Web aplikacija za integraciju ponuda poslova iz više izvora

Lukač, Matteo

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:759478

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Matteo Lukač

Web aplikacija za integraciju ponuda poslova iz više izvora

Diplomski rad

Pula, rujan, 2022. godine

Sveučilište Jurja Dobrile u Puli

Fakultet informatike u Puli

Matteo Lukač

Web aplikacija za integraciju ponuda poslova iz više izvora

Diplomski rad

JMBG: 0016093208, redoviti student

Studijski smjer: Sveučilišni diplomski studiji Informatike

Predmet: Napredni algoritmi i strukture podataka

Znanstveno područje: Društvene znanosti

Znanstveno polje: Informacijske i komunikacijske znanosti

Znanstvena grana: Informacijski sustavi i informatologija

Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan, 2022. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Matteo Lukač kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Diplomskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Matteo Lukač

U Puli, rujan, 2022 godine

IZJAVA
o korištenju autorskog djela

Ja, Matteo Lukač dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Web aplikacija za integraciju ponuda poslova iz više izvora“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu

U Puli, 11. rujna 2022. godine

Potpis

Matteo Lukač

Sadržaj

1.Uvod 1
2. Kritička analiza postojećih rješenja 3
3. Korištene tehnologije 6
3.1 C# 6
3.2 Flutter i Dart 7
3.3 Puppeteer 7
3.4 Entity 8
4 Implementacija 10
4.1 Frontend web klijent 10
4.1.1 Arhitektura 10
4.1.2 MyApp 10
4.1.3 Promjena putanje i Dependency Injection 12
4.1.4 Stranica učitavanja web aplikacije 13
4.1.5 Dohvat i slanje na backend 15
4.1.6 Glavna stranica aplikacije 25
4.1.7. Stranica oglasa 28
4.2. Frontend web scraping klijent 28
4.2.1. Stranica učitavanja web scraping aplikacije 28
4.2.2. Klasa Posaohr 29
4.2.3. Klasa MojPosaoNet 35
4.2.4. Klasa BikaNet 38
4.3. Backend server i baza podataka 41
4.3.1. Api 41
4.3.2. Aplication 43
4.3.3. Domain i Persistance 44
5. Zaključak 46
Literatura 47
Popis slika 49
SAŽETAK 51
ABSTRACT 51

1.Uvod

Pronalaženje posla postaje sve veći problem s obzirom na to da broj web stranica za oglašavanje poslova raste. Svaka web stranica za oglašavanje poslova ima mogućnost objave oglasa. Zbog toga korisnici moraju posjećivati veći broj web stranica i pri tome često gledati iste oglase jer se poslodavci oglašavaju na više web stranica. Svakodnevno posjećivanje više web stranica kako bi se provjerilo je li u međuvremenu objavljen novi oglas iziskuje mnogo vremena. Postupak objave oglasa vodi poslodavce kroz određene korake, te se pri tom od poslodavaca traži da popune određene podatke, koji se često razlikuju s obzirom na to da svaka od tih web stranica ima različite podatke čiji unos je obavezan za korištenje te web stranice. Tako na web stranici posao.hr nudi poslodavcu mogućnost odabira djelatnosti "Neprofitne organizacije", dok na moj-posao.net nudi poslodavcu odabir djelatnosti „Državna služba i neprofitne organizacije“. Iz navedenog je vidljivo kako svaka web stranica ima slobodu dizajna svoje stranice, odnosno da ne postoje standardi kojih se moraju pridržavati. Nedostatak tih standarda predstavlja probleme za korisnike (kako za poslodavce tako i osobe koje traže posao) te iziskuje dodatno vrijeme za snalaženje po web stranicama jer se svaka web stranica znatno razlikuje od druge.

Predmet ovog projekta je jedno od mogućih rješenja navedenog problema rastućeg broja web stranica za oglašavanje poslova. Na glavnoj stranici imamo mogućnost brzog i jednostavnog pretraživanja oglasa na četiri različita načina. Moguća je pretraga po riječi iz teksta oglasa, pretraga po naziva grada, pretraga izborom jedne od ponuđenih djelatnosti i pretraga odabirom jedne od ponuđenih županija. Moguće je pretraživati oglase odabirom samo jedne od navedenih tražilica ili više njih. Oglasi koji se prikazuju na web stranici poredani su od najnovijeg prema najstarijem. Svaki prikazani oglas u sebi sadrži naziv radnog mjesta koje se oglašava, mjesto rada i istek oglasa. Odabirom oglasa otvara se stranica oglasa iz koje korisnik iščitava podatke samog oglasa. Ukoliko je taj oglas objavljen na više web stranica korisnik može usporediti te oglase. Svaki taj oglas je prikazan u zasebnoj kartici sa nazivom web stranice sa koje je isti pokupljen. Svaka kartica sadrži hipervezu prema web stranci oglasa. Nakon što korisnik završi sa pregledom oglasa omogućen mu je povratak na glavnu stranicu uz pomoć gumba za povratak koji se nalazi u gornjem lijevom kutu. Ovaj projekt ima backend napravljen u C# i dva frontend-a napravljena u Flutter-u.

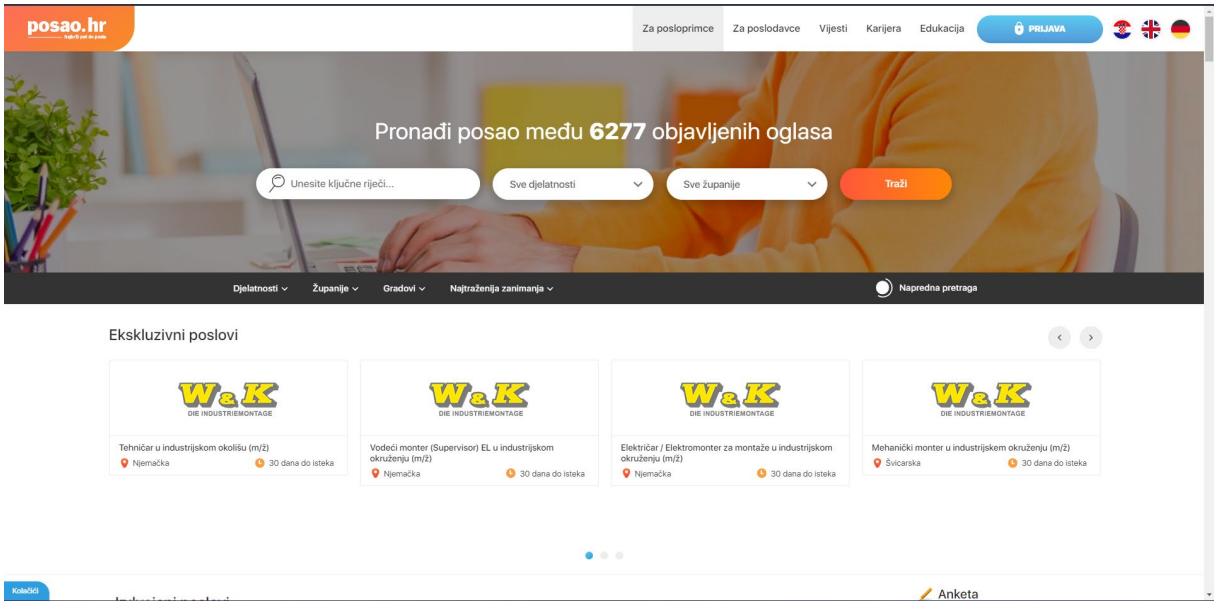
Jedan frontend-a je napravljen kao web aplikacija za prikaz i traženje oglasa dok drugi frontend služi za web scraping i napravljen je kao windows aplikacija. Web scraping je napravljen u frontend-u jer je Puppeteer paket razvijeniji za Flutter nego za C#.

Ovaj rad se sastoji od uvoda i četiri poglavlja o razvoju web aplikacije za integraciju ponuda poslova iz više izvora te zaključka. Drugo poglavlje sastoji se od kritičke analize tržišta, treće poglavlje upoznaje nas s tehnologijom koja se koristi u ovom projektu. Četvrto poglavlje opisuje način rada web klijenta, web scraping-a klijenta te api aplikacije sa isječcima koda iz programa koji je napisan u svrhu demonstracije razvoja za ovaj rad.

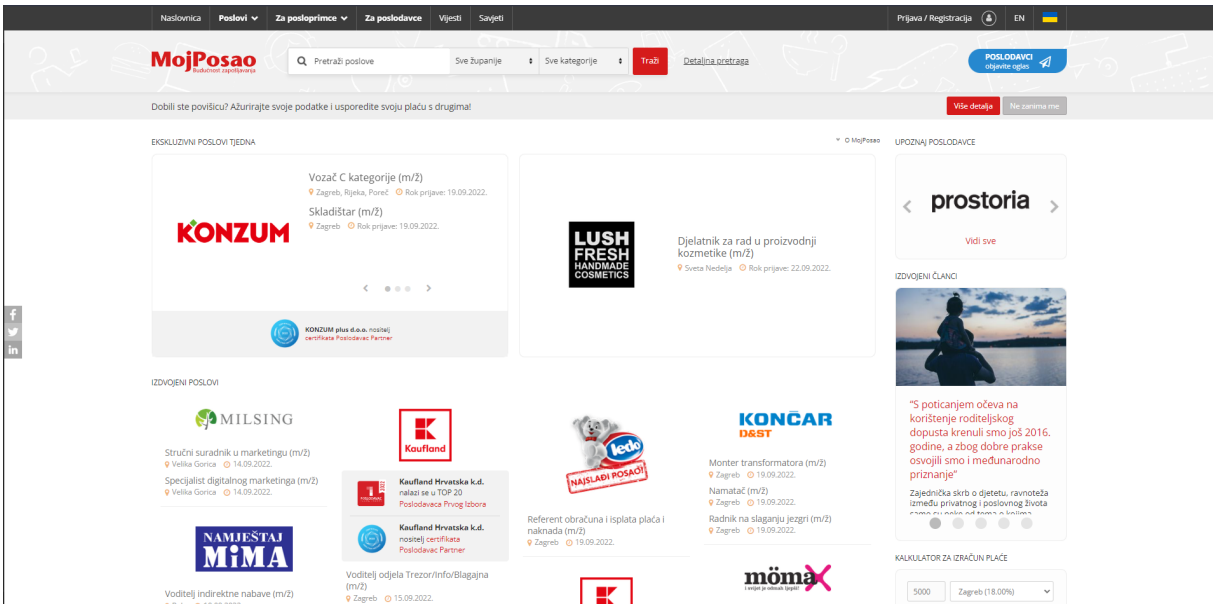
2. Kritička analiza postojećih rješenja

U današnje vrijeme kada ima sve veći broj web stranica za oglašavanje poslova za očekivati bi bilo da neka od web stranica prikuplja podatke o oglasima sa svih web stranica konkurencije, ali ovim istraživanjem utvrđeno je da to nije tako. Naime, većina web stranica prikuplja podatke samo iz jednog izvora. Primjerice, web stranica Posao.hr prikuplja podatke sa web stranice burzarada.hzz.hr, dok web stranica hr.jooble.org prikuplja podatke sa web stranice bika.net.

Web stranice sa najvećim brojem oglasa za rad na području Republike Hrvatske su posao.hr (slika 1) i moj-posao.net (slika 2).



Slika 1. Prikaz web sučelja posao.hr



Slika 2. Prikaz web sučelja posao.hr

Jedan od razloga za veći broj oglasa na navedenim web stranicama u usporedbi s drugim web stranicama je da navedene dvije daju mogućnost objave dostavljenog HTML-a, što omogućuje poslodavcima koji objavljuju oglase na tim stranicama da se istaknu u odnosu na ostalu konkurenciju. Nadalje, drugi razlog je to što navedene web stranice (moj-posao.net i posao.hr) na svojim početnim stranicama imaju predviđenu mogućnost filtriranja oglasa po županijama i djelatnostima, a što omogućuje korisnicima brži pronalazak oglasa.

Sve web stranice za oglašavanje poslova imaju mogućnost pretraživanja po pojmovima koji su sadržani u oglasima, ali sve web stranice nemaju prikazane iste informacije u oglasu odnosno ne sadrže jednake pojmove. Tako na primjer web stranica posao.hr ne sadrži informaciju o tome kada je oglas objavljen, pa ako korisnika ta informacija interesira tada bi morao koristiti mogućnost naprednog pretraživanja te u naprednoj tražilici popuniti polje „Prikaži samo oglase objavljene u posljednjih:“ počevši od 0, što prikazuje oglase od samo tog dana. Iz istog razloga bi morao dalje povećavati broj dok ne bi pronašao traženi oglas odnosno oglase objavljene ranije.

Iako nema direktne konkurencije ovom projektu odnosno proizvodu, u praksi postoje slični indirektni proizvodi od kojih je jedna web stranica jeftinije.hr koja omogućuje korisnicima pretraživanje proizvoda s većeg broja različitih web stranica, a koje onda korisnicima prikazuje u obliku liste istih proizvoda i to počevši od web stranice s cjenovno najpovoljnijeg takvog proizvoda pa nadalje do web stranice s najvišom

cijenom takvog proizvoda. Web stranica jeftinije.hr tako omogućuje korisnicima da pronađu cjenovno najpovoljniji proizvod i web stranicu na kojoj je taj proizvod oglašen odnosno putem koje se isti može kupiti.

3. Korištene tehnologije

Na ovom projektu kao primarne tehnologije korištene su: C# za backend, Flutter/Dart za frontend, Puppeteer za web scraping i SQL/ Entity framework za bazu podataka.

3.1 C#

C# je objektno i komponentno orijentiran programski jezik kojeg je razvila tvrtka Microsoft i to 2000. godine. Na ovom projektu korišten je .net 6.0 programski okvir koji je otvorenog koda. 2002. godine je izašla prva inačica .NET-a. 2016 izlazi . NET Core a 2020 se ta dva programski okvira spajaju u zajednički naziv .NET. C# je kompajlirani jezik što znači da developer mora kompajlirati izvorni jezik u srednji jezik (Intermediate Language). Prilikom izvršavanja koda srednji jezik se kompajlira u strojni kod trenutno ciljanog sustava kojega zatim procesor izvršava. Prednost kompajliranja najprije u srednji jezik je to da se kod može izvršavati na bilo kojem od postojećih sustava naprimjer Windows, Linux, Playstation i drugi. Međutim, nedostatak kompajliranja najprije u srednji jezik je u tome da za razliku od C++ koji se odmah kompajlira u strojni kod, C# ima dodatne troškove izvedbe s obzirom na to da najprije mora pretvoriti srednji jezik u strojni kod [1].

