Mobilna aplikacija s internetskim sučeljem za izračun cijene uvoza automobila

Košturjak, Mateo

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

MOBILNA APLIKACIJA S INTERNETSKIM SUČELJEM ZA IZRAČUN CIJENE UVOZA AUTOMOBILA

Završni rad

Mateo Košturjak

Osijek, 2024

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 18.02.2024.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Mateo Košturjak
Studij, smjer:	Računarstvo
Mat. br. Pristupnika, godina upisa:	R 4368, 22.07.2019.
OIB Pristupnika:	53133499406
Mentor:	prof. dr. sc. Irena Galić
Sumentor:	Marin Benčević, mag. ing. comp.
Sumentor iz tvrtke:	
Naslov završnog rada:	Mobilna aplikacija s internetskim sučeljem za izračun cijene uvoza automobila
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Razviti i postaviti na poslužitelj aplikacijsko programsko sučelje bazirano na REST protokolu za izračun cijene uvoza automobila. Razviti mobilnu aplikaciju koja sadrži grafičko sučelje za korištenje internetskog sučelja. Opisati proces i korištene alate za izradu aplikacijskog programskog sučelja, i mobilnih aplikacija. Opisati REST protokol. Prikazati nedostatke razvijenog rješenja i moguća poboljšanja. Sumentor: Marin
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	18.02.2024.
Datum potvrde ocjene od strane Odbora:	21.02.2024.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.02.2024.

Ime i prezime studenta:	Mateo Košturjak
Studij:	Računarstvo
Mat. br. studenta, godina upisa:	R 4368, 22.07.2019.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Mobilna aplikacija s internetskim sučeljem za izračun cijene uvoza automobila**

izrađen pod vodstvom mentora prof. dr. sc. Irena Galić

i sumentora Marin Benčević, mag. ing. comp.

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Zadatak završnog rada je razvijanje i postavljanje na poslužitelj aplikacijskog programskog sučelja bazirano na REST protokolu za izračun cijene uvoza automobila, koje pojednostavnjuje isti izračun. Također, razviti mobilnu aplikaciju koja sadrži grafičko sučelje za korištenje sučelja koje je postavljeno na poslužitelj. Unutar aplikacije korisniku se omogućuje unos 6 parametara koji su potrebni za izračun.

U drugom poglavlju diskutira se generalno o alatima koji su upotrijebljeni za stvaranje aplikacije i drugih usluga, s njihovom konkretnom primjenom. S druge strane, treće i četvrto poglavlje fokusira se na strukturu aplikacijskog programskog sučelja i aplikacije, te ulogu koju su pojedini dijelovi odigrali u njezinoj izgradnji. Svaki segment strukture će biti detaljnije opisan, praćen konkretnim primjerima primjene unutar same aplikacije. Ovo poglavlje ima više dijelova, svaki posvećen opisu različitih komponenti koje su korištene tijekom razvoja aplikacije. Konačno, u petom poglavlju, iznosi se kratki zaključak o postignutim ciljevima u okviru zadatka završnog rada te o samoj osnovnoj ideji iza razvoja aplikacije. Glavni cilj aplikacije je pružiti korisnicima pomoć pri izračunu cijene uvoza automobila.

