stručni prijediplomski studij
Informacijske tehnologije

DANIEL DRAGIĆ

PRIMJENA FLUTTER: DARTA I IZRADA MOBILNE APLIKACIJE

ZAVRŠNI RAD

Zaprešić, 2024. godine

VELEUČILIŠTE
s pravom javnosti
BALTAZAR ZAPREŠIĆ
Zaprešić

Stručni prijediplomski studij
Informacijske tehnologije

ZAVRŠNI RAD

PRIMJENA FLUTTER: DARTA I IZRADA MOBILNE APLIKACIJE

Mentor:
doc. dr. sc. Matija Varga

Student:
Daniel Dragić

Naziv kolegija:
WEB PROGRAMIRANJE I MOBILNE APLIKACIJE

JMBAG studenta:
0307014130

SADRŽAJ

SAŽETAK

Prilikom izrade mobilne aplikacije nailazi se na ogroman broj čimbenika, koji direktno ili indirektno utječu na funkcionalnost mobilne aplikacije. Iako u današnje vrijeme imamo širok spektar prilikom odabira programskih jezika za izradu mobilne aplikacije, u ovom radu sam se odlučio za Flutter: Dart kombinaciju. Tijekom studiranja, paralelno sam učio programski jezik Dart koji u kombinaciji sa Flutter-om omogućava izradu aplikacija na više platformi, iako sam na studiju učio druge programske jezike, većinu znanja i tehnika mogu primijeniti u Flutter: Dart mobilnoj aplikaciji. Tema je odabrana, jer upravo ovaj rad predstavlja sve što sam naučio prilikom studiranja i samostalnim učenjem i istraživanjem. Očekivani doprinos ovog završnog rada je popularizacija Flutter softverskog paketa i Dart objektno orijentiranog programskog jezika, kako bi se dodatno približio u količini korištenja i konkurirao drugim više rasprostranjenim programskim jezicima za izradu mobilnih aplikacija poput React Native-a i Swift programskog jezika razvijenog od strane Apple-a. Prilikom pisanja rada detaljno sam opisao samu instalaciju Flutter SDK-a, instalaciju Microsoft Visual Studio Code-a, izrada mobilne aplikacije, te važnost state managementa u aplikaciji i svrhu programskog sučelja aplikacije (eng. Application Programming Interface, API)

Ključne riječi: Flutter, Dart, programski jezik, mobilna aplikacija, state management, API

USE OF FLUTTER: DART AND MOBILE APPLICATION DEVELOPMENT

ABSTRACT

Huge number of factors are encountered upon creating a mobile application, which can affect directly or indirectly the functionality of the mobile application. Altough nowadays we have a wide spectrum when choosing a programming language for creating a mobile application, in this paper I decided on the Flutter: Dart combination. During my studies, I studied the Dart programming language in parallel, which in combination with Flutter enables the creation of applications on multiple platforms, although I studied different programming languages during my studies, I can apply most of the knowledge and techniques in Flutter: Dart mobile application. The topic was chosen because this work represents everything I learned during my studies and through independent study and research. The expected contribution of this final paper is the popularization of the Flutter SDK and the Dart object-oriented programming language, in order to further approach in the amount of use and compete with other more widespread programming languages for creating mobile applications such as React Native and the Swift programming language developed by Apple. When writing the paper, I described in detail the installation of the Flutter SDK, the installation of Microsoft Visual Studio Code, the creation of a mobile application, the importance of state management in the application and the purpose of the Application Programming Interface (API)

Key words: Flutter SDK, Dart, programming language, mobile application, state management, API

1. UVOD

Prije pojave framework-a za razvoj aplikacija na više platformi[1], organizacije su morale izraditi mobilne aplikacije zapošljavajući poseban tim ljudi za izradu aplikacija na web platformi te posebno tim stručnjaka za iOS i Android platformu. Drugim riječima se zovu Native aplikacije[2], odnosno aplikacije koje su lojalne određenom operacijskom sustavu. Nativne aplikacije kreirane za Android uređaje, ne rade na iOS uređajima i obrnuto. Ukoliko je potrebna izrada mobilne aplikacije za obje platforme, neophodna je izrada više verzija iste aplikacije, koja zahtjeva dodatne napore kodiranja, što rezultira većim troškovima izrade mobilne aplikacije. Ovaj je problem riješio Google 2017. godine izradom Flutter-a. Za izradu aplikacija sa Flutter-om, potrebno je koristiti također Google-ov programski jezik Dart. Neke glavne značajke Flutter-a su:

1. „Hot reload“ – primjenjuje promjene napravljene u kodu te prikazuje na aplikaciji bez potrebe ponovnog kompajliranja[3]
2. „Google's Material Design“ – prilikom izrade aplikacije može se koristiti više vizualnih i bihevioralnih widgeta
3. Flutter se ne oslanja na tehnologiju web preglednika, već ima vlastiti mehanizam za iscrtavanje widgeta

Primjere korištenja Flutter framework-a vidimo u aplikacijama poput eBay-a, Alibab-e, Google Pay-a i raznih drugih. Završni rad je podijeljen u 10 poglavlja. U početku se govori o instalaciji Flutter framework-a u Windows okruženju sa svim koracima, zatim o arhitekturi mobilne aplikacije, potom o GitHub[4] REST API-ju, State Managementu[5], i na kraju o samom dizajnu odnosno izgledu aplikacije.

[1] Cross-platform – višeplatformski softver dizajniran za rad na više operacijskih sustava

[2] Nativna aplikacija – izvorni softver aplikacije dizajniran za rad na određenom operacijskom sustavu

[3] Kompajliranje – prevođenje napisanog koda u strojno čitljiv kod

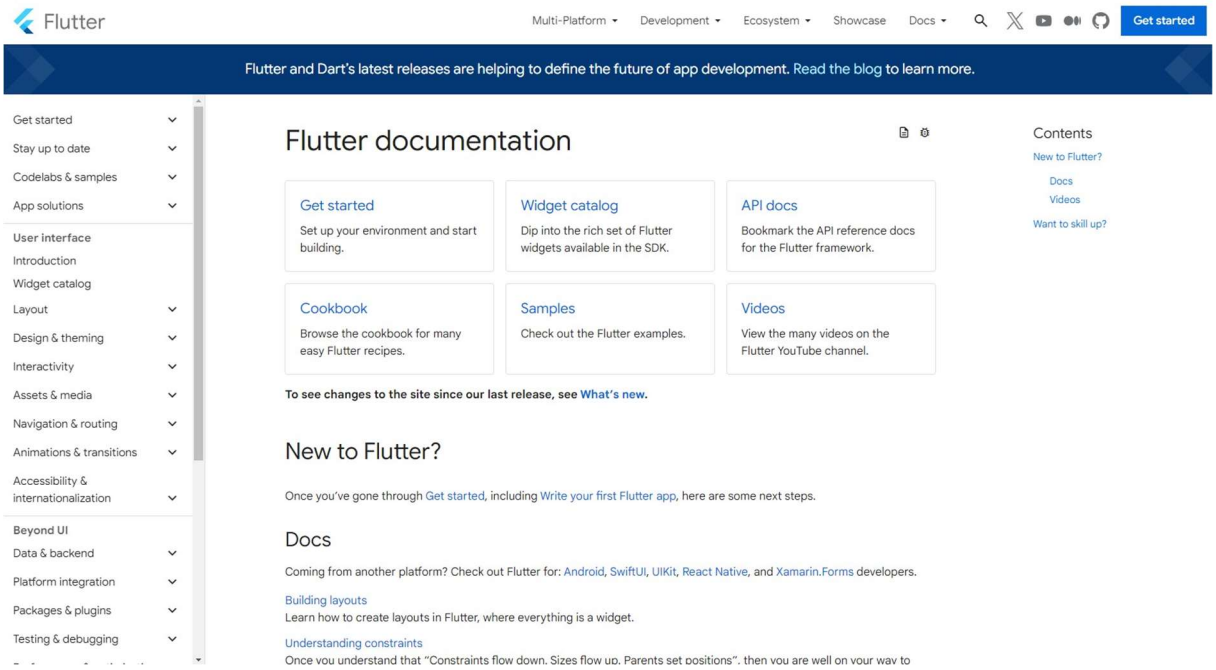
[4] GitHub – razvojna platforma koja omogućuje stvaranje, pohranu, upravljanje i dijeljenje koda

[5] State management – upravljanje stanjem aplikacije

2. INSTALACIJA FLUTTER SDK U WINDOWS OKRUŽENJU

Prvi korak u izradi završnog rada bio je instalacija Flutter programskog seta razvojnih alata ili (eng. Software development kit, SDK). Budući da je sama instalacija kompleksna, u nastavku je podijeljena u par koraka i detaljnije objašnjena uz korištenje slika i opisa

Prvi korak: zahtjeva odlazak na službenu stranicu Flutter-a, odnosno na https://docs.flutter.dev/, prikazan na Slika 1.