S druge strane jedna od prednosti C# je to da je C# objektno orijentiran jezik visoke razine, a što omogućuje da kod bude čitljiviji i lakši za održavanje. Pri tome objektna orijentiranost omogućava nam da kod pišemo u manjim cjelinama pomoću klasa i metoda što olakšava snalaženje i povećava razumijevanje koda. Svaka funkcionalnost koda tada se može iskazati u zasebnoj metodi koja se može imenovati, a metode se pri tome mogu grupirati u klase. C# spada u jezik visoke razine budući da je sličan ljudskom jeziku odnosno ima visoku razinu apstrakcija od strojnog koda što omogućuje lakoću čitanja računalnog koda.

Neke od ostalih prednosti C# su [2]:

- Ugrađeni sakupljač smeća,
- Jezik siguran za tip,
- Temeljita dokumentacija za C# i .net (video serijali, interaktivne poduke i objašnjenje problema).

3.2 Flutter i Dart

Flutter je komplet za razvoj softvera koji je otvorenog koda kojeg je 2017. godine razvio Google. On podržava razvoj aplikacija koje se mogu izvoditi na Windows, Linux, macOS, Android, iOS, Google Fuchsia sustavima, a na svim sustavima se izvodi iz iste kodne baze.

Glavne komponente Fluttera su [4] :

- Dart platforma,
- Flutter engine (Skia Graphics Engine),
- Foundation biblioteka,
- Dizajn-specifični widgeti,
- Flutter razvojni alati (DevTools).

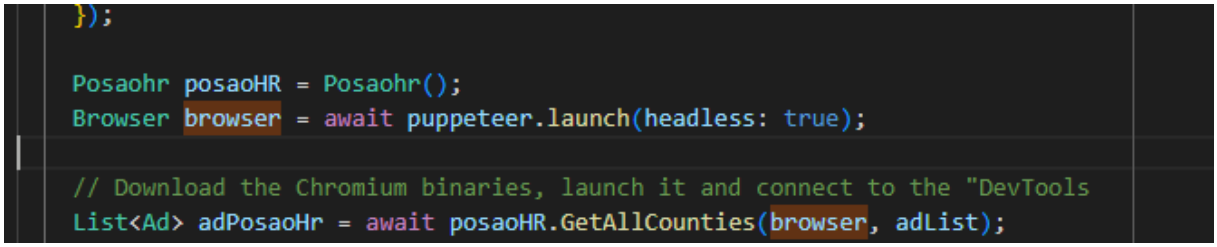
Dart je programski jezik koji je 2011. godine razvio Google. Namijenjen je za razvoj na klijentskoj strani ali podržava i razvoj serverske strane i desktop aplikacija [5]. Dart sintaksa je slična C a kao i C# objektivno je orijentiran. Neka od njegovih svojstva su: ugrađeni sakupljač smeća, null sigurnost, jezik siguran za tip, sučelja, apstraktne klase. Može se kompilirati u izvorni kod ili u Javascript [6].

3.3 Puppeteer

U ovom projektu je korištena Dart biblioteka Puppeteer koja je port biblioteke Puppeteer Node.JS. JavaScript Puppeteer je razvijen od The Chrome DevTools tima 2018. godine, dok je port za Dart u razvoju od 2019. godine. Puppeteer je API koji kontrolira Headless Chromium browser za automatizaciju. Puppeteer omogućava automatizaciju većine stvari koje je moguće ručno raditi u pregledniku [7][8].

Na slici 3 prikazano je dodavanje Puppeteer paketa u projekt dok na slici 4 možemo vidjeti njegovo korištenje.

Slika 3. Dodavanje Puppeteer paketa u projekt



Slika 4. Korištenje Puppeteer paketa

Neki od primjera korištenja Puppeteer paketa su [8]:

- „Generiranje snimke zaslona i PDF-ove stranica,
- Pretraživanje SPA (Aplikacija na jednoj stranici) i generiranje unaprijed renderiranog sadržaja (tj. "SSR" (Prikazivanje na strani poslužitelja)),
- Automatiziranje podnošenja obrazaca, testiranje korisničkog sučelja, unos tipkovnicom itd.,
- Stvaranje ažurnog, automatiziranog okruženja za testiranje. Pokretanje svojih testova izravno u najnovijoj verziji Chroma koristeći najnoviju verziju JavaScript i značajka preglednika.“

3.4 Entity

„Entity Framework je ORM okvir otvorenog koda za .NET aplikacije koje podržava Microsoft. Programerima omogućuje rad s podacima koristeći objekte klasa specifičnih za domenu bez fokusiranja na temeljne tablice i stupce baze podataka u kojima su ti podaci pohranjeni. Uz Entity Framework, programeri mogu raditi na višoj razini apstrakcije kada se bave podacima i mogu stvarati i održavati aplikacije orijentirane na podatke s manje koda u usporedbi s tradicionalnim aplikacijama.„ [9].

Postoje dva načina rada sa Entity Frameworkom. Prvi način rada sa Entity frameworkom je „baza prva“ pristup koji se koristi s već postojećom bazom podataka na temelju koje se stvaraju modeli podataka u kodu, dok je drugi način rada da se

najprije stvori modele u kodu na temelju kojih će Entity Framework stvoriti bazu podataka.

4 Implementacija

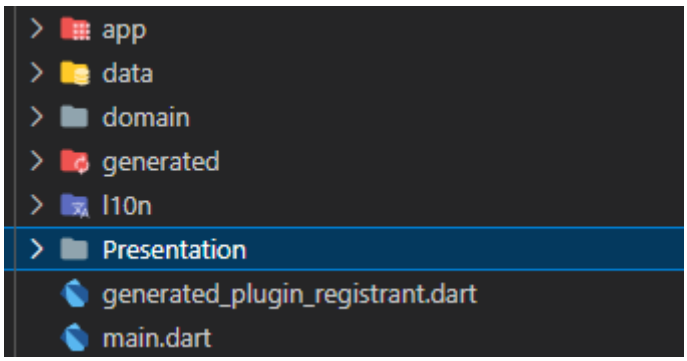
4.1 Frontend web klijent

Frontend web klijent je dio projekta koji je u interakciji sa korisnikom. Korisnik započinje interakciju dolaskom na web stranicu u svom web pregledniku.

4.1.1 Arhitektura

Projekt je podijeljen u četiri glavna dijela (foldera) kao što je prikazano na slici 5:

- App
- Data
- Domain
- Presentation



Slika 5. Pregled arhitekture web klijenta

App folder sadrži konfiguracije i klase koje se odnose na cijelog klijenta. Data folder sastoji se od cijelog koda koji služi za komunikaciju između klijenta i api-a servera. Domain folder uključuje modele projekta i poslovnu logiku. Presentation folder inkapsulira foldere od svake stranice u kojim je smješten View i ViewModel.

4.1.2 MyApp

MyApp je jedna od prvih klasa koje se pokreću u projektu. U MyApp klasi stvara se prvi widget aplikacije koji će služiti kao roditelj svim ostalim widget-ima (vidi sliku 6).

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

import '../presentation/Resources/routes_manager.dart';
import '../presentation/Resources/theme_manager.dart';

// ignore: must_be_immutable
class MyApp extends StatefulWidget {
  int appState = 0;
  MyApp._internal();

  static final MyApp instance = MyApp._internal();

  factory MyApp() => instance;

  @override
  // ignore: library_private_types_in_public_api
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      onGenerateRoute: RouteGenerator.getRoute,
      initialRoute: Routes.splashRoute,
      theme: getApplicationTheme(),
    );
  }
}
```

Slika 6. Klasa MyApp – izrada početkog widget-a

„Flutter widget-i su sagrađeni koristeći moderni programski okvir koji uzima inspiraciju od React-a. Centralna ideja je da sagradiš vlastiti UI koristeći widget-e. Widget-i opisuju kako će njihov pogled izgledati u danoj konfiguraciji i stanju.“ [10].

Unutar widgeta postavljen je MaterialApp widget unutar kojeg su postavljene sljedeće postavke:

- onGenerateRoute: u kojoj je postavljena metoda koja će se izvršavati prije svake promjene putanje
- initialRoute: je postavljena prva putanja koja se učitava nakon inicijalizacije MaterialApp-a.

- theme: je zadana default tema koja implicitno određuje vizualne aspekte aplikacije.

4.1.3 Promjena putanje i Dependency Injection

Prilikom poziva promjene putanje (stranice) poziva se metoda getRoute() (slika 7) sa parametrom tipa RouteSettings koji u sebi sadrži ime putanje i objekt koji se treba poslati u metodu na koju putanja pokazuje .

```
class RouteGenerator {
  static Route<dynamic> getRoute(RouteSettings routeSettings) {
    switch (routeSettings.name) {
      case Routes.splashRoute:
        initSplashModule();
        return MaterialPageRoute(builder: (_) => SplashView());    Prefer const with constant
      case Routes.mainPageRoute:
        initMainPageModule();
        return MaterialPageRoute(
            builder: (_) =>
                MainPage(listAd: routeSettings.arguments as List<Ad>)); // MaterialPageRoute
      case Routes.adRoute:
        initAdModule();
        return MaterialPageRoute(
          builder: (_) => presentationAd.Ad(
              mainPageAd: routeSettings.arguments as MainPageAd), // presentationAd.Ad
        ); // MaterialPageRoute
      default:
        return unDefinedRoute();
    }
  }
```

Slika 7. Klasa RouteGenerator – inicijalizacija rute

Metoda GetRoute() pronalazi slučaj za proslijeđenu putanju koja je spremljena u šifrarniku u kojoj se izvršava inicijalizacija ovisnosti i pokreće konstruktor od metode na koju pokazuje putanja.

Inicijalno zadavanje ruta možemo vidjeti na slici 8.

```
class Routes {
  static const String splashRoute = '/';
  static const String mainRoute = '/main';
  static const String mainPageRoute = '/mainpage';
  static const String adRoute = '/ad';
}
```

Slika 8.Klasa Routes – Imenovanje putanja

Na slici 9 vidi se metoda za Dependency Injection (DI).

```
ture<void> initAppModul() async {
  //shared prefs instance
  final sharedPrefs = await SharedPreferences.getInstance();
  instance.registerLazySingleton<SharedPreferences>(() => sharedPrefs);

  //App prefes instance
  instance
      .registerLazySingleton<AppPreferances>(() => AppPreferances(instance()));

  // network info

  //dio factory
  instance.registerLazySingleton<DioFactory>(() => DioFactory(instance()));

  // app service client

  final dio = await instance<DioFactory>().getDio();
  instance.registerLazySingleton<AppServiceClient>(() => AppServiceClient(dio));

  //remote data source
  instance.registerLazySingleton<RemoteDataSource>(
      () => RemoteDataSourceImplementer(instance()));

  // repository
  instance.registerLazySingleton<Repository>(() => RepositoryImpl(instance()));
}
```

Slika 9. Metoda initAppModul - Izrada početnog Dependency Injection

„Dependency Injection (DI) je dizajn obrazac koji se koristi za implementaciju Inverzije kontrole. Omogućuje stvaranje zavisnih objekata izvan klase i daje te objekte klasama na različite načine. Koristeći DI, pomičemo stvaranje i vezanje zavisnih objekata izvan klase koja o njima ovisi. To donosi višu razinu fleksibilnosti, odvajanja i lakšeg testiranja.“ [11].

Metoda initAppModul poziva se prilikom pokretanja aplikacije i u sebi sadrži objekte koje se koriste kroz cijelu aplikaciju, dok se ostatak zavisnih objekata dodaje po potrebi prilikom prvog ulaska u putanju.

4.1.4 Stranica učitavanja web aplikacije

Stranica učitavanja (eng. Splash screen) prikazana na slici 10, se prikazuje prilikom početnog dohvata podataka o oglasima.

Slika 10. Izgled stranice učitavanja

Prilikom inicijalizacije poziva se metoda _start() koja u ViewModel-u poziva metodu za dohvat podataka i čeka završetak. Nakon završetka poziva izvršava se metoda za usmjeravanje na glavnu stranicu sa referencom na objekt koji sadrži listu oglasa prikazano na slici 11.

```
class _SplashViewState extends State<SplashView> {
  final SplashViewModel _viewModel = instance<SplashViewModel>();

  //AppPreferances _appPreferances = instance<AppPreferances>();
  _start() async {
    await _viewModel.getData();

    Navigator.pushReplacementNamed(context, Routes.mainPageRoute,
        arguments: _viewModel.adBodyList);    Do not use BuildContex
  }

@override
  void initState() {
    super.initState();
    _start();
```

Slika 11. Prikaz inicijalizacije stranice učitavanja

ViewModel (slika 12) sadrži metodu getData koja je asinkrona i služi za poziv dohvata podataka iz backend api-ja i prikaz greške ukoliko dohvat je neuspješan.

```
class SplashViewModel extends BaseViewModel
    with SplashViewModelInputs, SplashViewModelOutputs {
  GetAdListUseCase _getAdListUseCase;    Private field could be final.

  SplashViewModel(this._getAdListUseCase);

  @override
  void start() async {}

  List<Ad> adBodyList = <Ad>[];
  getData() async {
    (await _getAdListUseCase.execute(null)).fold((failure) {
      inputState.add(ErrorState(
          StateRendererType.FULL_SCREEN_ERROR_STATE, failure.message));
    }, (listaAdBody) {
      inputState.add(ContentState());
      adBodyList = listaAdBody;
    });
  }
}
```

Slika 12. Prikaz ViewModel-a stranice učitavanja

4.1.5 Dohvat i slanje na backend

Dohvat i slanje podataka počinju iz ViewModel-a pozivom usecase-a pripadajućeg api. Na slici 13 je prikazana metoda koja sadrži poziv usecase metode. Prilikom povratka iz usecase metode provjerava da li se desila greška, te ukoliko nije zapisuju se rezultati u listu. Ukoliko ista pronađe grešku ona se ispisuje na ekran.