1.1. Zadatak završnog rada

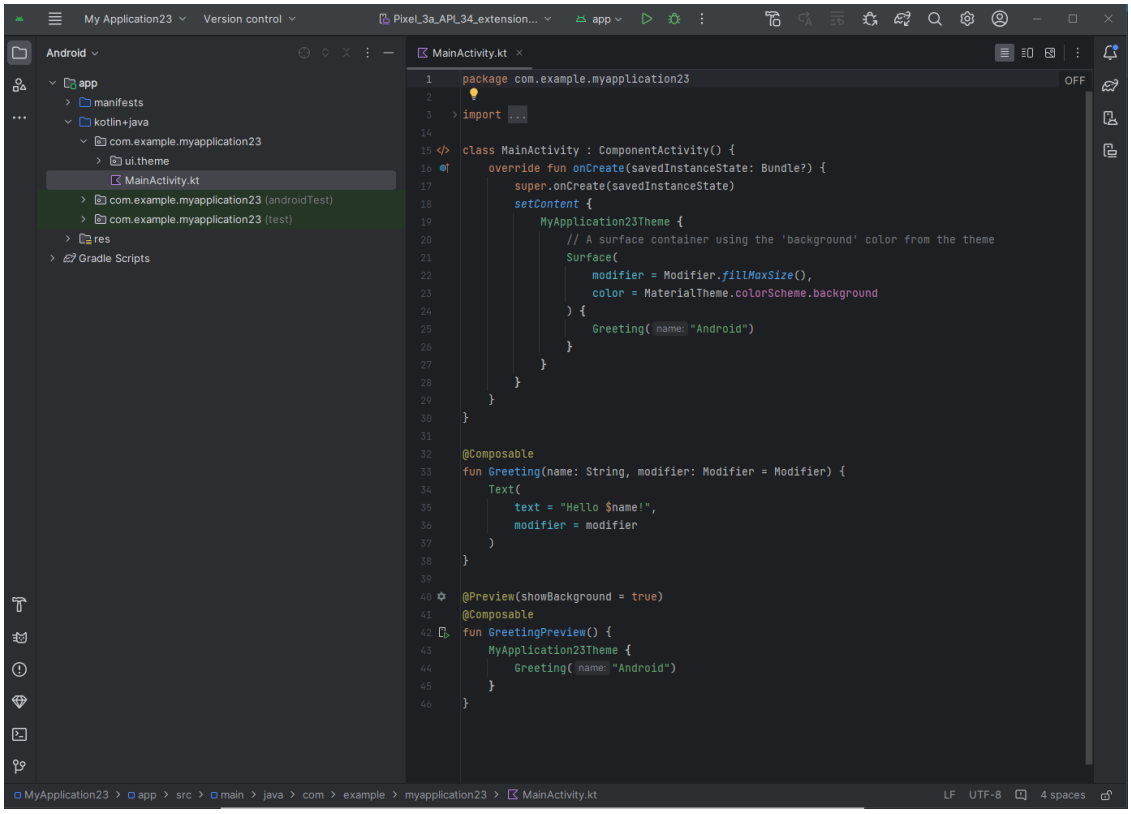
U završnom radu potrebno je razviti i postaviti na poslužitelj aplikacijsko programsko sučelje bazirano na REST protokolu za izračun cijene uvoza automobila. Razviti mobilnu aplikaciju koja sadrži grafičko sučelje za korištenje internetskog sučelja. Opisati proces i korištene alate za izradu aplikacijskog programskog sučelja, i mobilnih aplikacija. Opisati REST protokol. Prikazati nedostatke razvijenog rješenja i moguća poboljšanja.

2. ALATI

Ovo poglavlje govori o alatima koji su korišteni za izradu aplikacije i aplikacijsko programskog sučelja. U nastavku poglavlja opisat će se korišteni alati i njihova primjena. Ukratko će se opisati o svakom alatu njegova funkcija, kako radi i njegov nastanak.

2.1 Android Studio

Android Studio [1], službeno integrirano razvojno okruženje za Googleov operativni sustav Android, specifično je dizajnirano za razvoj Android aplikacija i temelji se na IntelliJ Idea softveru [2]. Dostupno je za operativne sustave poput Windowsa, macOS-a i Linuxa, kao primarni IDE za izvorni razvoj Android aplikacija. Programski jezik Kotlin je postao Googleov najpopularniji jezik za razvoj Android aplikacija, zamjenjujući Javu. Android Studio pruža jednostavno i jedinstveno okruženje za sve Android uređaje. Pri razvoju aplikacija za Android platformu, praksa je razdvajanja koda od resursa, uključujući izgled, slike i druge podatke. Ovo razdvajanje olakšava prilagodbu različitim uređajima i tržištima bez potrebe za dodatnim kodom ili zahtjevima od programera. Korisničko sučelje je prikazano na slici 2.1.