Slika 1 Službena stranica Flutter-a

Izvor: Službena Flutter stranica

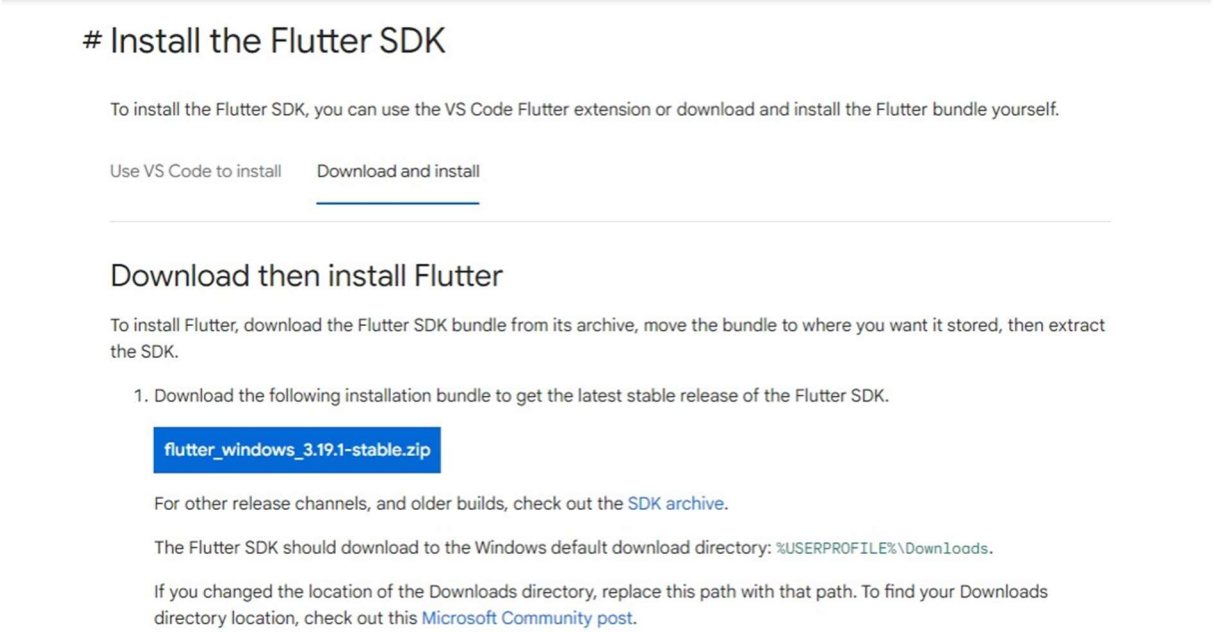
Drugi korak: u instalaciji zahtjeva odabir „Get started“ ikone u gornjem desnom kutu.

Treći korak: potrebno je odabrati razvojnu platformu koja će se koristiti za instalaciju Flutter SDK-a. U ovom završnom radu korištena je Windows razvojna platforma.

Četvrti korak: budući da Flutter omogućava izradu aplikacija za mobilne uređaje, web i stolna računala, nije važno koja je od tri ponuđene opcije odabrana.

Peti korak: ovisno o odabiru u prethodnom koraku prati se slijed instrukcija i zahtjeva Flutter SDK-a u vidu hardvera i softvera.

Šesti korak: naknadno se preuzima ZIP verziju Flutter-a, klikom na plavi okvir prikazan na Slika 2.

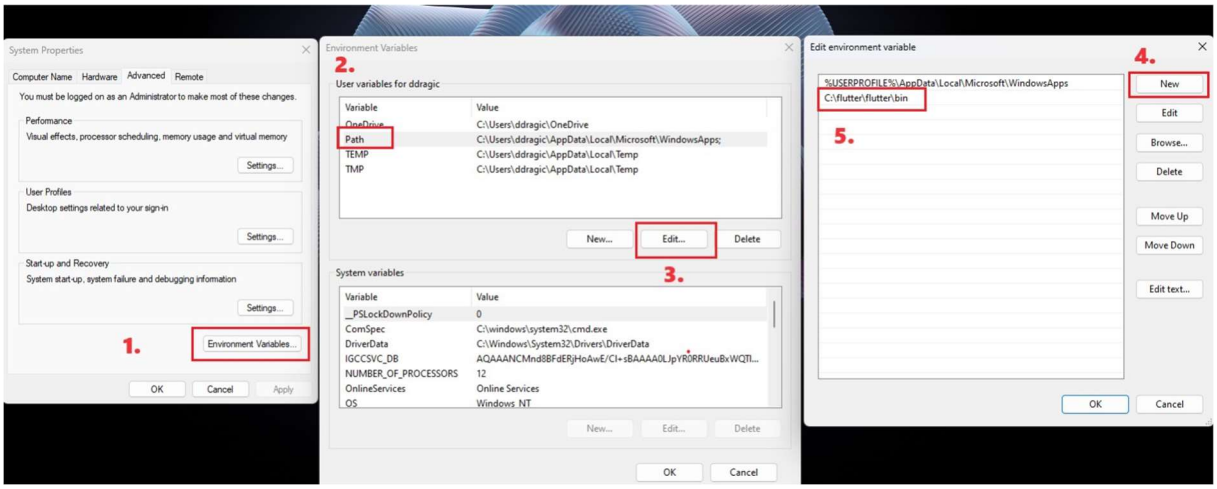


Slika 2 Preuzimanje Flutter-a

Izvor: Službena Flutter stranica

Sedmi korak: potrebno je izdvojiti iz preuzete ZIP datoteke, u mapu koja je odabrana.

Osmi korak: dodavanje Flutter-a kao varijablu u okruženje Windows sustava, na način da se kopira točna adresa podmape „bin“ iz prethodno izdvojene datoteke, te se zalijepi u putanju Edit the system environment variables>Environment Variables>Path>Edit>New. Prikazano na Slika 3,

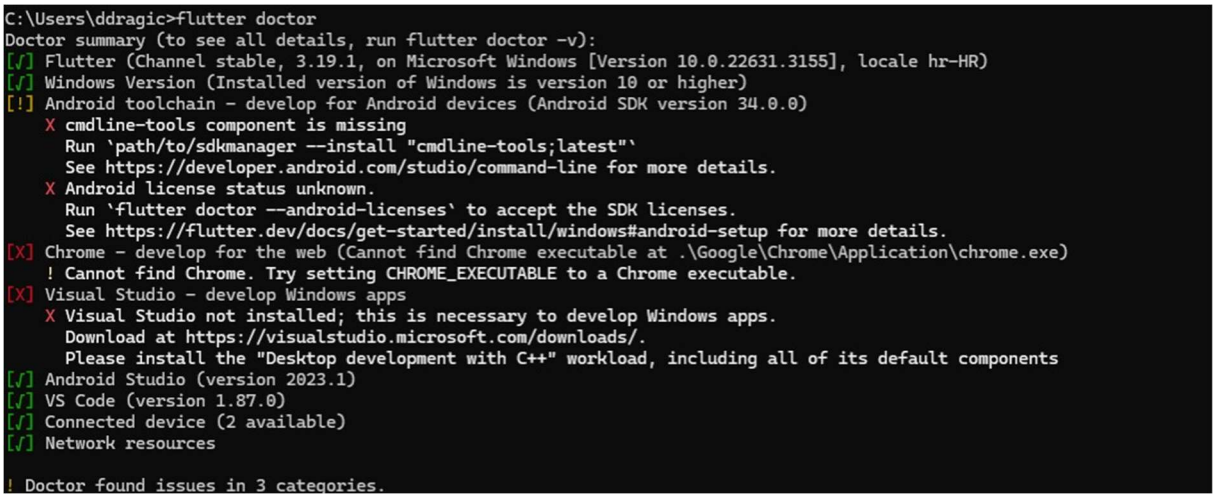


Slika 3 Dodavanje Fluttera kao varijablu u okruženje Windows sustava

Izvor: Izrada autora

Deveti korak: Preuzimanje i instalacija Android Studio programa sa službene stranice https://developer.android.com/studio

Deseti korak: Pokretanje Command Prompt-a (Run as administrator), i provjera funkcionalnosti Flutter-a, tako da se upiše naredbu flutter doctor, kao na Slika 4.



Slika 4 Naredba flutter doctor

Izvor: Izrada autora

Zadnjim korakom (deseti korak) gdje je provjerena funkcionalnost Flutter-a završava instalacija programa i sve potrebne pripreme za izradu mobilnih ili web aplikacija.