```
List<Ad> adBodyList = <Ad>[];
getData() async {
  (await _getAdListUseCase.execute(null)).fold((failure) {
    inputState.add(ErrorState(
        StateRendererType.FULL_SCREEN_ERROR_STATE, failure.message));
  }, (listaAdBody) {
    inputState.add(ContentState());
    adBodyList = listaAdBody;
  });
}
```

Slika 13. Metoda getData – dohvat oglasa

Na slici 14 vidimo primjer usecase-a koji u ovom projektu služi samo za poziv repository_implementator naredbi ali je isti napravljen radi preglednijeg koda zbog eventualnog proširenja ovog projekta. U budućnosti će usecase metode služiti za mapiranje input varijabli u objekt, a isti će se prosljeđivati u naredbu repository_implementator-a.

```
class GetAdListUseCase extends BaseUseCase<void, List<Ad>> {
  Repository _repository;    Private field could be final.

  GetAdListUseCase(this._repository);

  @override
  Future<Either<Failure, List<Ad>>> execute(void input) async {
    return await _repository.GetAdList();
  }
}
```

Slika 14. Klasa GetAdListUseCase

Metode u repository_implementator-u koriste se za poziv metoda iz klase RemoteDataSourceImplementer prikazano na slici 15. Po povratku rezultata iz api-ja provjerava se koji je status vratio. Ukoliko kao status za dohvat ili slanje nismo dobili vrijednost „uspješan“, na ekranu se prikazuje vraćena poruka greške, a ako api nije vratio poruku ili status ispisuju se njihove default-ne vrijednosti. U slučaju da je odgovor uspješan rezultat se mapira u modele klijenta uz pomoć metode napisane u datoteci mapper.dart.

```dart
@override
Future<Either<Failure, List<Ad>>> GetAdList() async {    Name non-constant id
  try {
    // its safe to call the API
    final response = await _remoteDataSource.getAdList();

    if (response.status == ApiInternalStatus.SUCESS) // success
    {
      return Right(response.toDomain());
    } else {
      // return biz logic error
      // return left
      return Left(Failure(response.status ?? ApiInternalStatus.FAILURE,
          response.message ?? ResponseMessage.DEFAULT)); // Failure // Left
    }
  } catch (error) {
    return (Left(ErrorHandler.Handle(error).failure));
  }
}
```

Slika 15. Metoda GetAdList

Na slici 16. vidljiv je poziv metode toDomain() na varijabli response koja je tipa adListResponse. Tip adListResponse je objekt prijenosa podataka (DTO).

```dart
extension AdListResponseMapper on AdListResponse? {
  toDomain() {
    var b = <Ad>[];

    this?.listadResponse?.forEach((element) {
      b.add(element.toDomain());
    });

    return b;
  }
}
```

Slika 16. AdListResponseMapper

Objekt prijenosa podataka je objekt koji inkapsulira podatke iz različitih modela u jedan objekt koji sadrži samo podatke koji su potrebni za slanje [12][13][14].

Objekti prijenosa podataka u ovom projektu dodatno su korišteni kako bi se podijelio data sloj od domain sloja.

Metode toDomain() su mapper-i koji mapiraju objekte prijenosa podataka u modele podataka korištene u klijentu. Na slici 15. prikazan je poziv mapper-a toDomain() na

varijabli tipa AdListResponse. Mapiranje na slici 16 prikazuje način rada gore navedene metode na način da se iterira lista adResponse te se za svaku iteraciju adResponse poziva toDomain mapper.

Unutar mapper-a za tip AdResponse na slici 17 vidimo da svaki objekt koji taj mapper sadržava imaju poziv na mapper za taj objekt.

```
extension AdResponseMapper on AdResponse? {
  Ad toDomain() {
    return Ad(
        this?.adHead.toDomain() ?? AdHead.empty(),
        this?.adBody.toDomain(),
        this?.employer.toDomain() ?? Employer.empty(),
        this?.categoryList.toDomain()); // Ad
  }
}
```

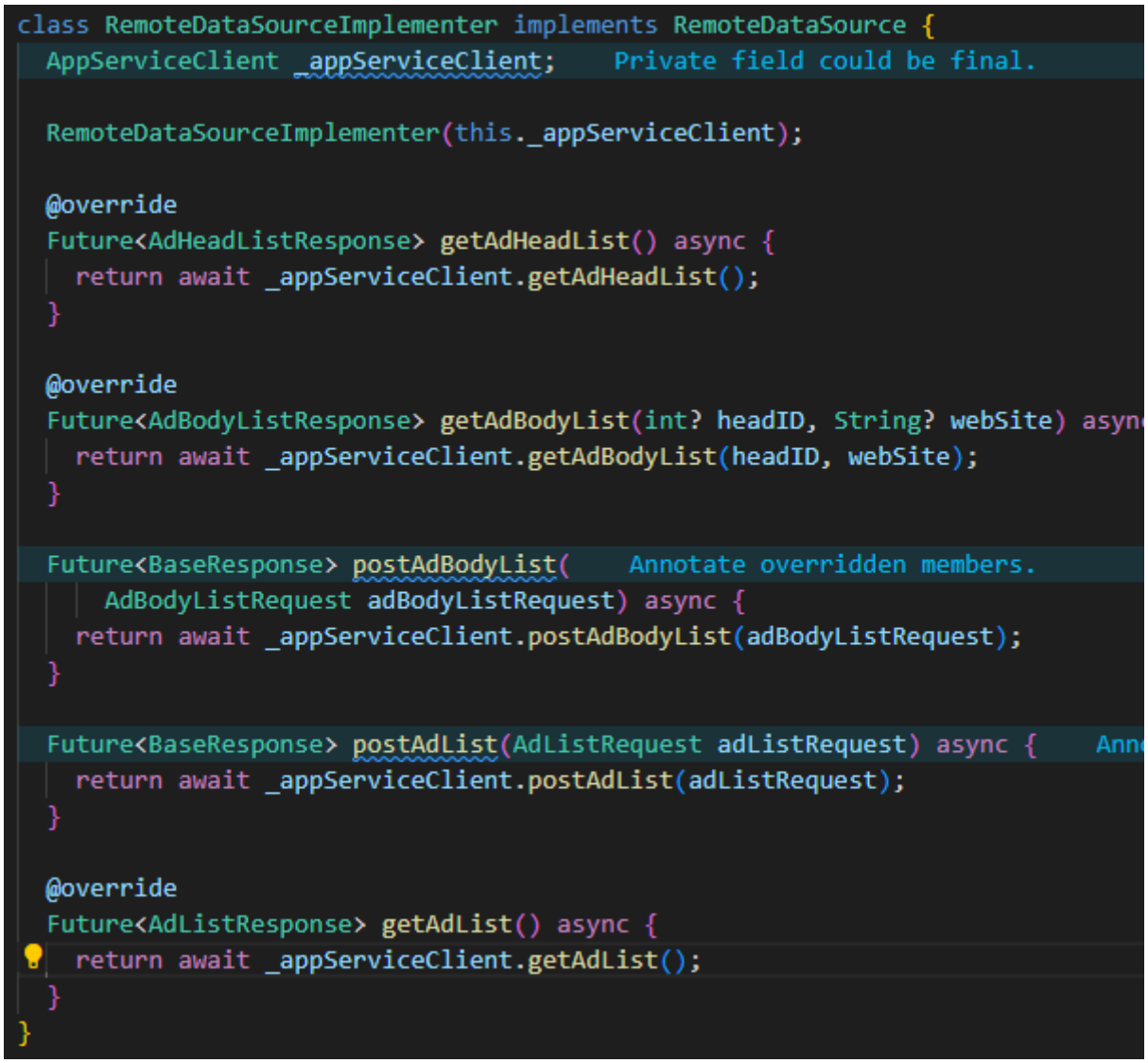
Slika 17. AdResponseMapper

Jedan od objekta koji se mapira je adHead, a njegov mapper prikazan je na slici 18. Za mapper adHeadResponseMapper vidljivo je da se svaka varijabla prosljeđuje u konstruktor modela, a prilikom prosljeđivanja varijabli provjerava se da li je nedozvoljenog tipa te ukoliko se utvrdi da je ista nedozvoljenog tipa postavlja se u default vrijednost.

```
extension AdHeadResponseMapper on AdHeadResponse? {
  AdHead toDomain() {
    return AdHead(
        this?.ID?.orZero() ?? ZERO,
        this?.title.orEmpty() ?? EMPTY,
        this?.employmentID?.orZero() ?? ZERO,
        this?.location.orEmpty() ?? EMPTY,
        this?.applicationDeadline.orEmpty() ?? FIRST,
        this?.county.orEmpty() ?? EMPTY);
  }
}
```

Slika 18. AdHeadResponeMapper

Podaci koji su proslijeđeni mapper-u dobiveni su pozivom metoda iz klase RemoteDataSourceImplementer. Na slici 19 prikazana je metoda getAdList() koja kao i sve ostale metode koje se pozivaju u RemoteDataSourceImplementer su generirane pomoću paketa Retrofit Generator [15].



Slika 19. Klasa RemoteDataSoureImplementer

Metode se generiranju na osnovu koda napisanog u datoteci app_api.dat vidljivog na slici 20. Retrofit Generator za generiranje metoda zahtijeva anotacije, a neke od anotacija su:

- @RestApi specificira da se klasa koristi za generiranje RestApi-ja kao parametar prima web putanju na api-ja,
- @GET specificira putanju od http metode za dohvat zapisa sa api-ja,
- @POST specificira putanju od http metode za kreiranje zapisa na api-ju,
- @PUT specificira putanju od http metode za ažuriranje zapisa na api-ju,
- @DELETE specificira putanju od http metode za brisanje zapisa na api-ju.

Linija koda part 'app_api.g.dart' proširuje datoteku sa generiranom datotekom.

```dart
import 'package:dio/dio.dart';
import 'package:wen_scraping/app/constant.dart';
import 'package:retrofit/http.dart';
import 'package:wen_scraping/data/request/request.dart';
import 'package:wen_scraping/data/responses/responses.dart';

part 'app_api.g.dart';

@RestApi(baseUrl: Constant.baseUrl)
abstract class AppServiceClient {
  factory AppServiceClient(Dio dio, {String baseUrl}) = _AppServiceClient;

  @POST("/AdBody/PostList")
  Future<BaseResponse> postAdBodyList(
      @Body() AdBodyListRequest? adBodyListRequest);

  @POST("/Ad/PostList")
  Future<BaseResponse> postAdList(@Body() AdListRequest? adListRequest);

  @GET("/AdBody")
  Future<AdBodyListResponse> getAdBodyList(
      @Query("headID") int? headID, @Query("website") String? webSite);

  @GET("/AdHead")
  Future<AdHeadListResponse> getAdHeadList();

  @GET("/Ad")
  Future<AdListResponse> getAdList();
}
```

Slika 20. Klasa AppServiceClient

Na slici 21 prikazan je primjer jedne od generiranih metoda. Metoda getAdList() koristi klasu Dio koja je implementirana u Dio paketu. Metoda getAdList() instancu Dio klase dobiva kroz konstruktor od klase AppServiceClient. Klasa AppServiceClient je dobila instancu uz pomoć dependency injection-a koji se izvršio prilikom pokretanja aplikacije. Dependency injection poziva metodu getDio() koja se nalazi u klasi DioFactory.

```dart
@override
Future<AdListResponse> getAdList() async {
  const _extra = <String, dynamic>{};
  final queryParameters = <String, dynamic>{};
  final _headers = <String, dynamic>{};
  final _data = <String, dynamic>{};
  final _result = await _dio.fetch<Map<String, dynamic>>(
      _setStreamType<AdListResponse>(
          Options(method: 'GET', headers: _headers, extra: _extra)
              .compose(_dio.options, '/Ad',
                  queryParameters: queryParameters, data: _data)
              .copyWith(baseUrl: baseUrl ?? _dio.options.baseUrl)));
  final value = AdListResponse.fromJson(_result.data!);
  return value;
}