Slika 2.1: Prikaz korisničkog sučelja Android Studia

2.2 Programski jezik Kotlin

Kotlin [3] predstavlja višeplatformski, suvremen, sažet i siguran programski jezik otvorenog koda. Ovaj jezik podržava funkcijsko i objektno orijentirano programiranje te se ističe izvrsnim mogućnostima testiranja. Godine 2017. Google je prepoznao Kotlin kao vodeći jezik za razvoj mobilnih aplikacija na Android platformi. JetBrains je bio kreator ovog jezika, a njegov nastanak datira iz 2011. godine. Osim toga, Kotlin je interoperabilan s programskim jezikom Java, dijeleći sličnu sintaksu s drugim jezicima poput Jave i C#.

2.2.1. Sintaksa

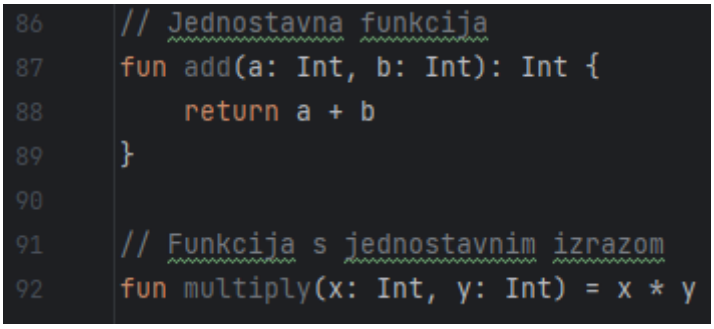
Deklaracije varijabli i popisi parametara u Kotlinu imaju tip podataka nakon imena varijable (i s dvotočkom razdjelnikom), slično kao u Adaži, BASICu, Pascalu, TypeScriptu i Rustu. Zarezi su opcionalni kao terminator izjave, u većini slučajeva dovoljna je nova linija kako bi se prevoditelju omogućilo da zaključi da je izjava završena. Varijable u Kotlinu mogu biti samo za čitanje, deklarirane s ključnom riječju val, ili promjenjive, deklarirane s ključnom riječju var. Članovi klase su javni prema podrazumijevanim postavkama, a same klase su određene prema podrazumijevanim postavkama, što znači da je stvaranje izvedene klase onemogućeno, osim ako se osnovna klasa ne deklarira s ključnom riječju open. Primjer sintakse se vidi na slici 2.2.

```
86  // Deklaracija varijable
87  val greeting: String = "Hello, Kotlin!"
88
89  // Tipovi podataka se mogu implicitno odrediti
90  val number = 42
91
92  // Promjenjive varijable
93  var counter = 0
```

Slika 2.2: Primjer sintakse varijabli u Kotlinu

Osim klasa i članskih funkcija objektno orijentiranog programiranja, Kotlin također podržava proceduralno programiranje s pomoću funkcija. Funkcije i konstruktori u Kotlinu podržavaju zadane argumente, popise argumenata promjenjive duljine, imenovane argumente i preopterećivanje jedinstvenim potpisom. Funkcije članova klase su virtualne, tj. izvršavaju se na temelju tipa objekta na koji su pozvane. Kotlin kod može se prevoditi u JavaScript, omogućavajući interoperabilnost između koda napisanog na ta dva jezika. To se može koristiti ili za pisanje

potpunih web aplikacija u Kotlinu ili za dijeljenje koda između Kotlin *backenda* i JavaScript *frontenda*. Primjer jednostavne funkcije se vidi na slici 2.3.