3. MOBILNA APLIKACIJA

Završni rad temeljen je na ideji izrade mobilne aplikacije sa svrhom pretraživanja svih repozitorija na GitHub stranici. Unutar projekta implementiran je Github REST API, kojim će se rad detaljnije baviti, razne dodatke razvijene od trećih strana odnosno eng. dependencies/plugins koji se definiraju u pubspec.yaml datoteci unutar aplikacije. Što se tiče arhitekture samog projekta implementirana je MVC arhitektura odnosno Model-View-Controller Više o njemu se može pročitati u poglavlju 3.1 ARHITEKTURA APLIKACIJE.. Za upravljanje stanjem aplikacije (eng. state management) korišten je Provider koji je poprilično apstraktan. Također potrebno je izdvojiti nekoliko paketa (eng. package) poput http koji upravlja povezivanjem aplikacije sa internetom, cached_network_image za prikaz slika sa interneta, url_launcher za otvaranje eksternih linkova unutar aplikacije. Osim pretraživanja samih repozitorija, omogućen je pregled korisničkih računa koji su izradili i objavili iste repozitorije, te značajka sortiranja liste abecedno ili po broju zvjezdica dobivenih na stranici. Ugledajući se na trenutno popularne mobilne aplikacije poput Instagram-a ili Facebook-a, implementiran je tzv. infinite loop, odnosno ukoliko se dođe do zadnja 3 repozitorija na listi, automatski učitava idućih 20 repozitorija koji odgovaraju unesenom opisu. Ovaj opis odgovara određenom terminu dinamičke liste. Potrebno je naglasiti, da su osjetljivi podaci poput ključeva odvojeni od datoteka predviđene za izgled stranice.

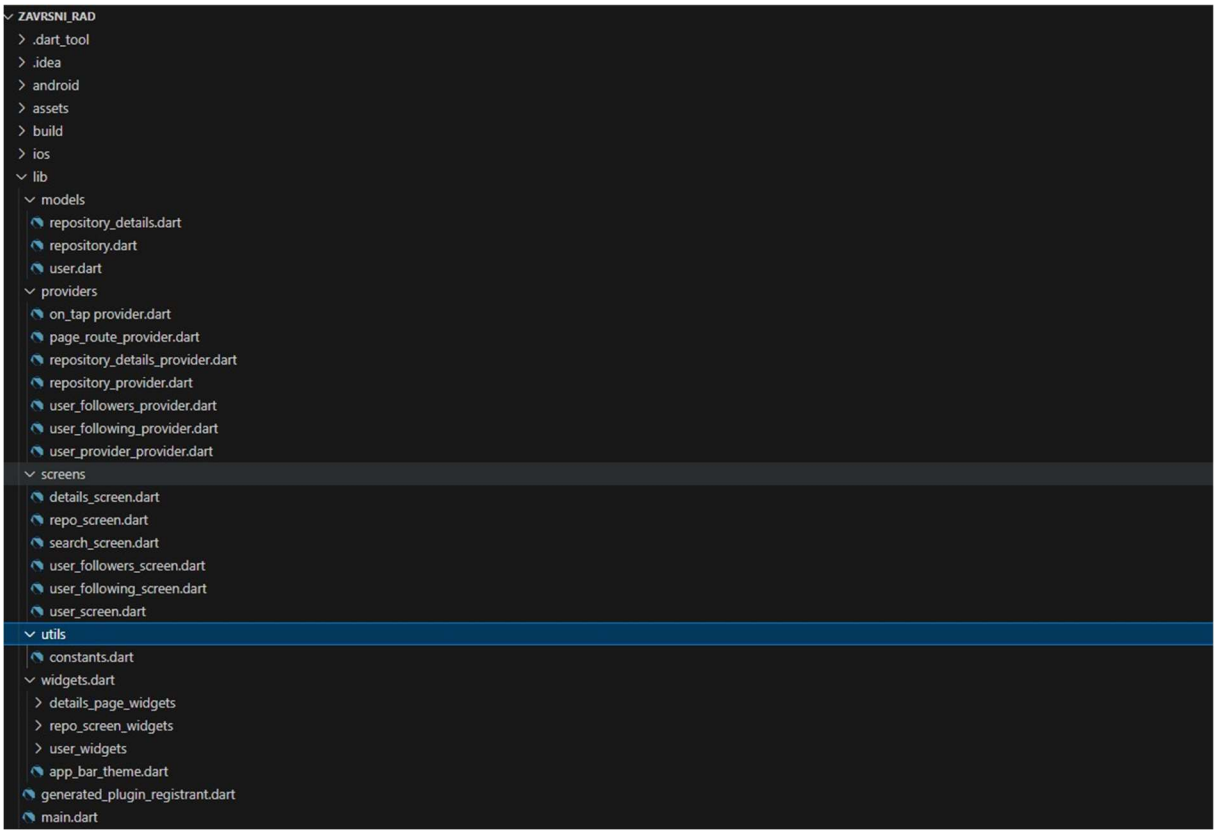
3.1 ARHITEKTURA APLIKACIJE

Prethodno rečeno korišten je Model-View-Controller, MVC. Unutar ovog projekta umjesto Controller-a, korišten je Provider, koji u pozadini funkcionira isto kao i kontroler u drugim programskim jezicima.

> Prema autorima Iqbal H. Sarker i K.Apu, „MVC razvojni okvir je široko postao standard u modernom razvoju softvera. Primjenom ponovne uporabe odgovarajućeg koda u sloju modela, view-a i controllera ne samo da može dovesti do odvajanja temeljne poslovne logike, kontrole procesa, te zaslona aplikacije, nego i značajno poboljšati skalabilnost i održavanje softvera. Ponovna uporaba koda je jedna od čestih formi u objektno orijentirano ponovnom korištenju (eng. reusability)“[6]

[6] Iqbal H. Sarker and K. Apu (2014.), MVC Architecture Driven Design and Implementation of Java Framework for Developing Desktop Application

Prilikom izrade arhitekture, bitno je s vremena na vrijeme pregledati arhitekturu, te ju optimizirati kako bi i druge osobe shvatile pojedine datoteke i logiku u projektu. Ukoliko se arhitektura ne posloži po standardima, nakon nekog vremena, moglo bi doći do toga da se sam autor projekta u njemu ne snalazi. Iako je ovaj rad i projekt samostalno izrađen, u poslovnom svijetu i većim tvrtkama i organizacijama radi se u timovima, te zbog toga postoje određeni standardi propisani kojih se treba pridržavati. Slika 5. prikazuje primjenu MVC uzorka u ovom projektu



Slika 5 MVC uzorak u projektu

Izvor 1 : Izrada autora

4. GITHUB REST API

4.1 REST API

Laički objašnjeno REST API je sustav koji se koristi za izmjenu podataka i informacija između sustava, kako računalnih tako i mobilnih preko interneta.

Ukoliko prevedemo IBM-ovu teoriju o REST API-jima, ona glasi da je REST API ili također zvan RESTful API ili RESTful web API sučelje za programiranje aplikacija koji je u skladu s načelima dizajna arhitektonskog stila prijenosa reprezentativnog stanja (REST). REST API-ji nam omogućuju ogromnu fleksibilnost, lakši način integracije aplikacija i povezivanje različitih komponenti u arhitekturama mikroservisa. [7]

> Prema istraživanju koje je proveo Varga (2022: 11-13): „REST protokol je trenutno najbolje rješenje za komunikaciju između tehnologija. REST je nastao iz WWW (World Wide Web) tehnologije tj. usluge i to uvođenjem određenih ograničenja. Ova ograničenja čine osnovne principe REST modela (Slika 6) koji određuju kako su resursi na globalnoj mreži Internet mogu koristiti

Slika 6 RESTful Web Service

Izvor : Izrada autora rada na temelju izvora: Ramesh Dilshan. RESTful Web services. URL: https://medium.com/@dilshanramesh81/restful-web-services-35a96401b42b. (16.11.2021.) (Ramesh, 2021.).

> Slika 6 prikazuje RESTful Web usluge koje obuhvaćaju aplikacije kreirane u programskim jezicima poput: (1) jave, (2) pHp-a, (3) C programskog jezika te *.net-a. Osim načela RESTa, kako svaki resurs ima jedinstveni indentifikator postoje još sljedeća načela poput:

[7] Službena stranica International Business Machines, IBM

(1) međusobnog povezivanja resursa, (2) uporabe standardnih metoda, (3) resursi s višestrukim reprezentacijama i (4) komunikacija bez održavanja stanja.“[8]

4.2 Krajnje točke (eng. endpoints)

API krajnja točka ili endpoint je URL(Uniform Resource Locator) koji se ponaša kao kontaktna točka između API klijenta i API servera. API klijenti šalju zahtjeve prema API krajnjim točkama da bi pristupili funkcionalnosti API-ja i podataka unutar istog. Tipični REST API ima više krajnjih točaka koji odgovaraju njegovim raspoloživim resursima. Na primjer, API koji pokreće aplikaciju društvenih medija vjerojatno bi uključivao krajnje točke za korisnike, postove i komentare. Zahtjevi prema krajnjim točkama API-ja moraju sadržavati metodu koja ukazuje na operaciju koju treba izvesti, kao i potrebna zaglavlja, parametre, vjerodajnice za provjeru autentičnosti i podatke o tijelu.[9]

Unutar završnog rada korištene su navedene krajnje točke za dohvaćanje podataka omogućene od strane Github-a, za: Listu svih javnih repozitorija i Korisnike.