RequestOptions _setStreamType<T>(RequestOptions requestOptions) {
  if (T != dynamic &&
      !(requestOptions.responseType == ResponseType.bytes ||
          requestOptions.responseType == ResponseType.stream)) {
    if (T == String) {
      requestOptions.responseType = ResponseType.plain;
    } else {
      requestOptions.responseType = ResponseType.json;
    }
  }
  return requestOptions;
}
}
```

Slika 21. Metoda getAdList

Metoda getDio() (slika 22) stvara instancu Dio klase i za tu instancu se postavljaju sljedeće postavke:

- content-type,
- accept,
- language,
- connectTimeout,
- receiveTimeout.

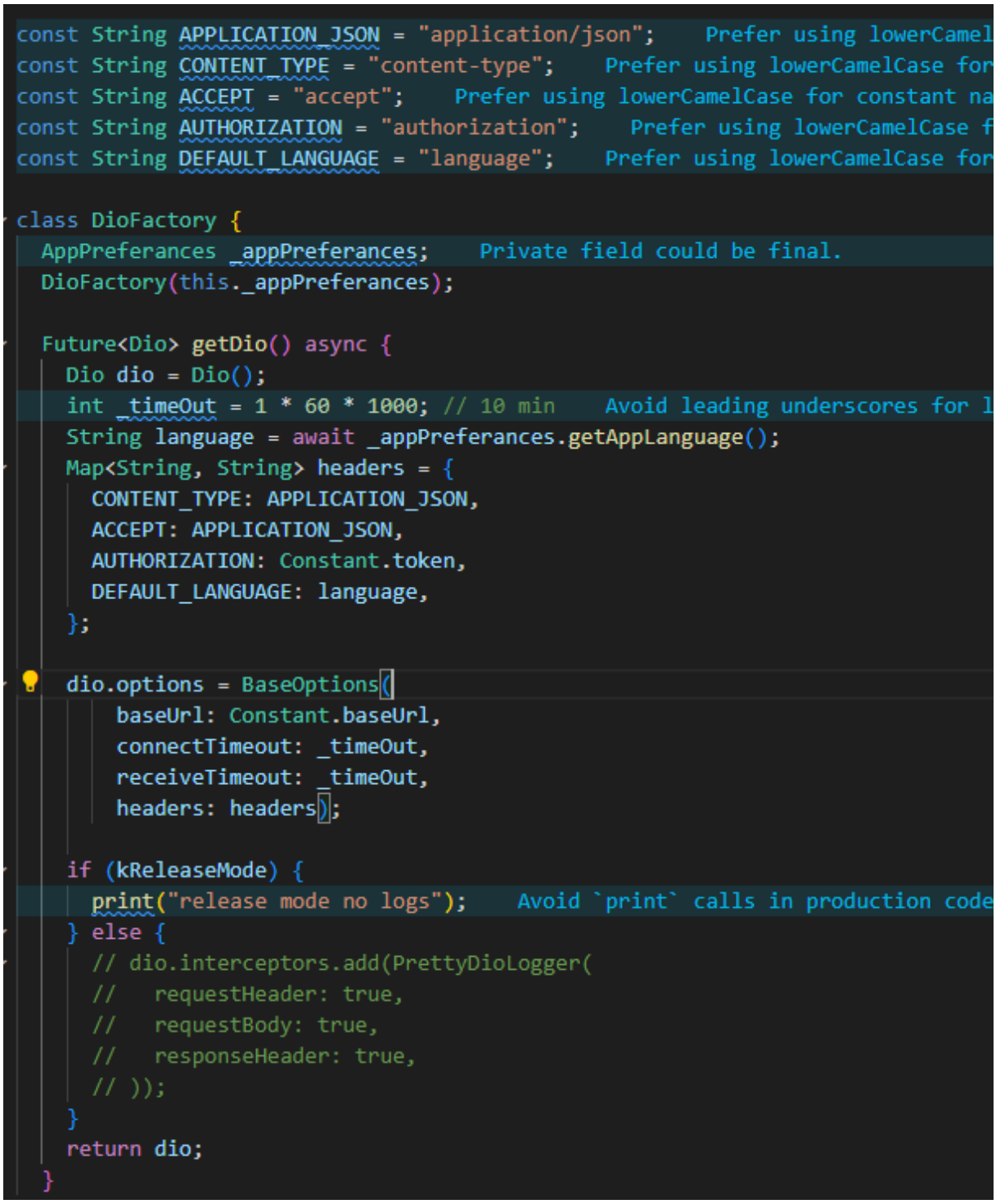
Content-type i accept specificiraju da će se koristit json datotečni format za razmjenu podataka između klijenta i api-ja.

Language specificira da će se koristiti hrvatska abeceda kako bi se mogli slati znakovi čćžšđ. Vrijednost jezika se dohvaća iz AppPreferances koja se inicijalizira prilikom pokretanja aplikacije. Postavke aplikacije su za sad postavljene isključivo na hrvatski, ali omogućeno je njezino proširenje tako da podržava i ostale jezike ukoliko se napravi izbor određenog jezika na ekranu.

Connectiontimeout specificira koliko će se dugo čekati na dobivanje odgovora od api-ja prilikom uspostave veze, postavka je postavljena da čeka jednu minutu.

Receive Timeout specificira koliko će se dugo čekati na dobivanje odgovora na postavljeni zahtjev, a isti je također postavljen da čeka jednu minutu.

Nakon specificiranja postavki instanca Dio objekta se prosljeđuje uz pomoć dependency injection-a kroz aplikaciju. U datoteci app_api.g.dart koristi se ta instanca Dio objekta za komunikaciju sa backend api-jem. Retrofit Generator na osnovi anotacije generira dodatne Dio opcije koje su među ostalima HTTP metoda i putanja do api metode. U ovom projektu zbog prije navedenih postavki Dio instanca razmjenjuje zahtjeve i odgovore u JSON obliku. Opisano je kako se pomoću mapper-a pretvara model u zahtjev ili odgovor u model ali nije opisan izgled zahtjeva ili odgovor i njihovih metoda među kojima su i metode za pretvorbu u JSON i iz JSON-a.



Slika 22. Klasa DioFactory

U zahtjevima i odgovorima koristi se JSON Serializable paket kako bi uz pomoć anotacije mogle generirati metode za pretvorbu zahtjeva i odgovora u JSON format. Uz pomoć JSONKey anotacije dodjeljujemo varijabli ime koje će se koristi u JSON-u a uz pomoć JSONSerializable anotacije specificiramo za koje klase se generiraju metode vidljivo na slici 23.

```dart
@JsonSerializable()
class AdResponse {
  @JsonKey(name: 'adHead')
  AdHeadResponse? adHead;

  @JsonKey(name: 'adBodyList')
  List<AdBodyResponse>? adBody;

  @JsonKey(name: 'employer')
  EmployerResponse? employer;

  @JsonKey(name: 'categoryList')
  List<CategoryResponse>? categoryList;

  AdResponse(this.adHead, this.adBody, this.employer, this.categoryList);

  factory AdResponse.fromJson(Map<String, dynamic> json) =>
      _$AdResponseFromJson(json);

  Map<String, dynamic> toJson() => _$AdResponseToJson(this);
}

@JsonSerializable()
class AdListResponse extends BaseResponse {
  @JsonKey(name: 'listAd')
  List<AdResponse>? listadResponse;

  AdListResponse(this.listadResponse);

  factory AdListResponse.fromJson(Map<String, dynamic> json) =>
      _$AdListResponseFromJson(json);

  Map<String, dynamic> toJson() => _$AdListResponseToJson(this);    Annotate
}
```

Slika 23. Klasa Response

Usporedivši sliku 23 sa slikom 24 vidljivo je da je slična sintaksa zahtjeva i odgovora. Jedina razlika je u smjeru kretnje objekta budući da zahtjev putuje prema api-ju dok se odgovor vraća iz api-ja.

```dart
@JsonSerializable()
class AdBodyRequest {
  @JsonKey(name: 'id')
  int? ID;    Name non-constant identifiers using lowerCamelCase.

  @JsonKey(name: 'headID')
  int? headID;

  @JsonKey(name: 'body')
  String? body;

  @JsonKey(name: 'website')
  String? website;

  @JsonKey(name: 'url')
  String? url;

  @JsonKey(name: 'publishDate')
  DateTime publishDate;

  AdBodyRequest(this.ID, this.headID, this.body, this.website, this.url,
      this.publishDate);

  factory AdBodyRequest.fromJson(Map<String, dynamic> json) =>
      _$AdBodyRequestFromJson(json);

  Map<String, dynamic> toJson() => _$AdBodyRequestToJson(this);
}
```

Slika 24. Klasa Request

4.1.6 Glavna stranica aplikacije

Glavna stranica je prikazana na slici 25. Ona je najbitniji dio aplikacije budući da na njoj korisnik pretražuje oglase kako bi pronašao one koji ga interesiraju. Korisnik na istoj može pretraživati oglase po županiji, po djelatnosti, po mjestu kao i po riječi iz oglasa. Na primjer odabirom županije npr. Istarska županija nude se poslovi samo za tu županiju dok su poslovi za ostale županije izostavljeni i ne prikazuju se.

Slika 25. Izgled glavne stranice

Prilikom inicijalizacije stranice poziva se metoda sortAds() (slika 26) kojom se preslažu oglasi počevši od najnovijeg oglasa sve do najstarijeg oglasa te se isti prikazuju na stranici.

```
void sortAds(List<Ad> adList) {
  adList.sort(((a, b) =>
      b.adBody.first.publishDate.compareTo(a.adBody.first.publishDate)));
  adHead = adList.map((e) => e.adHead).toList();
}
```

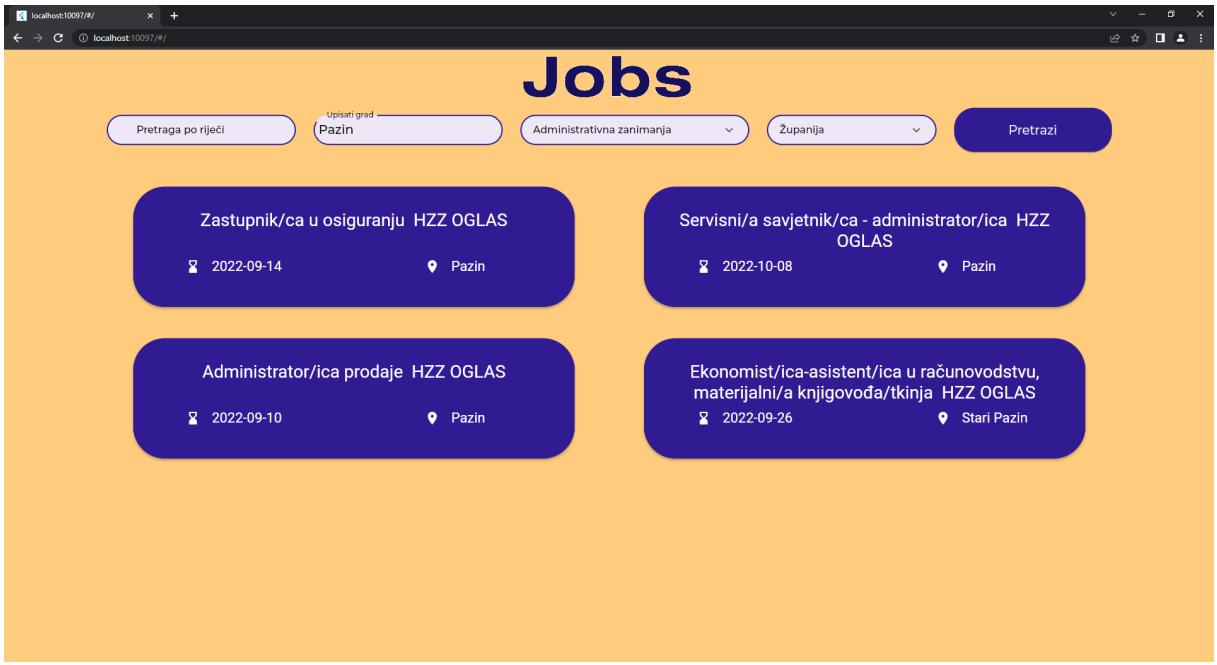
Slika 26.Metoda sortAds

Pritiskom na gumb poziva se metoda filterAds() koja iterira kroz sve oglase i prikazuje samo one oglase koji odgovaraju izabranim uvjetima. Metoda filtriranja vidljiva je na slici 27.