Slika 2.3: Primjer sintakse funkcija u Kotlinu

2.2.2. Usporedba s Javom

Kotlin je jezik koji sjedinjuje funkcijsko i objektno orijentirano programiranje, pružajući programerima fleksibilnost za korištenje oba pristupa. Ovaj jezik omogućava elegantno kombiniranje jasne sintakse, sigurnosti tipova te visoke izražajnosti. Svojom kombinacijom, Kotlin se ističe kao snažan jezik koji se lako uči i primjenjuje. U usporedbi s Javom, Kotlin ne zahtijeva deklaraciju iznimki, eliminira nepotrebne null reference podrazumijevajući da varijable nisu nullable te pruža jednostavniju i čišću sintaksu. Dodatno, podržava proširenja, olakšavajući dodavanje funkcionalnosti klasama bez nasljeđivanja. Koncept podatkovnih klasa automatski generira korisne metode, pojednostavljujući rad s podacima. Kotlin također ostvaruje potpunu interoperabilnost s Javom, omogućavajući integraciju postojećeg Java koda unutar Kotlin projekta i obratno.

Kotlin omekšava ograničenje Jave koje dopušta postojanje statičkih metoda i varijabli samo unutar tijela klase. Statički objekti i funkcije mogu se definirati na vrhu paketa bez potrebe za suvišnim razinama klase. Radi kompatibilnosti s Javom, Kotlin pruža JvmName napomenu koja specificira naziv klase koji se koristi kada se paket promatra iz Javina projekta. [4] Na slici 2.4 je prikazan jednostavan Kotlin kod.

```kotlin
class Person(val name: String, val age: Int) {
    fun greet() {
        println("Hello, my name is $name and I am $age years old.")
    }
}

fun main() {
    val person = Person( name: "John",  age: 25)
    person.greet()
}
```

Slika 2.4: Primjer jednostavnog Kotlin koda

2.3 XML

XML [5], kratica za Extensible Markup Language, predstavlja jezik za označavanje podataka. Zbog svoje jednostavnosti i lakoće čitanja, postao je iznimno popularan. Podaci se pohranjuju u formatu običnog teksta, što omogućuje neovisnost o softveru i hardveru prilikom pohrane i prijenosa podataka. Korisničko sučelje cijele aplikacije, uključujući svaki njegov element, detaljno je opisano putem XML-a, a primjer dijela XML dokumenta prikazan je na slici 2.5.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:background="@color/teal_700">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="172dp"
        app:srcCompat="@drawable/dizajn_bez_naslova" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_marginTop="16dp"