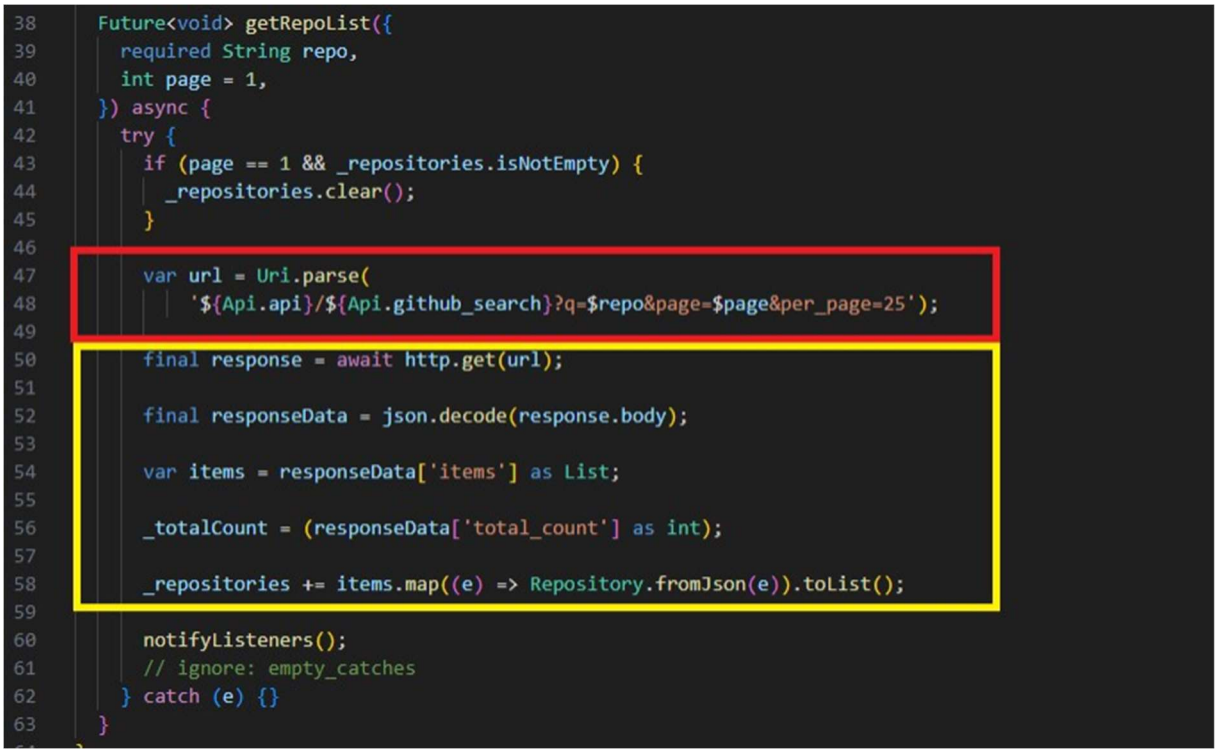
4.2.1 Lista svih javnih repozitorija

Podatke iz ove krajnje točke dobiju u JSON (JavaScript Object Notation) formatu prikazanom na Slika 7. Cijela dokumentacija je dostupna na službenoj stranici Github-a, https://docs.github.com/en/rest/repos/repos?apiVersion=2022-11-28#list-public-repositories

Unutar projekta pristupljeno je listi svih javnih repozitorija prikazan na Slika 7, gdje je unutar crvenog okvira prikazana krajnja točka, odnosno jedan od načina na koji se preko HTTP paketa pristupa krajnjoj točki. U žutom okviru je također prikazan jedan od načina spremanja dobivenih podataka iz JSON formata u model Repository-a (Slika 8).

[8] Znanje o modernim okruženjima za razvoj WEB usluga s osvrtom na SOAP protokol i Angular kod učenika četvrtih razreda smjera web dizajner i medijski tehničar, Matija Varga (2022)

[9] Što je API krajnja točka? Postman (2023)



```dart
Future<void> getRepoList({
  required String repo,
  int page = 1,
}) async {
  try {
    if (page == 1 && _repositories.isNotEmpty) {
      _repositories.clear();
    }

    var url = Uri.parse(
        '${Api.api}/${Api.github_search}?q=$repo&page=$page&per_page=25');

    final response = await http.get(url);

    final responseData = json.decode(response.body);

    var items = responseData['items'] as List;

    _totalCount = (responseData['total_count'] as int);

    _repositories += items.map((e) => Repository.fromJson(e)).toList();

    notifyListeners();
    // ignore: empty_catches
  } catch (e) {}
}
```

Slika 7 Prikaz pristupa krajnjoj točki, i način spremanja dobivenih rezultata u model Repository

Izvor : Izrada autora

```dart
class Repository {
  final String? description;
  final String? reponame;
  final String? full_name;
  final int? stars;
  final int? id;
  final String? url;
  final String? author;
  final int? fork_count;
  final int? issue_count;
  final String? avatarUrl;
  final String? default_branch;
  final String? topics;
  final String? created_date;
  final String? last_pushed;
  final int? watchers_count;
  final String? language;
  final int? total_count;

  Repository({
    this.id,
    this.reponame,
    this.description,
    this.stars,
    this.url,
    this.author,
    this.fork_count,
    this.issue_count,
    this.avatarUrl,
    this.created_date,
    this.default_branch,
    this.full_name,
    this.last_pushed,
    this.topics,
    this.watchers_count,
    this.language,
    this.total_count,
  });

  factory Repository.fromJson(Map<String, dynamic> json) {
    return Repository(
      id: json['id'],
      reponame: json['name'],
      stars: json['stargazers_count'],
      description: json['description'],
      url: json['html_url'],
      fork_count: json['forks_count'],
      issue_count: json['open_issues_count'],
      author: json['owner']['login'],
```

Slika 8 Model Repository

Izvor : Izrada autora

```
Status: 200
```

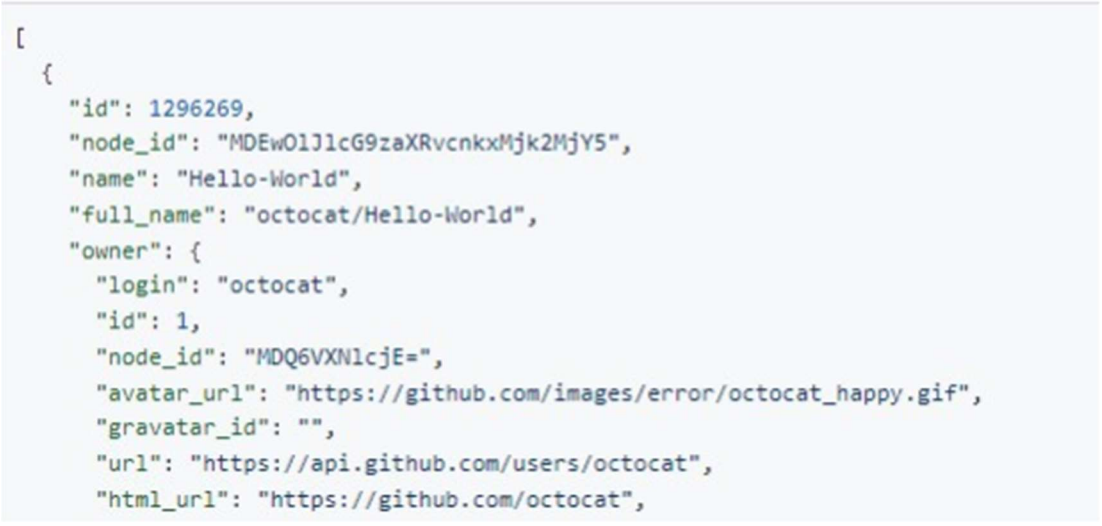
```
{
  "type": "array",
  "items": {
    "title": "Minimal Repository",
    "description": "Minimal Repository",
    "type": "object",
    "properties": {
      "id": {
        "type": "integer",
        "examples": [
          1296269
        ]
      },
      "node_id": {
        "type": "string",
        "examples": [
```

Slika 9 Shema odgovora

Izvor : Službena dokumentacija Github stranice

Na temelju sheme dobiveni su rezultati sa krajnje točke Github-a u JSON formatu prikazanom na Slika 8.

```
Status: 200
```



```
[
  {
    "id": 1296269,
    "node_id": "MDEwOlJlcG9zaXRvcnkxMjk2MjY5",
    "name": "Hello-World",
    "full_name": "octocat/Hello-World",
    "owner": {
      "login": "octocat",
      "id": 1,
      "node_id": "MDQ6VXNlcjE=",
      "avatar_url": "https://github.com/images/error/octocat_happy.gif",
      "gravatar_id": "",
      "url": "https://api.github.com/users/octocat",
      "html_url": "https://github.com/octocat",
```