```
void filterAds(List<Ad> adList) {
  adHead = adList
      .where((element) =>
          (zupanijeDropdownValue == "Županje" ||
              element.adHead.county == zupanijeDropdownValue) &&
          (djelatnostiDropdownValue == "Djelatnosti" ||
              element.categoryList.any(
                  (element) => element.name == djelatnostiDropdownValue)) &&
          (search == "" ||
              element.adBody
                  .any((element) => element.body.contains(search))) &&
          (searchCity == "" ||
              element.adHead.location
                  .toLowerCase()
                  .contains(searchCity.toLowerCase())))
      .map((e) => e.adHead)
      .toList();
}
```

Slika 27. Metoda filterAds

Prikaz oglasa nakon filtriranja vidljiv je na slici 28.

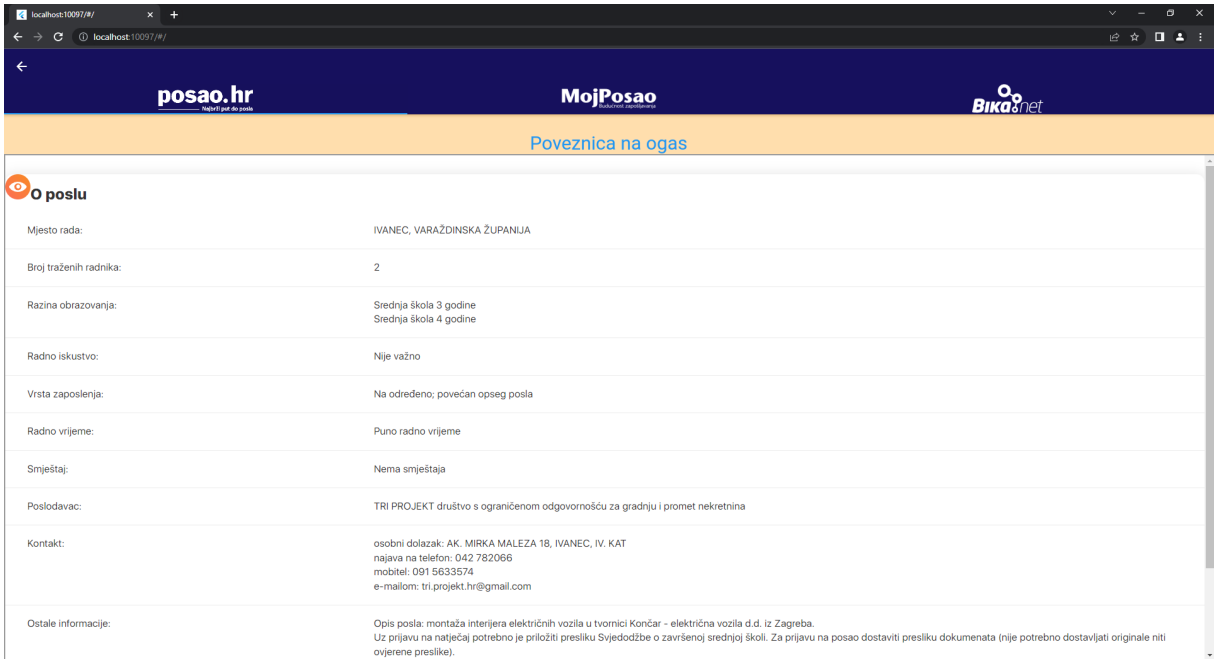


Slika 28. Prikaz oglasa na filtriranja

Pritiskom na karticu oglasa pokreće se metoda navigate() koja služi za usmjeravanje na stranicu oglasa.

4.1.7. Stranica oglasa

Stranica oglasa prikazana na slici 29 sastoji se od tri taba od kojih svaki predstavlja jednu web stranicu. U svakom tabu prikazan je sadržaj oglasa s te web stranice. U slučaju da isti oglas postoji na sve tri web stranice, sva tri taba će biti popunjena sadržajem tog oglasa. Ukoliko traženi oglas ne postoji na nekom od ta tri taba isti se neće popuniti, polje će ostati prazno. Na primjer, ukoliko traženi oglas postoji na web stranici posao.hr i na web stranici Moj-posao.net ta dva taba će se popuniti dok će treći tab Bika.net ostati prazan budući da taj oglas na toj web stranici ne postoji.



Slika 29. Izgled web stranice oglasa

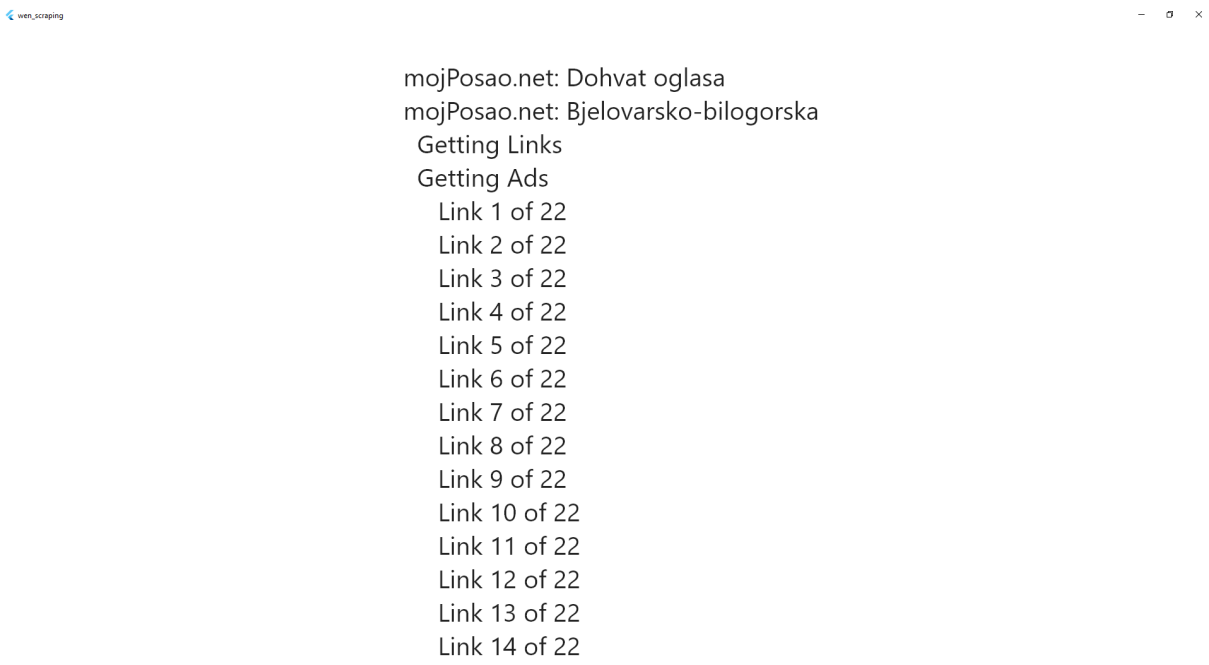
4.2. Frontend web scraping klijent

Frontend web scraping klijent dio je projekta koji služi za dohvat podataka sa različitih web stranica. Uz pomoć paketa Puppeteer vrši se dohvat sadržaja web stranice te se iz istog povlače podaci i šalju na api backend servera kako bi se isti spremili u bazu podataka.

4.2.1. Stranica učitavanja web scraping aplikacije

Nakon inicijalizacije web scraping-a aplikacije prva stranica koja se prikazuje je stranica učitavanja. Stranica učitavanja stvara instancu chromium preglednika te se nakon toga poziva dohvat svih oglasa iz api-ja koji se onda zajedno sa instancom preglednika prosljeđuje metodi GetAllCounties(). Metoda GetAllCounties() vraća listu oglasa koji nisu pohranjeni u bazi te se ta lista prosljeđuje metodi za slanje na api. Na

identičan način nakon toga se provodi isti postupak i za sljedeće web stranice moj-posao.net i bika.net. Napredak kroz svaki dohvat oglasa prikazuje se na stranici učitavanja a što je prikazano na slici 30.

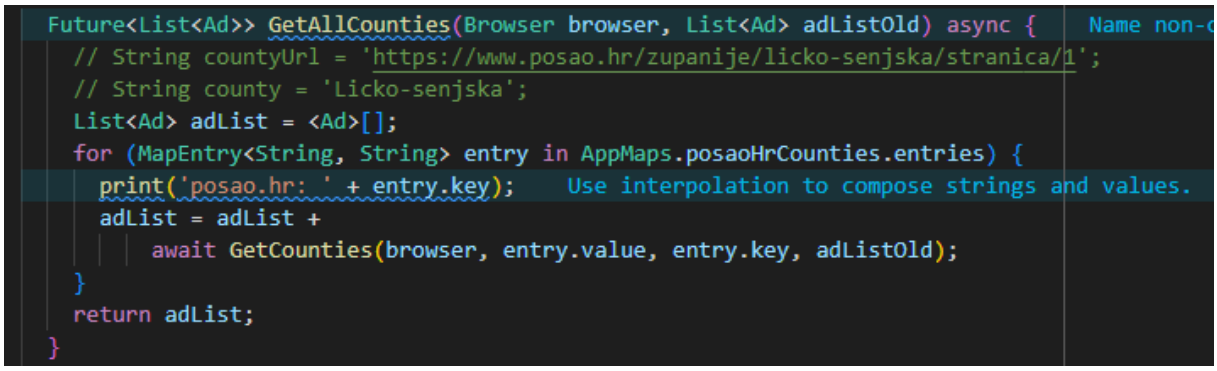


Slika 30. Prikaz stranice učitavanja

4.2.2. Klasa Posaohr

Klasa Posaohr sadrži metode potrebne za dohvat i mapiranje podataka u modele tipa ad. Prva metoda koja se poziva je GetAllCounties() prikazana na slici 31. Pomoću te metode iteriramo kroz mapu županija te za svaku iteraciju pozivamo metodu GetCounty() kojoj prosljeđujemo sljedeće parametre:

- Instancu chromium preglednika,
- Link na prvu stranicu oglasa trenutne županije,
- Naziv trenutne županije,
- Listu svih oglasa iz baze podataka.



Slika 31. Dohvat županija

Metoda GetCounty() prikazana na slici 32, poziva metodu za dohvat broja stranica oglasa za trenutnu županiju. Nakon što povratno ista vrati broj stranica filtrira se lista oglasa pristigle iz baze podataka kako bi dobili listu poveznica oglasa. Nakon završetka filtriranja iterira se kroz sve stranice trenutne županije. U svakoj iteraciji dohvaća se lista poveznica za sve oglase na toj web stranici uz pomoć metode GetOnePageAdUrls() te se odmah vrši i provjera da li su dohvaćene poveznice oglasa već ranije spremljena u bazu podataka. Ukoliko je neka od poveznica već ranije pohranjena u bazi podataka tada se ista izbacuje iz liste. Po završetku prolaska kroz sve web stranice lista poveznica za pohranu u bazu podataka prosljeđuje se metodi GetAds().

```
Future<List<Ad>> GetCounty(    Name non-constant identifiers using lowerCamelCas
    Browser browser, String countyUrl, String county, List<Ad> adList) async {
  String siteUrlNoNumber = countyUrl.substring(0, countyUrl.length - 1);
  int brojStanica = await GetNumberOfPages(browser, countyUrl);
  Set AllUrlList = {};    Name non-constant identifiers using lowerCamelCase.
  List<String> temp = <String>[];
  adList.forEach((element) =>    Avoid using `forEach` with a function literal.
      element.adBodyList.forEach((element2) => temp.add(element2.url)));    Avoi
  AllUrlList = temp.toSet();
  List AllNewAdLink = List.empty(growable: true);    Name non-constant identifie
  print('Getting Links');    Avoid `print` calls in production code.
  for (int i = 1; i != brojStanica; i++) {
    var pageURLs =
        await GetOnePageAdUrls(browser, siteUrlNoNumber + i.toString());
    var newURLs = pageURLs.toSet().difference(AllUrlList);

    if (newURLs.isEmpty) {
      continue;
    } else {
      AllNewAdLink += newURLs.toList();
      AllUrlList.addAll(newURLs.toSet());
    }
  }
  return getAds(browser, AllNewAdLink, county, adList);
}
```

Slika 32. Dohvat županije

Slika 33 prikazuje Metodu GetOnePageAdUrls() služu za dohvat svih poveznica oglasa sa jedne stranice. Ona poziva metodu getLinksFromPage() koja vraća listu svih poveznica te se ta lista nakon toga filtrira da ostanu samo poveznice oglasa.

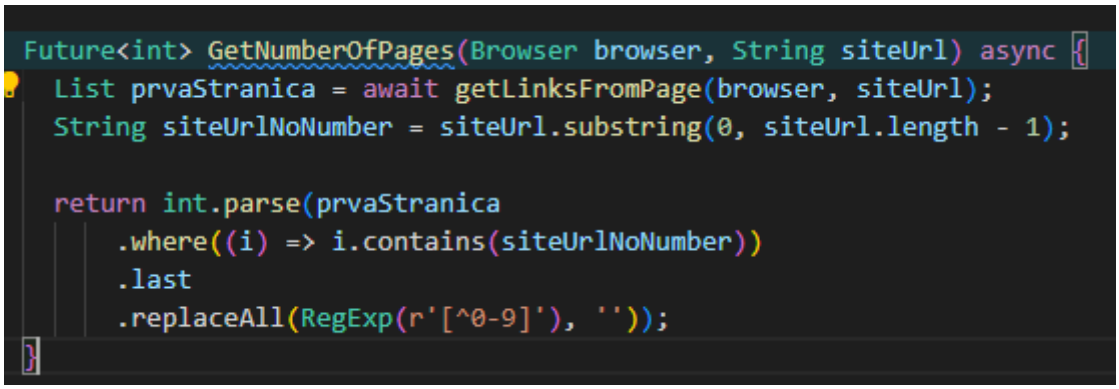
```
Future<List<String>> GetOnePageAdUrls(    Name non-constant identifiers using lowerCamelCase.
    Browser browser, String pocetnaStranica) async {
  List<String> links = await getLinksFromPage(browser, pocetnaStranica);
  return links.where((i) => i.contains('oglasi')).toList();
}
```

Slika 33. Dohvat poveznica oglasa sa jedne web stranice

GetLinksFromPage() metoda (vidljiva na slici 34) stvara novi objekt tipa Page koji je dio Puppeteer paketa. Koristeći taj objekt vrši se dohvat stranice te se iz dohvaćene stranice izdvajaju sve poveznice unutar specificirane regije HTML-a. Regija je

Slika 34. Dohvat svih poveznica unutar jedne regije HTML-a

Metoda getNumberOfPages() dohvaća sve poveznice jedne stranice koristeći metodu GetLinksFromPage() te izabire samo poveznice koje sadrže poveznice na stranice trenutne županije isto je prikazano na slici 35. Po izvršenom odabiru poveznica izabiru se samo brojevi i najveći broj među njima je vrijednost koju ova metoda vraća.



Slika 35. Dohvat broja stranica oglasa u jednoj županiji

Kroz listu poveznica svih novih oglasa iteriramo uz pomoć metode getAds() ta iteracija je prikazana na slici 36. U svakoj iteraciji poziva se metoda getAd() za trenutnu poveznicu oglasa te se rezultat te metode dodaje u listu novih oglasa. Po završetku iteracije kroz sve poveznice novih oglasa lista oglasa vraća se u metodu getData() gdje se ta lista šalje api-ju za spremanje u bazu podataka.

```
Future<List<Ad>> getAds(Browser browser, List allAdUrls, String county,
    List<Ad> adListOld) async {
  List<Ad> adList = <Ad>[];
  int urlCount = allAdUrls.length;
  int i = 0;
  print('Getting Ads');    Avoid `print` calls in production code.
  for (String url in allAdUrls) {
    i++;
    adList.add(await GetAd(browser, url, county, adListOld));
    print('Link: $i of $urlCount');    Avoid `print` calls in production code.
  }
  print('Done Getting Ads');    Avoid `print` calls in production code.
  return adList;
}
```

Slika 36. Prolazak kroz sve poveznice novih oglasa

Za dohvat podataka svakog novog oglasa zadužena je metoda getAd(). Metoda getAd() u klasi posaoHr poziva slijedeće dohvate:

- getAdHead() koji je zadužen za dohvat podataka o informacijama oglasa,
- getAdBody() zadužen je za dohvat sadržaja oglasa,
- getTitleAd() dohvaća naslov oglasa.

Po izvršenom dohvatu gore navedenih podataka izvršava se provjera da li postoje slični oglasi u bazi podataka. Provjera se izvršava po županiji, nazivu oglašivača i naslovu oglasa. Nakon izvršene provjere ukoliko postoji slični oglas njegov identifikacijski broj prepisuje se preko identifikacijskog broja trenutnog oglasa. Po završetku podaci se prosljeđuju konstruktorima modela adHead, adBody, employer, categoryList te se instance tih modela prosljeđuju konstruktoru modela oglasa ad. Model oglasa ad se vraća kao rezultat ove metode.

Metoda updatePublishDate() izvodi se samo prilikom prvog punjenja baze sa podacima od web stranice posao.hr. Metoda služi za dohvat datuma objave oglasa a ista je napravljena budući da se u oglasu na navedenoj web stranici ne vidi datum objave oglasa pa je do navedenog moguće doći samo pomoću napredne tražilice. Napredna tražilica ima mogućnost prikaza svih oglasa koji su objavljeni prije određenog broja dana. Pomoću ove metode prolazimo kroz 120 dana počevši od oglasa koji su danas objavljeni. Nakon što se dohvate oglasi koji su danas objavljeni na web stranici posao.hr isti se dodaju u listu oglasa sa današnjim datumom. Sljedeći

dohvat sa web stranice posao.hr je za dva dana a to su današnji i jučerašnji datumi. Po izvršenom dohvatu oglasa brišu se oglasi koji su u prethodnoj iteraciji dohvaćeni, a oglasi koji preostanu njima se mijenja datum objave u jučerašnji datum te se njima ažurira datum objave u bazi podataka. U sljedećoj iteraciji to isto radimo ali za broj iteracija uvećano za jedan i brišemo oglase koji su dodani u ranijim iteracijama, a preostalima mijenjamo datum objave u današnji dan umanjeno za broj iteracija te se istima ažurira datum objave u bazi podataka. Isto je moguće vidjeti na slici 37.