        android:layout_height="40dp"
        android:background="@color/teal_700"
        android:orientation="horizontal">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="center_vertical"
            android:layout_marginEnd="8dp"
            android:layout_weight="1"
            android:text="Rabljeni Automobil"
            android:textColor="#333"
            android:textSize="16sp"/>
```

Slika 2.5: Primjer XML dokumenta.

2.4 Flask

Flask [6] se klasificira kao mikrookvir napisan u Phytonu, jer ne zahtijeva određene alate ili knjižnice. Nema sloj apstrakcije baze podataka, provjeru valjanosti obrazaca ni druge komponente gdje postojeće knjižnice trećih strana pružaju zajedničke funkcije. Ipak, Flask podržava proširenja koja mogu dodati značajke aplikacije kao da su implementirane izravno u samom Flasku. Postoje proširenja za objektno relacijsko preslikavanje, provjeru valjanosti obrazaca, upravljanje prijenosom datoteka, različite tehnologije otvorene autentifikacije i nekoliko uobičajenih alata povezanih s okvirom. Primjeri aplikacija koje koriste Flask okvir uključuju Pinterest i LinkedIn.

2.4.1. Povijest Flaska

Flask je stvorio Armin Ronacher iz Pocooa, međunarodne skupine zaljubljenika u Python osnovane 2004. godine. Prema Ronacheru, ideja je izvorno bila šala za Prvi april koja je postala dovoljno popularna da se pretvori u ozbiljnu aplikaciju. Kada su Ronacher i Georg Brandl 2004. godine stvorili sustav za oglasnu ploču pisan u Pythonu, razvijeni su Pocoo projekti Werkzeug i Jinja.

2.4.2. Python

Flask je napisan u Pythonu [7], koji je programski jezik opće namjene visoke razine. Njegova filozofija dizajna ističe čitljivost koda, nosi potpis svog tvorca, Guida van Rossuma, od kada je prvi put objavljen 1991. godine. Python je dinamički tipiziran i koristi sustav automatskog prikupljanja smeća. Podržava više programskih paradigmi, uključujući strukturirano (posebno proceduralno), objektno orijentirano i funkcionalno programiranje. Ovaj jezik naglašava čitljivost, jednostavnost i fleksibilnost, te sve više privlači programere zbog svoje lakoće učenja i praktičnosti.

Kao interpretirani jezik, Python obrađuje izvorni kod tijekom izvođenja, čime se izbjegava potreba za prethodnim kompajliranjem. Python se često koristi u znanstvenim istraživanjima, analizi podataka i području strojnog učenja, pronalazeći primjenu u različitim industrijama poput zdravstva, financija i tehnologije. Njegova široka dostupnost privukla je veliku zajednicu korisnika, rezultirajući obimnom standardnom bibliotekom i mnoštvom modula dostupnih za upotrebu. Python ostaje izuzetno svestran jezik, omogućavajući programerima raznolike mogućnosti i olakšavajući rad u raznim sektorima.

2.4.3. Postavljanje Flaska

Flask pruža jednostavne korake za stvaranje web aplikacija. Početna instalacija obavlja se s pomoću pip naredbe, a nakon toga, uobičajena struktura Flask aplikacije obuhvaća datoteke za aplikaciju, statičke datoteke i predloške. Prva faza uključuje postavljanje HTTP servera. Flask može koristiti vlastiti razvojni server ili se integrirati s drugim WSGI [8] kompatibilnim serverima kako bi komunicirao s web preglednicima i primao HTTP zahtjeve.

Flask koristi dekoratore za rutiranje. Rutiranje je ključna funkcionalnost Flaska, gdje dekoratori, poput @app.route('/'), povezuju URL staze s odgovarajućim funkcijama pogleda. Ove funkcije obrađuju specifične URL staze kada korisnici pristupe određenim resursima. Funkcije pogleda predstavljaju Python funkcije koje se izvršavaju kada korisnik pristupi određenom URL-u. Te funkcije generiraju odgovor koji će biti poslan korisniku. Flask koristi request objekt za čitanje podataka koje korisnik šalje, dok response objekt predstavlja odgovor koji će biti poslan korisniku. Ova interakcija omogućava dinamičko generiranje i prikazivanje sadržaja na web stranicama. Flask se često koristi za izgradnju manjih i srednjih web aplikacija, RESTful aplikacijsko programskih sučelja i prototipova, nudeći mogućnosti proširenja za naprednije funkcionalnosti poput rad s bazama podataka i autentikacije. Slika 2.6 prikazuje osnovnu Flask aplikaciju.