Slika 10 Prikaz dobivenih rezultata sa krajnje točke Github-a

Izvor : Službena dokumentacija Github stranice

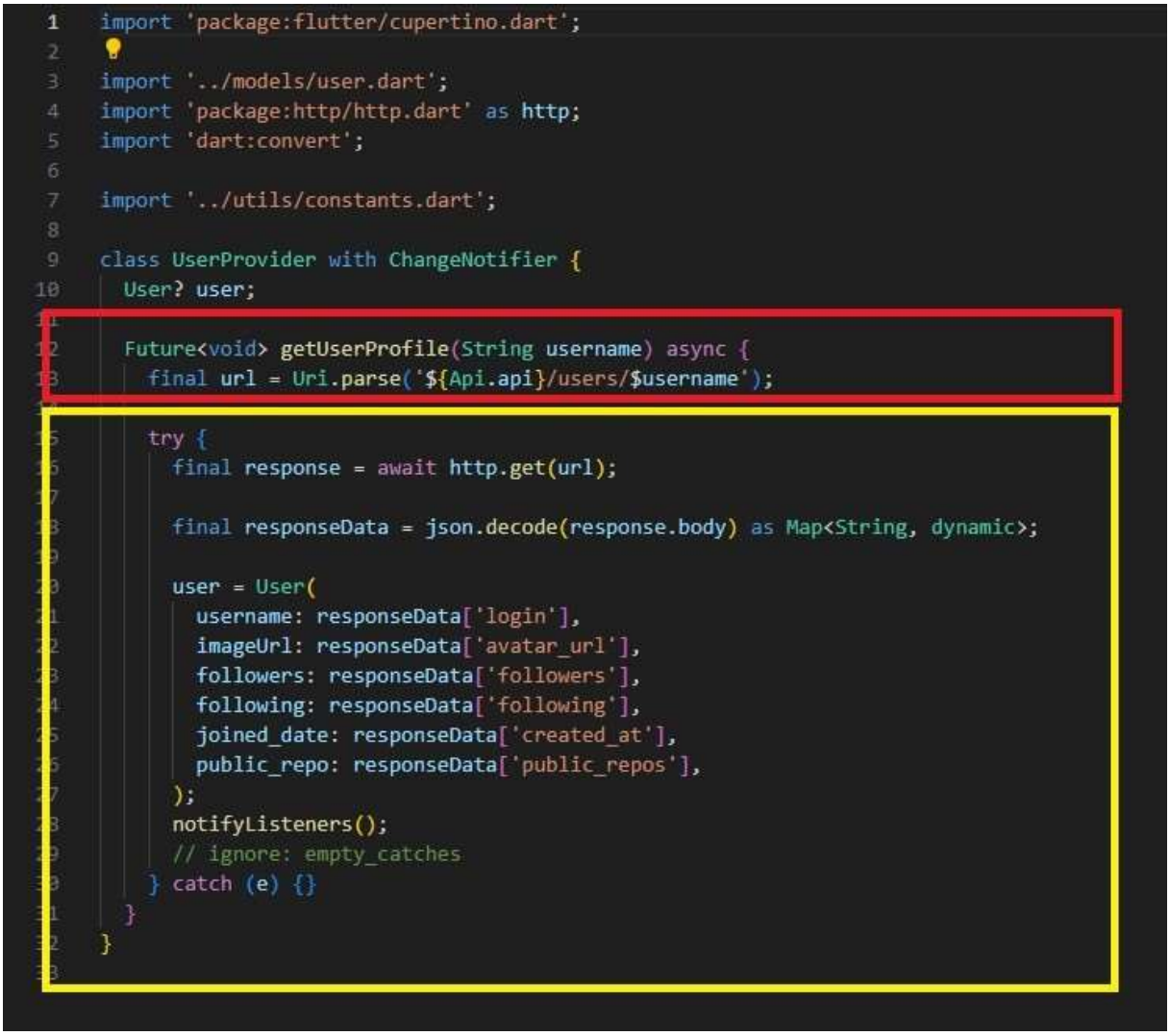
4.2.2 Krajnja točka korisnika

Unutar ovog endpointa, razdvojena je konverzija i spajanje podataka iz JSON formata u model repozitorija sa samom datotekom modela Usera odnosno korisnika. Na Slika 11 Vidljiva je konstanta Api, koju sam izdvojio u posebnu datoteku constants.dart, budući da je često bila korištena unutar projekta.

Slika 11 Datoteka constants.dart – token je osjetljiva informacija koju sam izostavio sa slike

Izvor : Izrada autora

Prikazano na Slika 12, vidljivo je unutar crvenog okvira asinkronu funkciju getUserProfile sa tipom Future gdje spaja na krajnju točku za ispis detalja o korisniku koji je izradio prethodno otvoreni repozitorij, dok je unutar žutog okvira vidljivo rukovanje iznimkama, naredbama „try-catch“.

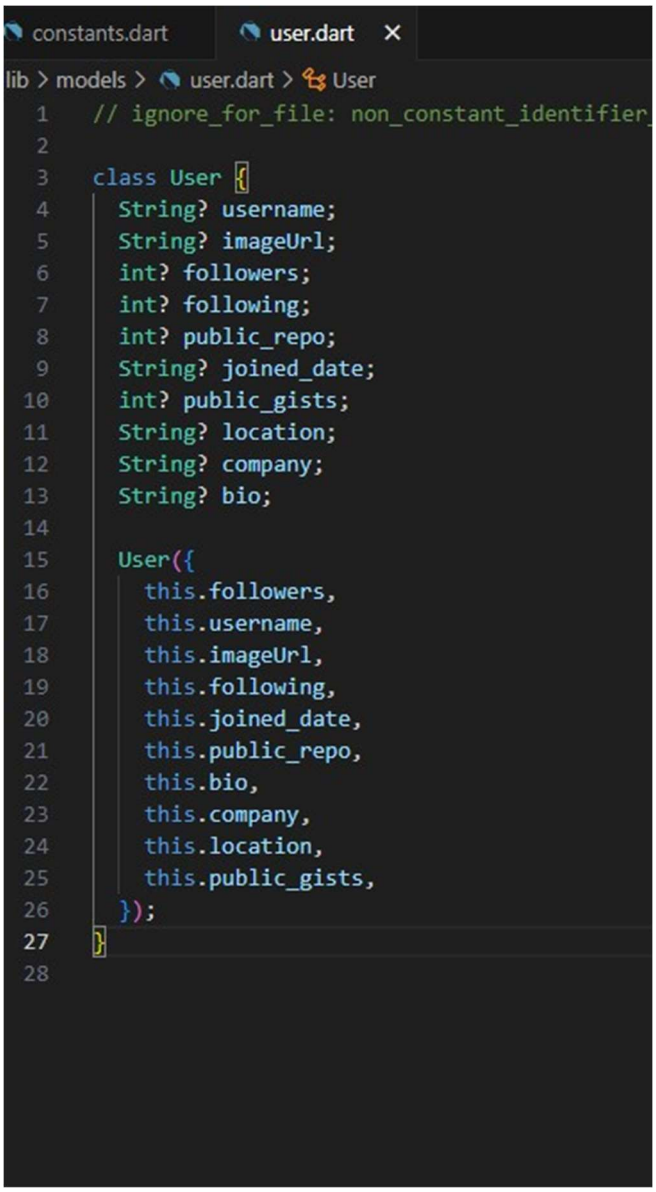


```dart
import 'package:flutter/cupertino.dart';

import '../models/user.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

import '../utils/constants.dart';

class UserProvider with ChangeNotifier {
  User? user;

  Future<void> getUserProfile(String username) async {
    final url = Uri.parse('${Api.api}/users/$username');

    try {
      final response = await http.get(url);

      final responseData = json.decode(response.body) as Map<String, dynamic>;

      user = User(
        username: responseData['login'],
        imageUrl: responseData['avatar_url'],
        followers: responseData['followers'],
        following: responseData['following'],
        joined_date: responseData['created_at'],
        public_repo: responseData['public_repos'],
      );
      notifyListeners();
      // ignore: empty_catches
    } catch (e) {}
  }
}
```

Slika 12 Prikaz funkcije spajanja na krajnju točku, i drugačiji pristup spremanja datoteke u model Usera

Izvor : Izrada autora

Detaljan prikaz modela Korisnika odnosno User-a, vidi se na Slika 13. Klasa User u ovom slučaju ima deset atributa : username (korisničko ime), imageUrl (URL slike sa GitHub-a), followers (broj sljedbenika), following (broj pratitelja), public_repo (broj javnih repozitorija), joined_date (datum prve prijave), public_gists (objavljene ideje), location (lokacija), company (organizacija), bio (biografija). String je tekstualni tip podataka koji, dok je int numerički tip podataka. Budući da se radi sa GitHub REST API-jem, velike su mogućnosti da neće svaki User pratiti druge korisnike ili imati svoje pratitelje, stoga da se spriječe buduće greške gdje je moguće da će se dobiti vrijednost nule, dodaje se „?“ na kraju tipa atributa. Upitnik u ovom kontekstu daje do znanja da navedena vrijednost može i smije biti nula.