```
Future<List<AdBody>> UpdatePublishDates(    Name non-constant identifiers using lowerCamelCase
    Browser browser, List<AdBody> adBodyList) async {
  DateTime now = DateTime.now();
  DateTime dateNow = DateTime(now.year, now.month, now.day);
  Set allUpdatedAdURLs = {};

  print('Updating Dates');    Avoid `print` calls in production code.
  for (int i = 0; i <= 120; i++) {
    int brojStanica = await GetNumberOfPages(browser,
        'https://www.posao.hr/poslovi/objavljeno-prije/$i/stranica/1');
    if (i != 0) {
      if (i.toString().length == 1) {
        brojStanica = brojStanica % 100;
      } else if (i.toString().length == 2) {
        brojStanica = brojStanica % 1000;
      } else if (i.toString().length == 3) {
        brojStanica = brojStanica % 10000;
      }
    }
    DateTime adsBefore =
        DateTime(dateNow.year, dateNow.month, dateNow.day - i);
    Set allNewAdURLs = {};

    print('Before $i');    Avoid `print` calls in production code.
    for (int j = brojStanica; j > 0; j--) {
      //print('Before $i: page: $j');
      var pageURLs = await GetOnePageAdUrls(browser,
          'https://www.posao.hr/poslovi/objavljeno-prije/$i/stranica/$j');
      Set newURLs = pageURLs.toSet().difference(allUpdatedAdURLs);

      if (newURLs.isEmpty) {
        //j = 2;
        print('Before $i: page: $j : empty');    Avoid `print` calls in production code.
      } else {
        allNewAdURLs.addAll(newURLs);
        print('Before $i: page: $j :not empty');    Avoid `print` calls in production code.
      }
    }
```

Slika 37. Izmjena datuma objave oglasa sa web stranice Posao.hr

4.2.3. Klasa MojPosaoNet

Klasa MojPosaoNet se razlikuje od posaoHr samo u nekoliko metoda te zbog toga navodimo samo razlike između njih. Jedna od razlika je u mapi županija a razlika je u putanji za svaku županiju. Razliku možemo vidjet prikazanu na sljedećim slikama (slika 38, 39) .

```
static const Map<String, String> posaoHrCounties = {
  'Bjelovarsko-bilogorska':
      'https://www.posao.hr/zupanije/bjelovarsko-bilogorska/stranica/1',
  'Brodsko-posavska':
      'https://www.posao.hr/zupanije/brodsko-posavska/stranica/1',
  'Dubrovacko-neretvanska':
      'https://www.posao.hr/zupanije/dubrovacko-neretvanska/stranica/1',
  'Istarska': 'https://www.posao.hr/zupanije/istarska/stranica/1',
  'Karlovacka': 'https://www.posao.hr/zupanije/karlovacka/stranica/1',
  'Koprivnicko-krizevacka':
      'https://www.posao.hr/zupanije/koprivnicko-krizevacka/stranica/1',
  'Krapinsko-zagorska':
      'https://www.posao.hr/zupanije/krapinsko-zagorska/stranica/1',
  'Licko-senjska': 'https://www.posao.hr/zupanije/licko-senjska/stranica/1',
  'Medimurska': 'https://www.posao.hr/zupanije/medimurska/stranica/1',
  'Osjecko-baranjska':
      'https://www.posao.hr/zupanije/osjecko-baranjska/stranica/1',
  'Pozesko-slavonska':
      'https://www.posao.hr/zupanije/pozesko-slavonska/stranica/1',
  'Primorsko-goranska':
      'https://www.posao.hr/zupanije/primorsko-goranska/stranica/1',
  'Sibensko-kninska':
      'https://www.posao.hr/zupanije/sibensko-kninska/stranica/1',
  'Sisacko-moslavacka':
      'https://www.posao.hr/zupanije/sisacko-moslavacka/stranica/1',
  'Splitsko-dalmatinsk':
      'https://www.posao.hr/zupanije/splitsko-dalmatinska/stranica/1',
  'Varazdinska': 'https://www.posao.hr/zupanije/varazdinska/stranica/1',
  'Viroviticko-podravska':
      'https://www.posao.hr/zupanije/viroviticko-podravska/stranica/1',
  'Vukovarsko-srijemska':
      'https://www.posao.hr/zupanije/vukovarsko-srijemska/stranica/1',
  'Zadarska': 'https://www.posao.hr/zupanije/zadarska/stranica/1',
  'Zagreb i Zagrebacka': 'https://www.posao.hr/zupanije/zagreb/stranica/1',
  'Inozemstvo': 'https://www.posao.hr/zupanije/inozemstvo/stranica/1'
};
```

Slika 38. Mapa županija web stranice Posao.Hr

```
static const Map<String, String> mojPosaoNetCounties = {
  'Bjelovarsko-bilogorska':
      'https://www.moj-posao.net/Zupanije/Bjelovarsko-bilogorska/?page=1',
  'Brodsko-posavska':
      'https://www.moj-posao.net/Zupanije/Brodsko-posavska/?page=1',
  'Dubrovacko-neretvanska':
      'https://www.moj-posao.net/Zupanije/Dubrovacko-neretvanska/?page=1',
  'Istarska': 'https://www.moj-posao.net/Zupanije/Istarska/?page=1',
  'Karlovacka': 'https://www.moj-posao.net/Zupanije/Karlovacka/?page=1',
  'Koprivnicko-krizevacka':
      'https://www.moj-posao.net/Zupanije/Koprivnicko-krizevacka/?page=1',
  'Krapinsko-zagorska':
      'https://www.moj-posao.net/Zupanije/Krapinsko-zagorska/?page=1',
  'Licko-senjska': 'https://www.moj-posao.net/Zupanije/Licko-senjska/?page=1',
  'Medimurska': 'https://www.moj-posao.net/Zupanije/Medimurska/?page=1',
  'Osjecko-baranjska':
      'https://www.moj-posao.net/Zupanije/Osjecko-baranjska/?page=1',
  'Pozesko-slavonska':
      'https://www.moj-posao.net/Zupanije/Pozesko-slavonska/?page=1',
  'Primorsko-goranska':
      'https://www.moj-posao.net/Zupanije/Primorsko-goranska/?page=1',
  'Sibensko-kninska':
      'https://www.moj-posao.net/Zupanije/Sibensko-kninska/?page=1',
  'Sisacko-moslavacka':
      'https://www.moj-posao.net/Zupanije/Sisacko-moslavacka/?page=1',
  'Splitsko-dalmatinsk':
      'https://www.moj-posao.net/Zupanije/Splitsko-dalmatinska/?page=1',
  'Varazdinska': 'https://www.moj-posao.net/Zupanije/Varazdinska/?page=1',
  'Viroviticko-podravska':
      'https://www.moj-posao.net/Zupanije/Viroviticko-podravska/?page=1',
  'Vukovarsko-srijemska':
      'https://www.moj-posao.net/Zupanije/Vukovarsko-srijemska/?page=1',
  'Zadarska': 'https://www.moj-posao.net/Zupanije/Zadarska/?page=1',
  'Zagreb i Zagrebacka':
      'https://www.moj-posao.net/Zupanije/Zagreb-i-zagrebacka/?page=1',
  'Inozemstvo': 'https://www.moj-posao.net/Zupanije/Inozemstvo/?page=1'
};
```

Slika 39. Mapa županija web stranice Moj-posao.net

Sljedeća razlika je u metodi getLinksFromPage() a razlika je u odabiru regije unutar HTML stranice. Razlog promjene odabira regije je različitost izgleda web stranica (slike 40 i 41).

```
const resultsSelector = '#main a[href]';
```

Slika 40. Odabir regije #main na web stranici Moj-posao.net

```
const resultsSelector = 'div.list.box a[href]';
```

Slika 41. Odabir regije box na web stranici Posao.hr

Jedna od razlika je u metodi getOnePageAdUrls() u riječi koja se traži unutar poveznice oglasa. Poveznice oglasa Posao.hr sadrže riječ „oglasi“ dok poveznice web stranice Moj-posao.net sadrže riječ „posao“ (slike 42 i 43).

```
return links.where((i) => i.contains('Posao')).toList();
```

Slika 42. Odabir poveznica koje sadrže riječ posao

```
return links.where((i) => i.contains('oglasi')).toList();
```

Slika 43. Odabir poveznica koje sadrže riječ oglasi

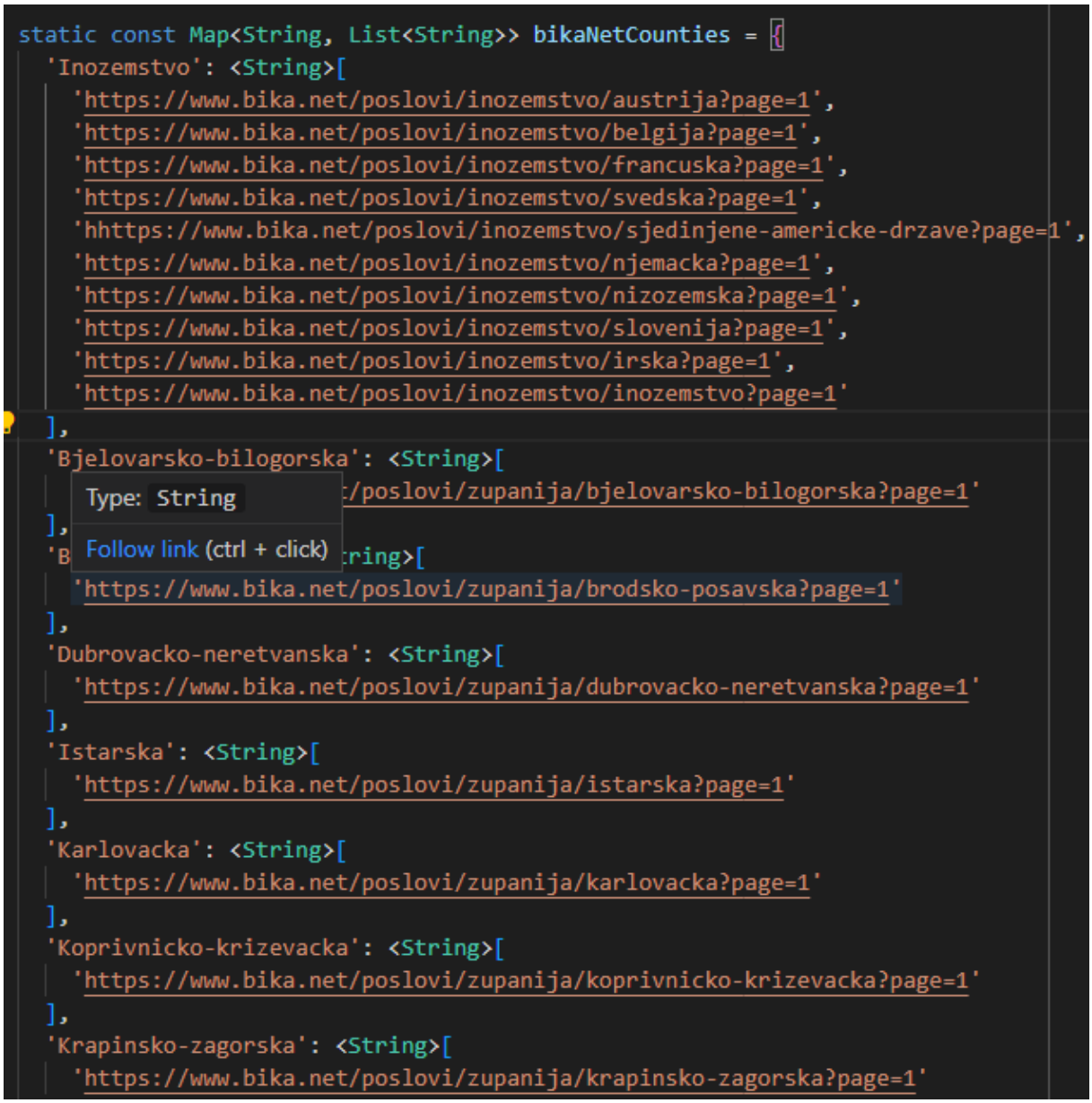
Preostale razlike su u dohvatu podataka iz oglasa budući da se izgled web stranice oglasa razlikuje. Na web stranici Posao.hr većina informacija o oglasu se nalazi u jednoj regiji HTML-a, dok na web stranici Moj-posao.net te iste informacije se nalaze u različitim regijama što iziskuje zasebni dohvat svake informacije pa je napravljena metoda za dohvat svake te informacije. Te metode su:

- GetAdAplicationDate() – dohvaća datum isteka oglasa pomoću selektora „#deadline > time“,
- GetAdEmployer() – dohvat oglašivača pomoću selektora „#ad-employer-info > ul > li“,
- GetAdLocation() – dohvat mjesta pomoću selektora „#job-detail > p.job-location“,
- GetAdPublishDate() – dohvat datuma objave oglasa pomoću selektora „#information > div > div:nth-child(1) > time“,
- GetAdCategory() – dohvat djelatnosti pomoću selektora „job-position > div > div:nth-child(1) > p.property-value“,
- GetAdBody() – dohvat sadržaja oglasa pomoću selektora „#job-standard“ ili selektora „#job-html“ a što ovisi o vrsti oglasa,
- GetTitleAd() – dohvat naslova oglasa pomoću selektora „#page-title“.

Oglas na web stranici Moj-posao.net sadrži informaciju o datumu objave oglasa pa nju dodatno i dohvaćamo. Zbog navedenog Moj-posao.net ne mora sadržavati metodu UpdatePublishDates().

4.2.4. Klasa BikaNet

Klasa BikaNet razlikuje se od MojPosaoNet u nekoliko metoda pa zbog toga navodimo samo bitne razlike između njih. Razlika je u mapi županija, a razlika je u putanji za svaku županiju. Tu razliku možemo vidjeti na slikama 39 i 44. Još jedna razlika u mapama je što se mapa bikaNetCounties sastoji od varijabli tipa String i tipa List<String>. Uvedena je lista budući da web stranica Bika.net dodatno razvrstava poslove u inozemstvu po pojedinim državama.



Slika 44. Mapa županija web stranice Bika.net

Jedna od sljedećih razlika je u metodi GetNumberOfPages() vidljivoj na slici 45, a koja je nastala zbog toga što web stranica Bika.net ne posjeduje poveznicu na zadnju stranicu, a ona nam je potrebna za dohvat broja stranica. Bika.net ima poveznice za pretragu jedanaest stranica, a pritiskom na gumb za jedanaestu stranicu otvara se jedanaesta stranica koja u sebi ima poveznice na slijedećih jedanaest stranica i tako ponovno do posljednje stranice liste oglasa. Iz tog razloga moramo učitati svaku jedanaestu stranicu kako bi utvrdili da li postoji još stranica. Dakle, nije moguće odmah vidjeti koja je zadnja stranica oglasa.