```
1  from flask import Flask, render_template
2
3  app = Flask(__name__)
4
5  @app.route('/')
6  def home():
7      return render_template('index.html', message='Hello, Flask!')
8
9  if __name__ == '__main__':
10     app.run(debug=True)
```

Slika 2.6: Prikaz osnovne Flask aplikacije

2.4.4. Rad Flaska

Jinja2 [9] predlošci su sredstvo za generiranje dinamičkih HTML stranica, što omogućuje ubacivanje promjenjivih podataka u HTML kod. Ovo čini proces stvaranja dinamičkih web stranica jednostavnim i fleksibilnim. Middleware u Flasku omogućava proširenje funkcionalnosti aplikacije. Oni mogu modificirati zahtjeve i odgovore prije nego što dođu do funkcija pogleda ili

prije nego što budu poslani korisniku. Proširenja u Flasku pružaju dodatne funkcionalnosti, omogućujući integraciju s različitim alatima i servisima, poput baza podataka, autentikacije [10] i drugih naprednih značajki. U razvojnom okruženju, Flask pruža način rada s debugging informacijama, olakšavajući proces otkrivanja i ispravljanja pogrešaka. Kada je aplikacija spremna za produkcijsko okruženje, Flask se može integrirati s raznim web serverima ili učahuriti u WSGI kompatibilne servere.

2.4.5. Primjena Flaska

Flask nudi pristupačan i prilagodljiv pristup razvoju Python web aplikacija i aplikacijsko programskih sučelja. Flask se ističe po jednostavnom dizajnu koji programerima pruža slobodu odabira elemenata i prilagodbe njihovih aplikacija prema vlastitim potrebama. Njegove prednosti uključuju ugrađeni razvojni server i brzi debugger, laganu strukturu, podršku za sigurne kolačiće, korištenje Jinja2 za predloške, raspodjelu zahtjeva putem REST-a te ugrađenu podršku za unit testiranje.

Uobičajene primjene Flaska obuhvaćaju raznovrsne web aplikacije, uključujući blogove, e-trgovine, društvene mreže te mnoge druge. Njegova jednostavnost i fleksibilnost čine ga prikladnim i za male i za veće, kompleksnije projekte. Flask je također popularan izbor za razvoj RESTful aplikacijsko programskih sučelja, omogućavajući komunikaciju između različitih softverskih sustava te dijeljenje podataka i integraciju usluga.

Osim toga, Flask se ističe i u području prototipiranja, omogućavajući brzu izradu i testiranje web-ideja i koncepata bez značajnog opterećenja. Također se koristi za izgradnju mikroservisa, interaktivne nadzorne ploče za stvaranje vizualizacija u stvarnom vremenu te obrazovne projekte zbog svoje jasne strukture i jednostavnosti učenja. Flask predstavlja lagani, ali snažan alat za razvoj, pružajući raznolike mogućnosti u svijetu web programiranja.

2.5 REST protokol

REST (Representational State Transfer) [11] je stil programske arhitekture koji je stvoren kako bi vodio dizajn i razvoj arhitekture World Wide Weba. REST definira skup ograničenja o tome kako bi arhitektura distribuiranog, Internet mjerila hipermedijskog sustava, poput Weba, trebala ponašati. REST arhitekturalni stil naglašava jednolična sučelja, neovisno implementiranje komponenata, skalabilnost interakcija između njih te stvaranje slojevite arhitekture radi poticanja predmemoriranja za smanjenje percepcije latencije korisnika i enkapsulaciju starih sustava.

2.5.1. Povijest

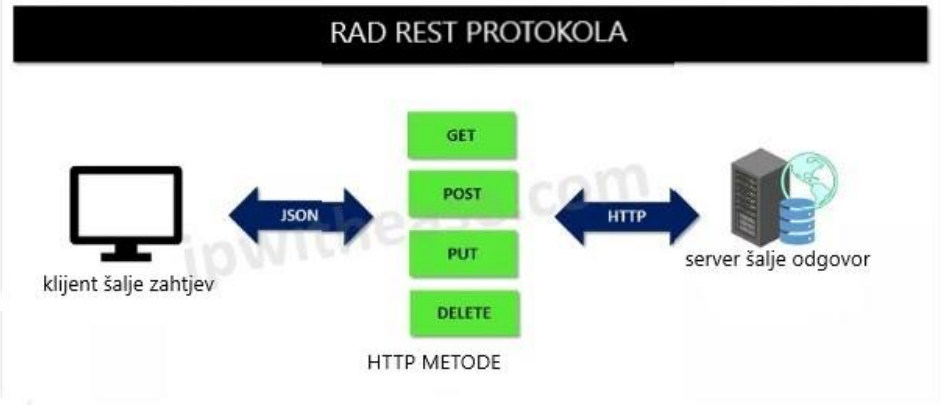