Slika 13 Model odnosno klasa Usera

Izvor : Izrada autora

5. STATE MANAGEMENT

Ne postoji točna definicija što je zapravo state management, budući da ga svi mogu interpretirati na svoj način, no ja sam ga najbolje razumio na definiciji Kumara (2023) state, u kontekstu Flutter aplikacije, reprezentira informaciju koja se može mijenjati tijekom vremena i utjecati na prikaz i ponašanje ostalih korisničkih sučelja. Primjeri stanja ili eng. state-a, u Flutter aplikaciji uključuju korisnički unos, mrežne podatke, orijentacije uređaja i još mnogo toga. Učinkovito upravljanje stanjem uključuje rukovanje i ažuriranje ovih podataka kako bi korisničko sučelje bilo sinkronizirano s logikom aplikacije.[10]

5.1 PRISTUPI STATE MANAGEMENTU

Više je različitih pristupa upravljanje stanjem, u ovome radu korišten je Provider, koji se također vidi na Slikama 7 i 12, no u globalu postoji preko 30 načina pristupanju upravljanju stanjem. Što je izuzetno dobra stvar, budući da je prethodno spomenuto da svi state management interpretiraju na svoj način, te što postoji više od 30 načina nije loša stvar. Jedina mana koju ja mogu spomenuti je ta da bez obzira na godine staža i rada unutar Flutter-a uvijek će postojati neki pristup upravljanju stanjem s kojim nismo upoznati.

Na idućoj slici (Slika 14) prikazani su razni pristupi upravljanju stanjem, te broj korisnika kojima se sviđa isti pristup, Pub score kao mjerilo kvalitete i Popularity postotak koji prikazuje koliko developera koristi koji paket (mjeri broj aplikacija koji ovise o određenom pristupu upravljanja stanjem u zadnjih 60 dana. Rezultat je prikazan na skali do 100%).[11]

[10] State Management in Flutter: A Comprehensive Guide, Nitin Kumar (2023)

[11] State Management approaches in Flutter, Dmitrii Slepnev (2020)

Name	Likes	PUB	Popularity
AsyncRedux	59	100	89%
BLoC	1163	110	99%
blocstar	1	100	0%
cubit	564	110	99%
Dartea	2	90	75%
fish_redux	32	110	91%
Flutter Hooks	294	110	96%
Get	1418	100	98%
InheritedWidget	-	-	-
maestro	6	110	24%
meowchannel	6	40	21%
MobX	409	110	98%
Momentum	63	110	69%
MVC_pattern	47	100	97%
mvcprovider	4	90	40%
mvvm_builder	9	100	54%
no_bloc	6	110	90%
OSAM	13	90	60%
Provider	2307	110	100%
ProviderScope	0	90	48%
rebloc	10	110	73%
Redux	153	105	97%
redux_compact	8	100	39%
riverpod	188	110	94%
RxVMS	30	105	87%
Scoped Model	89	100	97%
state_notifier	87	100	95%
states_rebuilder	238	100	96%
stream_state	3	110	63%
var_widget	0	90	0%
vmiso	0	80	31%
stacked	409	110	96%

Slika 14 Tablica popularnih pristupa upravljanju stanjem

Izvor : State Management approaches in Flutter, str 29-30, D. Slepnev (2020)

Obzirom da je prošlo 4 godine od istraživanja Slepneva, pronađena je tablica popularnosti napravljena of Flutter zajednice koja prikazuje promjenu u popularnosti kroz 4 godine. (Slika 15)

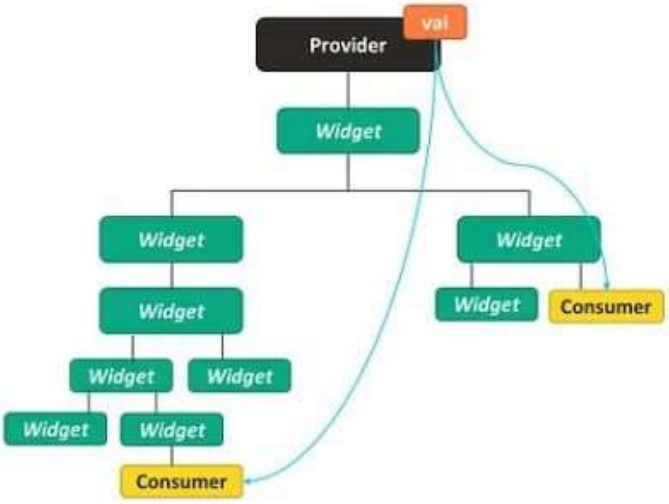
Package	GitHub URL	Stars
Bloc/Cubit	Bloc GitHub	11k
GetX	GetX GitHub	9.3k
Riverpod	Riverpod GitHub	5.2k
Provider	Provider GitHub	4.9k
Flutter Hooks	Flutter Hooks GitHub	2.9k
MobX	MobX GitHub	2.3k
Redux	Redux GitHub	1.6k

Slika 15 Prikaz trenutne popularnosti upravljanjem stanjem

Izvor : Flutter zajednica

5.2 PROVIDER

Na internetu možemo naći sto različitih definicija Flutter-a, no budući da je pojam apstraktan, smatram da navedena definicija poprilično detaljno pojašnjava kako Provider utječe na rad aplikacije, te komunikaciju između widgeta. Zamislimo da želimo podijeliti informaciju kroz „drvo widgeta“ i svaki put kada prelazimo s jednog ekrana na drugi, moramo dijeliti entitet sa zajedničkim podacima između njega. Ne zvuči bas praktično, stoga je developerski tim[12] iz Fluttera smislio pojam InheritedWidgets, koji omogućuju dijeljenje informacija kroz navedeno drvo widgeta. Nažalost korištenje InheritedWidgets-a je bilo izuzetno teško implementirati. Zbog navedenog razloga, paket The Provider započeo je kao proširenje InheritedWidget olakšavajući rukovanje informacijama kroz stablo widgeta. Na idućoj slici (Slika 16) vidimo izvedbu Providera. Može dijeliti informacije s onim tko ih treba, a Consumer može ažurirati podatke svog roditelja.[13]



Slika 16 Izvedba Providera

Izvor : Flutter Provider: What is it, what is it for, and how to use it? Daniel Herrera Sanchez (2022), Medium

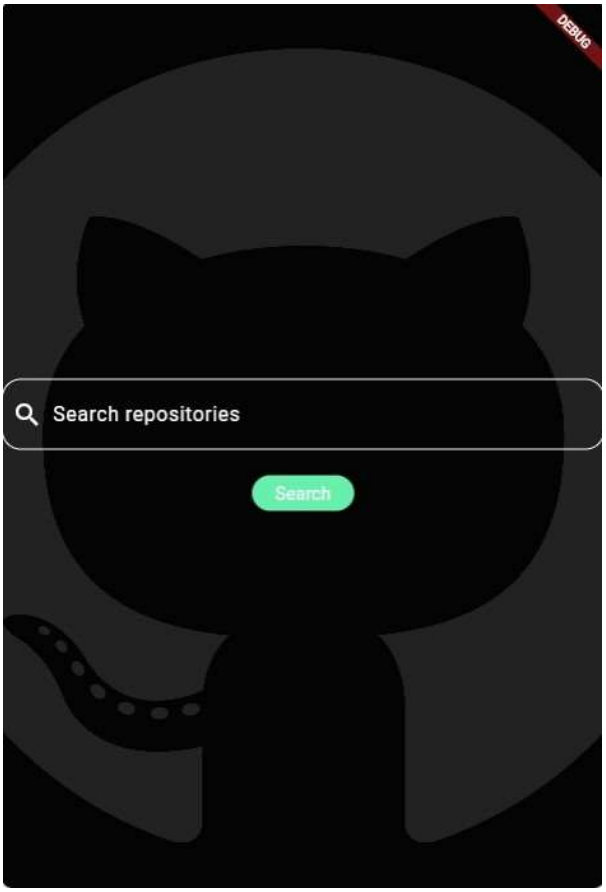
[12] Developerski tim – grupa programera

[13] Flutter Provider: What is it, what is it for, and how to use it? Daniel Herrera Sanchez (2022), Medium

6. DIZAJN

6.1 Search zaslon

Izgled search zaslona koji je ujedno i početni zaslon je prikazan na idućoj slici (Slika 17), dok je kod korišten za izradu početne stranice prikazan na Slika 18.