```
Future<int> GetNumberOfPages(Browser browser, String siteUrl) async {
  String siteUrlNoNumber = siteUrl.substring(0, siteUrl.length - 1);
  int maxPage = 0;
  int currentPage = 1;
  while (maxPage < currentPage) {
    maxPage = currentPage;
    List prvaStranica = await getLinksFromPage(
        browser, siteUrlNoNumber + currentPage.toString());

    currentPage = prvaStranica
        .where((i) => i.contains(siteUrlNoNumber))
        .map((e) => int.parse(e.replaceAll(RegExp(r'[^0-9]'), '')))
        .toList()
        .reduce(max);
  }
  return maxPage;
}
```

Slika 45. Dohvat broja stranica za web stranicu Bika.net

Ostale promjene u odnosu na web stranicu Moj-posao.net su u dohvatu svakog pojedinog podatka oglasa. Razlika je nastala zbog činjenice da web stranica Bika.net ne posjeduje imenovane regije u HTML-u po kojima bi se mogli dohvatiti potrebni podaci uz pomoć Puppeteer-a. Rješenje problema je da se za isto koristi LINQ i RegExp-a za pronalazak podataka unutar HTML-a cijele stranice. Korištenje LINQ i RegExp-a vidljivo je na slici 46.

```
String GetAdAplicationDate(String tijelo) {    Name non-constant iden
  final regexp = RegExp(r'<span>Rok prijave:</span>(.|\n)*?<\/li>');
  final match = regexp.firstMatch(tijelo);
  String text = match?.group(0) ?? "";
  return text
      .replaceFirst('<span>Rok prijave:</span>', '')
      .replaceFirst('</li>', '')
      .trim();
}
```

Slika 46. Dohvat datuma isteka oglasa na web stranici Bika.net

Kao primjer korištenja Linq i RegExp-a za dohvat podataka prikazat će se na metodi GetAdAplicationDate(). Pomoću RegExp-a dohvaća se tekst koji se nalazi između Mjesto: i <\/li>, a ujedno s time dobijemo i višak teksta Mjesto:, <\/li>. Zbog navedenog potrebno je nakon toga izbrisati taj

višak. U ostalim metodama koristimo isti način dohvata podataka samo se oni razlikuju ovisno o tome gdje se u tekstu nalaze.

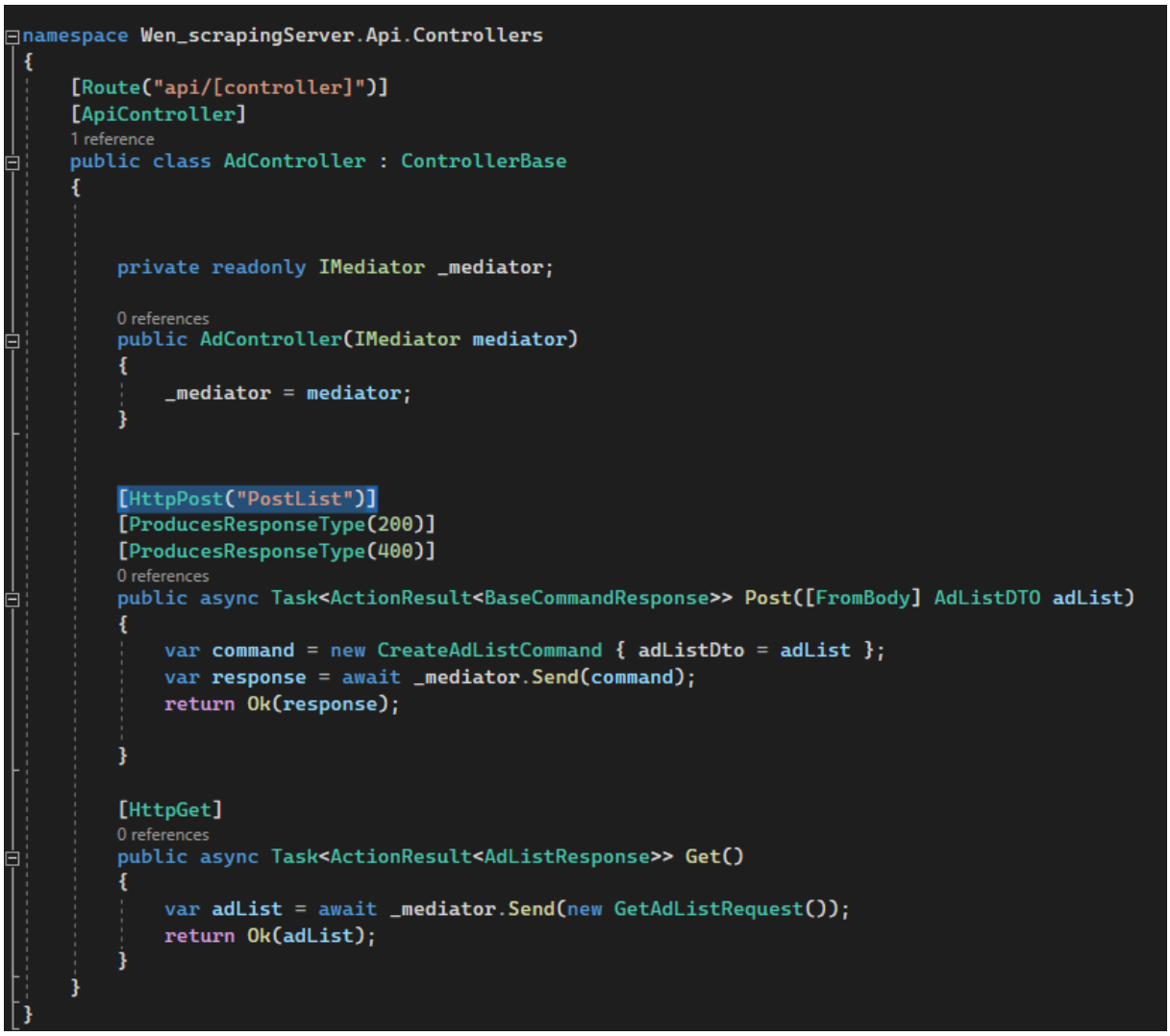
4.3. Backend server i baza podataka

Backend server je u ovom projektu zadužen samo za komunikaciju sa bazom podataka. Backend rješenje sastoji se od tri projekta a to su: api, aplication i persistance.

4.3.1. Api

U Api projektu nalaze se svi kontroleri koji šalju odgovore na pristigle zahtjeve. U api kontrolerima specifični su nazivi api-ja na koje se šalju zahtjevi klijenata. U ovom projektu kontroleri su podijeljeni u klase po imenu modela. U api projektu koristi se nuget paket Swashbuckle.AspNetCore koji omogućuje korištenje Swagger alata za testiranje api controlera. Jedna od klasa je adController (prikazana na slici 47) u kojoj se nalaze sljedeći kontroleri:

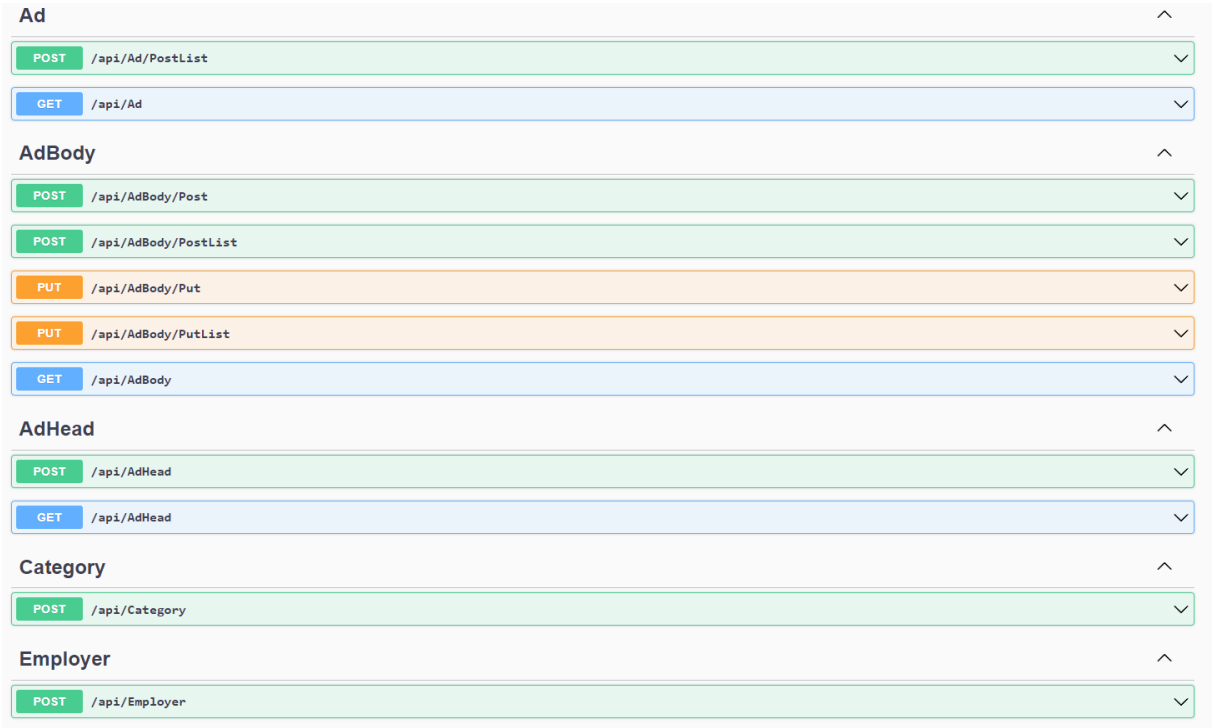
- HttpGet – koji na zahtjev vraća sve oglase iz baze podataka,
- HttpPost("PostList") imenovana HTTP post metoda koja u bazu podataka dodaje sve dobivene oglase iz liste.



```
namespace Wen_scrapingServer.Api.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 reference
    public class AdController : ControllerBase
    {


        private readonly IMediator _mediator;

        0 references
        public AdController(IMediator mediator)
        {
            _mediator = mediator;
        }


        [HttpPost("PostList")]
        [ProducesResponseType(200)]
        [ProducesResponseType(400)]
        0 references
        public async Task<ActionResult<BaseCommandResponse>> Post([FromBody] AdListDTO adList)
        {
            var command = new CreateAdListCommand { adListDto = adList };
            var response = await _mediator.Send(command);
            return Ok(response);