U ranim 1990-ima, s rastućom popularnošću Svjetske mreže, potreba za standardiziranim protokolima potaknula je Roya Fieldinga i radnu skupinu W3C i IETF na stvaranju formalnih opisa temeljnih standarda Weba: URI, HTTP i HTML. Fielding je, tijekom sljedećih šest godina, oblikovao REST arhitekturalni stil kako bi zadovoljio zahtjeve globalnih mrežno baziranih aplikacija, naglašavajući nisku prepreku za globalno usvajanje.

Unatoč tome što Web ne slijedi svaki REST ograničenje, Fieldingov rad pruža okvir za identifikaciju neslaganja prije standardizacije, a primjerice, istaknuo je zabrinutosti oko prakse ugradnje informacija o sesiji u URI i korištenja HTTP kolačića, upozoravajući na moguće probleme s keširanjem, skalabilnošću i pitanjima privatnosti. REST ostaje ključan vodič za ocjenu i poboljšanje arhitekture mrežno baziranih aplikacija.

2.5.2. Princip rada

Izraz "representational state transfer" uveo je i definirao 2000. godine informatičar Roy Fielding u svojoj doktorskoj disertaciji. Ova terminologija označava da će poslužitelj odgovoriti s prikazom resursa (danas često HTML, XML ili JSON dokument) koji sadržava hipermedijalne poveznice koje se mogu slijediti kako bi se promijenilo stanje sustava. Svaki takav zahtjev zauzvrat će primiti prikaz resursa, i tako dalje.

Važna posljedica je da je jedini identifikator koji treba biti poznat identifikator prvog zatraženog resursa, a svi ostali identifikatori bit će otkriveni. To znači da se ti identifikatori mogu mijenjati bez potrebe da se klijenta unaprijed obavijesti i da može postojati samo labava veza između klijenta i poslužitelja. Prikaz rada REST protokola prikazan je na slici 2.7.

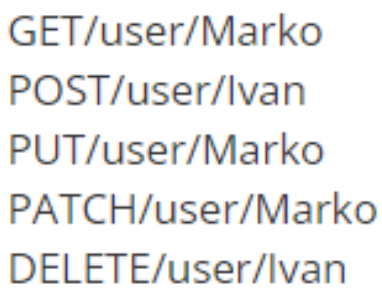


Slika 2.7: Prikaz rada REST protokola

2.5.3. HTTP metode

REST resursi moraju podržavati standardne HTTP zahtjeve bez stanja. Osnovna ideja je održavanje jednostavnosti i skalabilnosti komunikacije između računala na mreži. Za razliku od SOAP-a [12], gdje su se često koristila proizvoljna imena podržanih operacija u REST-u koriste standardne HTTP metode kao što su GET, POST, PATCH, PUT i DELETE.

Prva metoda, GET, koristi se za dohvaćanje podataka s određenog resursa na poslužitelju. Slijedi POST, koji se koristi za stvaranje novog resursa na serveru. Podaci koji se šalju idu u tijelu zahtjeva. PUT se koristi za ažuriranje postojećeg resursa ili stvaranje resursa ako ne postoji. Podaci se također šalju u tijelu zahtjeva. Slična metoda, PATCH, koristi se za djelomično ažuriranje resursa. Samo određeni dijelovi resursa mijenjaju se umjesto cijelog. Konačno, DELETE se koristi za brisanje određenog resursa. Na slici 2.8 se uočavaju HTTP metode.



Slika 2.8: Prikaz HTTP metoda

Primjer, GET/user/Marko bi bio zahtjev za dohvaćanje podataka korisnika Marka. Primjer URL-a s POST metodom: POST/user/Ivan, bi bio zahtjev za stvaranje korisnika Ivana. Primjer, PUT/user/Marko bi bio zahtjev za ažuriranje korisnika Marka. Primjer URL-a s PATCH

metodom: PATCH/user/Marko, bi bio zahtjev za djelomično ažuriranje korisnika Marka. Primjer, DELETE/user/Ivan bi bio zahtjev za brisanje korisnika Ivana.