Slika 17 Početni zaslon

Izvor : Izrada autora

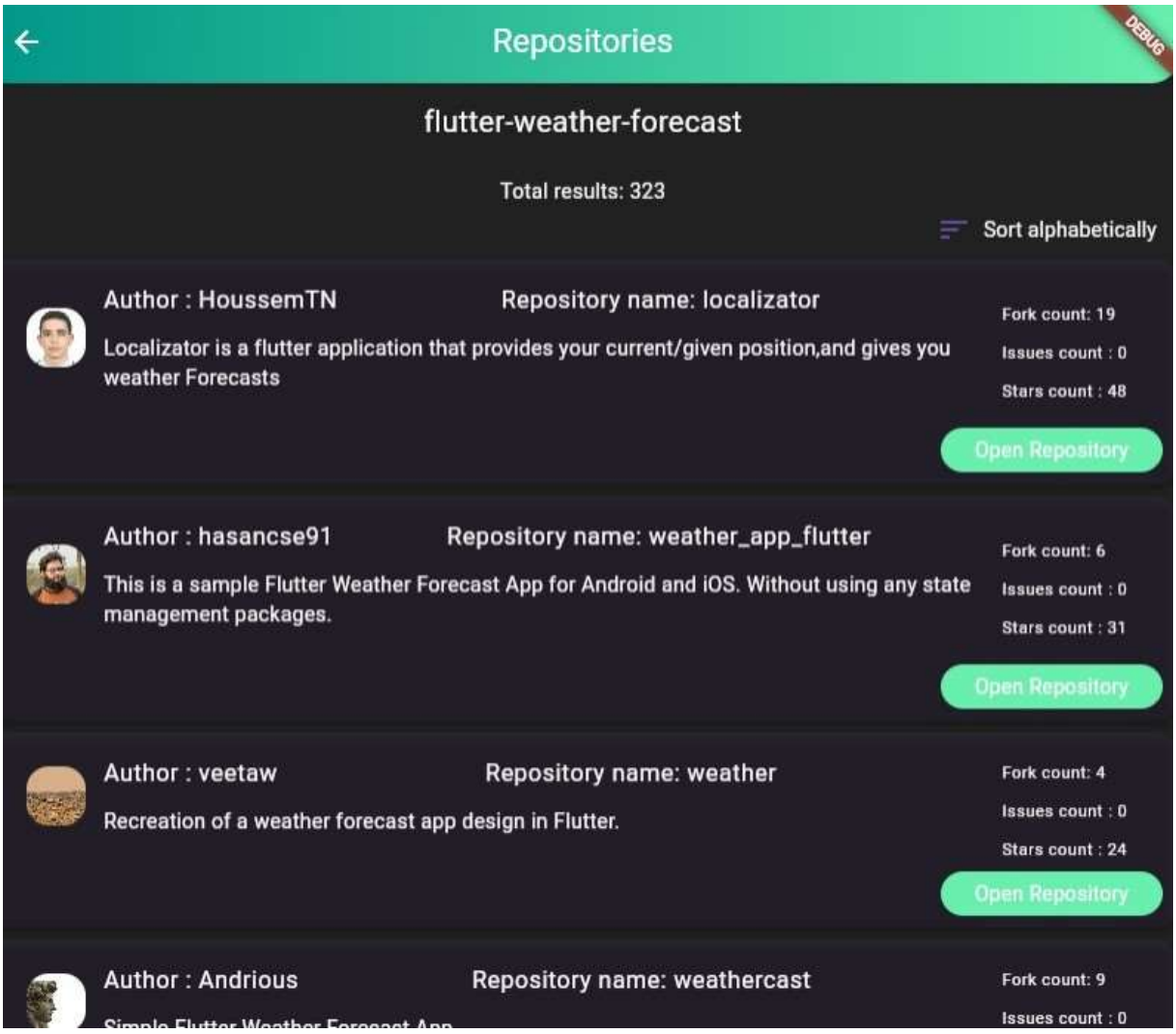
```
class _SearchScreemState extends State<SearchScreem> {
  Widget build(BuildContext context) {
    final controller = TextEditingController();
    return Scaffold(
      backgroundColor: Colors.black87,
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          Form(
            child: Column(
              children: <Widget>[
                TextField(
                  controller: controller,
                  style: const TextStyle(
                    color: Colors.white,
                  ), // TextStyle
                  decoration: InputDecoration(
                    enabledBorder: const OutlineInputBorder(
                        borderRadius: BorderRadius.all(Radius.circular(15)),
                        borderSide: BorderSide(color: Colors.white)), // OutlineInputBorder
                    floatingLabelBehavior: FloatingLabelBehavior.auto,
                    prefixIcon: const Icon(Icons.search, color: Colors.white),
                    labelText: 'Search repositories',
                    labelStyle: const TextStyle(color: Colors.white),
                    hintStyle: const TextStyle(color: Colors.white),
                    errorBorder: const OutlineInputBorder(
                      borderSide: BorderSide(color: Colors.red, width: 5),
                    ), // OutlineInputBorder
                    focusedBorder: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(30),
                      borderSide:
                          const BorderSide(color: Colors.greenAccent, width: 2),
                    ), // OutlineInputBorder
                  ), // InputDecoration
                ), // TextField
                const SizedBox(
                  height: 20,
                ), // SizedBox
                ClipRRect(
                  borderRadius: BorderRadius.circular(20),
                  child: Container(
                    color: Colors.greenAccent,

                    child: MaterialButton(
                      onPressed: () {
                        {
                          Future.delayed(const Duration(milliseconds: 3000));
                          Navigator.of(context).push(MaterialPageRoute(
                              builder: (context) => RepoScreen(
                                  repos: controller.text,
```

Slika 18 Programski kod zaslužan za početni zaslon

Izvor : Izrada autora

6.2 Prikaz svih repozitorija

Na slici 19 prikazani su svi repozitoriji koji u sebi sadrže ključnu riječ „flutter-weather-forecast“, te su sorirani po broju zvijezda. Također je moguće sortiranje abecedno pritiskom na tipku „Sort alphabetically“ u gornjem desnom kutu.



Slika 19 Prikaz repozitorija prilikom pretraživanja „flutter-weather-forecast“, sortirano po broju zvijezda

Izvor : Izrada autora

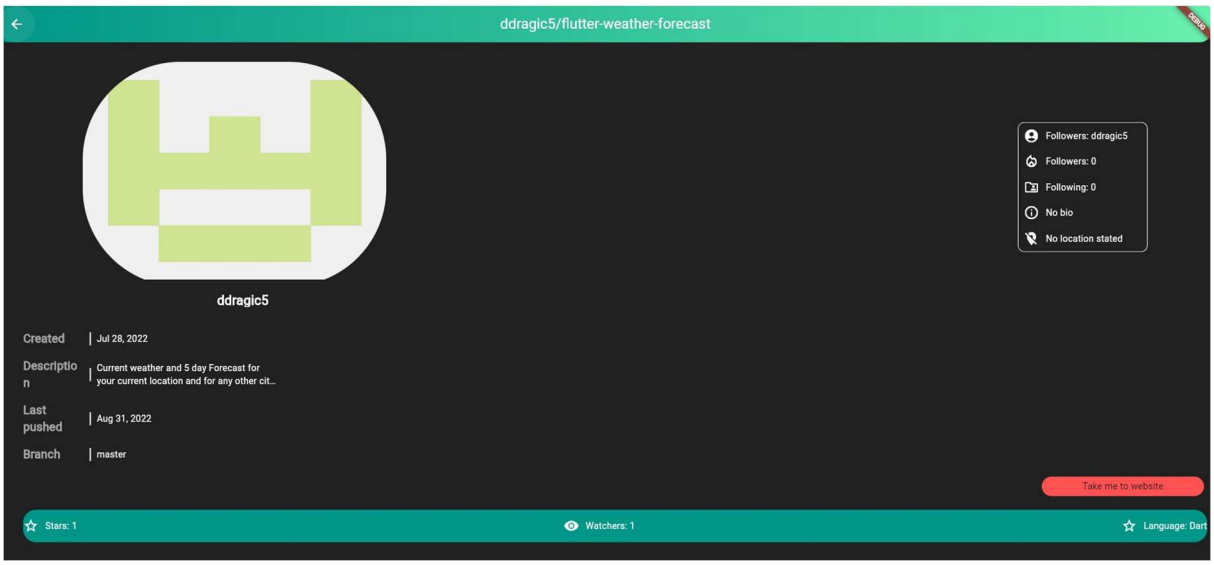
Na idućoj slici (Slika 20) prikazan je dio koda iz datoteke repo_screen.dart, u kojem se nalazi sva logika potrebna za prikaz svih repozitorija koji sadrže prethodno upisanu ključnu riječ.