        }

        [HttpGet]
        0 references
        public async Task<ActionResult<AdListResponse>> Get()
        {
            var adList = await _mediator.Send(new GetAdListRequest());
            return Ok(adList);
        }
    }
}
```

Slika 47. Ad Kontroler

Popis svih api-ja prikazan je na slici 48 uz pomoć Swagger nugeta.



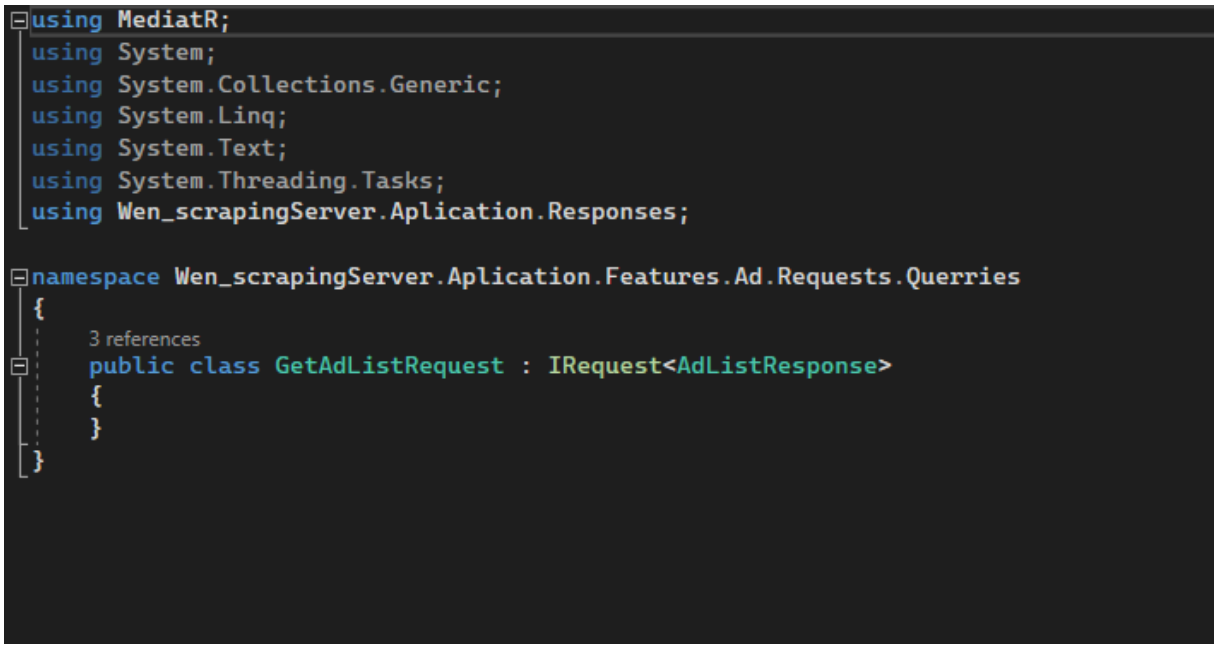
Slika 48. Prikaz svih api-a unutar swagger stranice

4.3.2. Aplication

Arhitektura Aplication projekta podijeljena je u pet zasebnih mapa a to su:

- Contracts – sadrži datoteke svih sučelja projekta,
- DTO – sadrži datoteke objekta prijenosa podataka koji su posloženi po mapama, a koje mape su imenovane prema nazivima modela,
- Features – sadrži datoteke za sve naredbe i njihove rukovatelje koji su također posloženi po mapa, a one su imenovane po nazivu modela,
- Profiles – sadrži datoteke mappera između modela i njegovog DTO-a,
- Responses – sadrži datoteke odgovora koji se šalju kao odgovor na upućeni zahtjev.

Zahtjev za sve oglase dolazi u klasu AdControler nad metodom koja ima atribut HttpGet. Izgled tog zahtjeva definiran je u klasi GetAdListRequest() u kojem je propisano da se očekuje AdListResponse kao odgovor prikazano na slici 49.



Slika 49. Izgled zahtjeva GetAdListRequest

Klasa AdListResponse sadrži objekte DTO-a koji se šalju klijentu (slika 50).

Slika 50. Izgled odgovora AdListResponse

Klasa koja je zadužena za popunjavanje AdListResponse-a sa podacima dobivenim iz baze podataka je GetAdListRequestHandler.

Metoda Handle unutar klase GetAdListRequestHandler poziva dohvate iz baze podataka i puni listu AdDTO s tim podacima. Ta lista se prosljeđuje se u konstruktor AdListResponse-a koji se šalje klijentu.

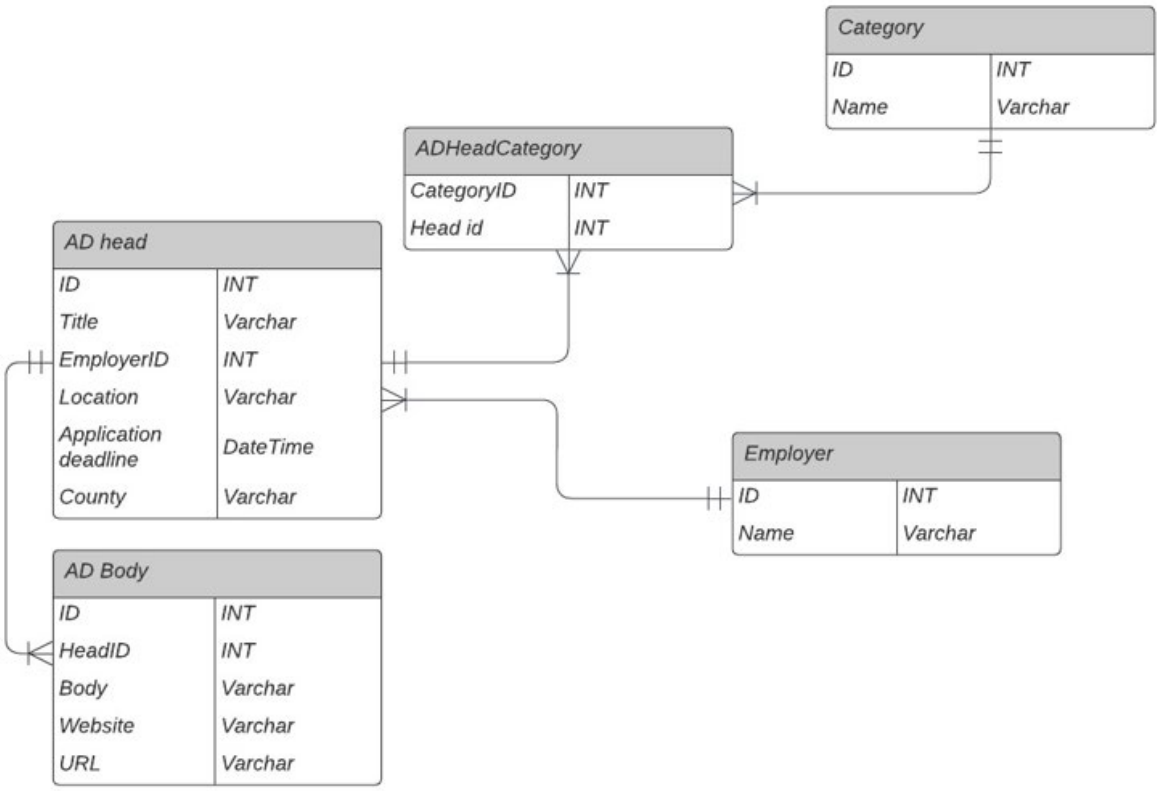
4.3.3. Domain i Persistance

Projekt Domain sadrži modele podataka koji su isti modelima koji se nalaze u bazi podataka a to su:

- AdBody,
- AdHead,

- AdHeadCategory,
- Category,
- Employer,
- Ad.

Persistance projekt zadužen je za rad sa bazom podataka. U istoj se nalaze potrebne postavke za Entity framework te naredbe potrebne za: dodavanje redova u bazu podataka, brisanje redova iz baze podataka, provjeru postojanja reda sa određenim id-om, dohvaćanje reda sa određenim id-om, dohvaćanje svih redova iz tablice i ažuriranje reda u bazi podataka. Za svaku tablicu u bazi podataka gore navedene naredbe nalaze se u datotekama koje se nalaze u mapi Repositories. Struktura podataka u bazi podataka prikazana je relacijskim dijagramom (slika 51).



Slika 51. Relacijski dijagram baze podataka

5. Zaključak

Ovim radom predstavljena je aplikacija za brže i jednostavnije pronalaženje zaposlenja. Ovom aplikacijom olakšano je pretraživanje i pronalaženje oglasa budući da ista prikuplja i sortira oglase s tri različite web stranice koje nude zaposlenja. Tako ona obuhvaća oglase koji su oglašeni na Moj-posao.net, Posao.hr i Bika.net. Na taj način korisnicima nudi brže i jednostavnije pronalaženje zaposlenja budući da isti ne moraju više puta pregledavati. Već na ovoj web stranici mogu pronaći oglase koji su oglašeni na sve tri web stranice. Prednost Flutter jezika kojom je ova aplikacija napisana je da nudi potpunu slobodu nad izgledom sučelja i podržava rad na sljedećim sustavima: Windovs, Linux, Android, iOS, Web. Rad aplikacije ne zahtijeva novo pisanje koda za te sustave nego iziskuje male dorade kako bi isti kod radio na svim aplikacijama. Izradom vlastitog backend-a omogućena je sloboda upravljanja i proširenja istog. Baza podataka je izrađena pomoću Entity framework-a na osnovi pristupa model first što je omogućilo jednostavno kreiranje baze i jednostavni rada s istoimenom bazom.

Trenutni nedostatak ove aplikacije je jednojezičnost. Naime, ista je samo na hrvatskom jeziku te ne nudi mogućnost odabira drugih jezika. Isto tako njome su sada obuhvaćene samo unaprijed navedene tri web stranice.

Ovo je prva verzija aplikacije i zasigurno će se još optimizirati postojeći kod kao i poboljšati samo sučelje aplikacije. Potencijalna proširenja ove aplikacije su stvaranje korisničkog računa, višejezičnost aplikacije, praćenje odabranih oglasa, dodavanje informacija o poslodavcu (njegovi podaci, adresa i slično) te dohvaćanje oglasa s većeg broja web stranica.

Literatura

[1] A tour of C# - Overview. (2022). Dohvaćeno 1 rujna 2022 iz https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/

[2] The Good and the Bad of C# Programming. (2021). Dohvaćeno 1 rujna 2022 iz https://www.altexsoft.com/blog/c-sharp-pros-and-cons/

[3] FAQ. (2022). Dohvaćeno 1 rujna 2022 iz https://docs.flutter.dev/resources/faq

[4] A. Tashmatova, A Topalidis. (2022). React Native vs Flutter. Dohvaćeno 1 rujna 2022 iz:https://maddevs.io/blog/react-native-vs-flutter-which-to-choose/

[5] What is Dart Programming. (2019). Dohvaćeno 1 rujna 2022 iz: https://www.javatpoint.com/flutter-dart-programming

[6] Dart overview. (2019) . Dohvaćeno 1 rujna 2022 iz: https://dart.dev/overview

[7] D. Schiemann. (2018). Google Releases Puppeteer 1.0. Dohvaćeno 1 rujna 2022 iz: https://www.infoq.com/news/2018/01/puppeteer-1-released/

[8] Puppeteer in Dart. (2022). Dohvaćeno 1 rujna 2022 iz: https://pub.dev/packages/puppeteer

[9] What is Entity Framework?. (2013). Dohvaćeno 1 rujna 2022 iz: https://www.entityframeworktutorial.net/what-is-entityframework.aspx

[10] Introduction to widgets. (2016). Dohvaćeno 1 rujna 2022 iz: https://docs.flutter.dev/development/ui/widgets-intro

[11] Z. Rehman. (2019). Dependency Injection In Flutter. Dohvaćeno 5 rujna 2022 iz: https://medium.com/flutter-community/dependency-injection-in-flutter-f19fb66a0740

[12] M. Fowler. (2017). Data Transfer Object. Dohvaćeno 5 rujna 2022 iz: https://martinfowler.com/eaaCatalog/dataTransferObject.html

[13] M. Fowler. (2004). LocalDTO. Dohvaćeno 5 rujna 2022 iz: https://martinfowler.com/bliki/LocalDTO.html

[14] Data Transfer Object DTO Definition and Usage. (2022). Dohvaćeno 5 rujna 2022 iz: https://www.okta.com/identity-101/dto/

[15] Retrofit generator. (2019). Dohvaćeno 5 rujna 2022 iz: https://pub.dev/packages/retrofit_generator

[16] Change height of GridView row to fixed height. (2022). Dohvaćeno 5 rujna 2022 iz: https://github.com/flutter/flutter/issues/55290

Popis slika

Slika 1. Prikaz web sučelja posao.hr 3
Slika 2. Prikaz web sučelja posao.hr 4
Slika 3. Dodavanje Puppeteer paketa u projekt 8
Slika 4. Korištenje Puppeteer paketa 8
Slika 5. Pregled arhitekture web klijenta 10
Slika 6. Klasa MyApp – izrada početkog widget-a 11
Slika 7. Klasa RouteGenerator – inicijalizacija rute 12
Slika 8.Klasa Routes – Imenovanje putanja 12
Slika 9. Metoda initAppModul - Izrada početnog Dependency Injection 13
Slika 10. Izgled stranice učitavanja 14
Slika 11. Prikaz inicijalizacije stranice učitavanja 14
Slika 12. Prikaz ViewModel-a stranice učitavanja 15
Slika 13. Metoda getData – dohvat oglasa 15
Slika 14. Klasa GetAdListUseCase 16
Slika 15. Metoda GetAdList 17
Slika 16. AdListResponseMapper 17
Slika 17. AdResponseMapper 18
Slika 18. AdHeadResponeMapper 18
Slika 19. Klasa RemoteDataSoureImplementer 19
Slika 20. Klasa AppServiceClient 20
Slika 21. Metoda getAdList 21
Slika 22. Klasa DioFactory 23
Slika 23. Klasa Response 24
Slika 24. Klasa Request 25
Slika 25. Izgled glavne stranice 26
Slika 26.Metoda sortAds 26
Slika 27. Metoda filterAds 27
Slika 28. Prikaz oglasa na filtriranja 27
Slika 29. Izgled web stranice oglasa 28
Slika 30. Prikaz stranice učitavanja 29
Slika 31. Dohvat županija 30
Slika 32. Dohvat županije 31
Slika 33. Dohvat poveznica oglasa sa jedne web stranice 31
Slika 34. Dohvat svih poveznica unutar jedne regije HTML-a 32
Slika 35. Dohvat broja stranica oglasa u jednoj županiji 32
Slika 36. Prolazak kroz sve poveznice novih oglasa 33
Slika 37. Izmjena datuma objave oglasa sa web stranice Posao.hr 34
Slika 38. Mapa županija web stranice Posao.Hr 35
Slika 39. Mapa županija web stranice Moj-posao.net 36
Slika 40. Odabir regije #main na web stranici Moj-posao.net 36
Slika 41. Odabir regije box na web stranici Posao.hr 37
Slika 42. Odabir poveznica koje sadrže riječ posao 37
Slika 43. Odabir poveznica koje sadrže riječ oglasi 37
Slika 44. Mapa županija web stranice Bika.net 39

Slika 45. Dohvat broja stranica za web stranicu Bika.net 40
Slika 46. Dohvat datuma isteka oglasa na web stranici Bika.net 40
Slika 47. Ad Kontroler 42
Slika 48. Prikaz svih api-a unutar swagger stranice 43
Slika 49. Izgled zahtjeva GetAdListRequest 44
Slika 50. Izgled odgovora AdListResponse 44
Slika 51. Relacijski dijagram baze podataka 45

SAŽETAK

Ovaj rad opisuje izradu aplikacije za integraciju ponuda poslova iz više izvora. Opisuje problematiku rastućeg broja web stranica za oglašavanje poslova kao i korištenje tehnologije i njezinu primjenu u ovom radu. Ovaj projekt se sastoji od dvije klijentske strane - jedna se bavi dohvatom podataka oglasa sa različitih web stranica koje se pohranjuju u bazu podataka dok druga služi za prikaz tih oglasa iz baze podataka i njihovo filtriranje kako bi se pronašli oglasi koji zanimaju tražitelja zaposlenja. Projekt se također sastoji od backend servera koji poslužuje klijente i komunicira s SQL bazom podataka. Razvoj API servisa i razvoj klijenta popraćen je slikama koda koji su zatim detaljno opisani.

Ključne riječi: C#, Flutter, Dart, Puppeteer, REST API, MVVM, web aplikacija, windows aplikacija, sql, entity framework.

ABSTRACT

This paper describes the creation of an application for integrating job offers from multiple sources. It describes the problem of the growing number of websites for advertising jobs as well as the use of technology and its application in this paper. This project consists of two clients sides – one that deals with retrieving ad data from different web pages that are then stored in the database, while the other serves to display those ads from the database and filter them to find ads that interest the job seeker the most. It also consists of a backend server that serves clients and communicates with the sql database. API service development and client development are accompanied by code images that are then described in detail.

Keywords: C#, Flutter, Dart, Puppeteer, REST API, MVVM, web application, windows application, sql, entity framework.