Slika 20 Prikaz koda repo_screen datoteke

Izvor : Izrada autora

6.3 Detaljan opis repozitorija

Na idućim slikama (Slika 21, 22) prikazan je detaljan opis mog repozitorija na GitHub servisu pod imenom „flutter-weather-forecast“, u kojem su prikazani detalji o repozitoriju. Poput datuma izrade repozitorija, opis repozitorija, datuma zadnje promjene, i na kojem branch-u[14] je napravljen repozitorij. U desnom dijelu zaslona može se vidjeti par detalja o korisniku koji je izradio repozitorij, odnosno o meni. Pri samom dnu je izrađen jedan container[15] u kojem su prikazani broj zvijezda repozitorija, koliko pregleda ima, te u kojem je programskom jeziku pisan kod.



Slika 21 Detaljan prikaz repozitorija autora pod imenom „flutter-weather-forecast“

Izvor : Izrada autora

[14] Branch – grana u GitHub-u, najčešća master (main)

[15] Container- widget koji izgleda kao kontejner

```dart
return Scaffold(
    appBar: appBarTheme(title: widget.full_name),
    body: provider(context).isLoading
        ? const Center(
            child: CircularProgressIndicator(),
          ) // Center
        : Container(
            color: Colors.black87,
            child: Padding(
              padding: const EdgeInsets.only(left: 30),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.center,
                children: <Widget>[
                  const SizedBox(
                    height: 30,
                  ), // SizedBox
                  Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                      AuthorImageAndName(context, repo),
                      Column(
                        children: <Widget>[
                          ClipRRect(
                            borderRadius: BorderRadius.circular(10),
                            child: Container(
                              margin: const EdgeInsets.only(right: 100),
                              decoration: BoxDecoration(
                                  color:
                                      const Color.fromRGBO(51, 51, 51, 0),
                                  border: Border.all(color: Colors.white),
                                  borderRadius: BorderRadius.circular(10)), // BoxDecoration
                              height: 200,
                              width: 200,
                              child: Padding(
                                padding: const EdgeInsets.all(8),
                                child: UserInfoDetailsPage(
                                    dateF, user, context),
                              ), // Padding
                            ), // Container
                          ), // ClipRRect
                        ], // <Widget>[]
                      ), // Column
                    ],
                  ), // Row
                  const SizedBox(
                    height: 20,
                  ), // SizedBox
                  MainContainerView(dateF, repo, context),
```

Slika 22 Programski kod details_screen.dart datoteke

Izvor : Izrada autora

7. ZAKLJUČAK

U ovom radu cilj je upoznavanje procesa izrade mobilne aplikacije na više platformi pomoću Flutter framework-a i objektno-orijentiranog programskog jezika Dart-a. Konstantan rast uporabe Flutter-a, je dovoljan pokazatelj efikasnosti. Google je dokazao kako je dovoljan jedan framework pri izradi aplikacija na primjerima eBay-a, Alibab-e i sličnih. Očekivani doprinos ovog rada je popularizacija Flutter-a, u cilju približavanja potražnje sa nativnim razvojnim okvirima za razvoj aplikacija poput Apple-ovog Swift-a. Naravno da Flutter ima svojih nedostataka, no nije za očekivati da relativno nova tehnologija funkcionira savršeno, bitan je kontinuirani razvoj koji Google pruža od samog početka. Problem izrade nativnih aplikacija je utrošena ogromna količina vremena i poprilično velik financijski zalogaj, što stvara izazove i određenim većim organizacijama.

Rezultat ovog rada izrada je funkcionalne višeplatformske aplikacije za pretraživanje GitHub repozitorija. Aplikacija je ostvarena sa još par dodatnih značajki poput prikazivanja korisničkog profila korisnika koji je izradio aplikaciju te prikazivanja detalja o javnom repozitoriju. Dodatno je potrebno naglasiti da je aplikacija testirana kao web aplikacija te je potrebno doraditi kod, kako bi se značajke ispravno prikazivale na iOS mobilnim uređajima i Android mobilnim uređajima.

8. IZJAVA

Izjava o autorstvu završnog rada i akademskoj čestitosti

Ime i prezime studenta: Daniel Dragić

Matični broj studenta: 0307014130

Naslov rada: Primjena Flutter: Darta i izrada mobilne aplikacije

Pod punom odgovornošću potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.

Potvrđujem da je elektronička verzija rada identična onoj tiskanoj te da je to verzija rada koju je odobrio mentor.

Datum _____

Potpis studenta _____

9. POPIS LITERATURE

9.1 KNJIGE I ČLANCI

Iqbal H. Sarker and K. Apu (2014) MVC Architecture Driven Design and Implementation of Java Framework for Developing Desktop Application. https://www.researchgate.net/profile/Iqbal-Sarker/publication/291098214_MVC_Architecture_Driven_Design_and_Implementation_of_Java_Framework_for_Developing_Desktop_Application/links/57e2011908aed96fbbb081bd/MVC-Architecture-Driven-Design-and-Implementation-of-Java-Framework-for-Developing-Desktop-Application.pdf

Matija Varga (2022) EDUizziv . "Izzivi poučevanja in vrednotenja znanja". 16. – 18. februar 2022. Znanstveni rad. Znanstvena konferenca. http://www.eduvision.si/Content/Docs/Zbornik_EDUizziv_Februar_22.pdf

9.2 INTERNETSKI IZVORI

International Business Machines, IBM. Preuzeto s https://www.ibm.com/topics/rest-apis (10. veljače 2024.)

GitHub REST API dokumentacija. Preuzeto sa službene stranice GitHub Documentations (13.veljače 2024.) https://docs.github.com/en/rest?apiVersion=2022-11-28

What is an API endpoint? Postman. Preuzeto s https://blog.postman.com/what-is-an-api-endpoint/ (25.veljače 2024.)

State Management in Flutter: A comprehensive Guide, Nitin Kumar (9. rujna 2023.). Preuzeto s https://medium.com/@enitinmehra/state-management-in-flutter-a-comprehensive-guide-7212772f026d (1.ožujka 2024.)

Flutter Provider: What is it, what i sit for, and how to use it? Daniel Herrera Sanchez (16. kolovoza, 2022.). Preuzeto s https://medium.com/bancolombia-tech/flutter-provider-what-is-it-what-is-it-for-and-how-to-use-it-47d6941860d7 (2.ožujka 2024.)

Kotlin. Preuzeto s https://kotlinlang.org/docs/cross-platform-frameworks.html#what-is-a-cross-platform-app-development-framework (7. ožujka 2024.)

POPIS SLIKA, TABLICA I GRAFIKONA

ŽIVOTOPIS

europass

Daniel Dragić

Datum rođenja: 17/10/1998 | **Državljanstvo:** hrvatsko | **Telefonski broj:** (+385) 993348734 (Mobilni telefon) |

E-adresa: daniel.dragic123@gmail.com |

Adresa: Ulica Mihovila Pavlinovića 15, Kuća, 31220, Osijek, Hrvatska (Kućna)

RADNO ISKUSTVO

01/04/2022 – TRENUTAČNO Darda, Hrvatska
INFORMATIČAR BELJE PLUS D.O.O.

-održavanje mrežne infrastrukture
-rad sa CISCO preklopnicima
-administracija ICT opreme
-prilagođavanje operacijskih sustava potrebama pojedinaca
-rad na domeni
-analizirati i poboljšavati sustave unutar organizacije
-integrirati nove elemente unutar ICT infrastrukture
-poboljšanje sigurnosti sustava

19/02/2019 – 08/12/2021 Gospić, Hrvatska
VOJNIK MINISTARSTVO OBRANE REPUBLIKE HRVATSKE

OBRAZOVANJE I OSPOSOBLJAVANJE

24/09/2013 – 13/05/2017 Osijek, Hrvatska
TEHNIČAR ZA RAČUNALSTVO Elektrotehnička i prometna škola Osijek

Internetske stranice www.elpros.net

10/10/2021 – TRENUTAČNO Zaprešić, Hrvatska
PRVOSTUPNIK INFORMACIJSKIH TEHNOLOGIJA (BACC. INF. TECH.) Veleučilište Baltazar Zaprešić

Internetske stranice www.bak.hr

JEZIČNE VJEŠTINE

Materinski jezik/jezici: **HRVATSKI**

Drugi jezici:

	RAZUMIJEVANJE		GOVOR		PISANJE
	Slušanje	Čitanje	Govorna produkcija	Govorna interakcija	
ENGLESKI	C1	C2	B2	C1	B2

Razine: A1 i A2: temeljni korisnik; B1 i B2: samostalni korisnik; C1 i C2: iskusni korisnik

DIGITALNE VJEŠTINE

Internet | Komunikacijski programi (Skype Zoom TeamViewer) | Rad na raunalu | MS Office (Word Excel PowerPoint) | Flutter | Dart | LDAP | Sposobnost prilagodavanja promjenama | MS Office Administration | Timski rad | BMC | Windows