Što se tiče idempotentnosti, to znači da isti zahtjev može biti izvršen više puta, ali će rezultat biti isti kao i prvi put. Na primjer, PUT i DELETE metode su idempotentne jer neovisno koliko puta ih izvršite, rezultat će biti isti ako su uvjeti isti. GET je također idempotentan jer ne mijenja stanje resursa na serveru. Važno je napomenuti da su PUT i DELETE metode idempotentne, što znači da ih možete izvršiti više puta, ali će rezultat biti isti ako su uvjeti isti. GET također spada u kategoriju idempotentnih metoda.

2.5.4. Arhitektura REST-a

REST protokol je prilagođen mrežnim aplikacijama, posebno onima temeljenim na modelu klijent-poslužitelj, ističe se svojom sposobnošću prilagodbe masovnoj upotrebi na internetu. Ključna mu je značajka snažna razdvojenost između klijenata i izvornog poslužitelja, potičući tako široko usvajanje. Ova snažna razdvojenost, s prijenosom informacija putem uniformnog adresiranja, čini temelj za ispunjenje ključnih zahtjeva Weba, uključujući robusnost, neovisno implementiranje komponenata, prijenos velikih količina podataka te smanjenje prepreka za korisnika.

Ograničenja REST arhitektonskog stila značajno oblikuju ključna arhitektonska svojstva. Performanse u interakcijama komponenata postaju ključne za percepciju korisničkih performansi i efikasnost mreže. Skalabilnost omogućava podršku velikom broju komponenata i njihovih interakcija, dok jednostavnost uniformnog sučelja pojednostavljuje arhitekturu. Modifikabilnost komponenata omogućava prilagodbu promjenjivim potrebama, čak i tijekom izvođenja aplikacije. Vidljivost komunikacije između komponenata olakšana je servisnim agentima. Prenosivost komponenata postiže se pomicanjem programske logike s podacima. Pouzdanost se očituje u otpornosti na neuspjeh na razini sustava, čak i u prisutnosti problema unutar komponenata, konektora ili podataka.

REST arhitektura definira se sa šest smjernih ograničenja koja, kada se primijene na arhitekturu sustava, donose svojstva poput skalabilnosti, jednostavnosti, izmjenjivosti, vidljivosti, prenosivosti i pouzdanosti. Ova ograničenja uključuju arhitekturu klijent-poslužitelj, promičući razdvajanje brige o korisničkom sučelju od brige o pohrani podataka. Apatridnost osigurava da primatelj, obično poslužitelj, ne zadržava informacije o sesiji.

Mogućnost keširanja omogućuje klijentima i posrednicima keširanje odgovora, poboljšavajući skalabilnost i performanse. Sustav u slojevima skriva izravnu vezu između klijenata i poslužitelja, omogućujući skalabilnost, sigurnost i izvršavanje koda po potrebi za privremeno proširenje funkcionalnosti. Ograničenje jedinstvenog sučelja, temeljno za RESTful sustave, pojednostavljuje i odvaja arhitekturu, naglašavajući identifikaciju resursa, manipulaciju putem prikaza i hipermediju kao pokretač aplikacijskog stanja, omogućujući dinamičko otkrivanje resursa bez unaprijed definiranja strukture poslužitelja u klijentu. [13]

3. APLIKACIJSKO PROGRAMSKO SUČELJE

Kako bi zadatak bio uspješno odrađen, za početak potrebno je analizirati zadani problem, te smisliti cjelokupnu strukturu zadatka. Prvi korak je kreiranje aplikacijsko programskog sučelja za izračun cijene uvoza automobila, te potom izrada android aplikacije koja omogućuje korištenje istog sučelja. Ovo poglavlje sadrži detaljan opis kreiranja aplikacijskog programskog sučelja.

3.1. Izrada REST aplikacijsko programskog sučelja

Za pisanje koda koji komunicira s REST aplikacijsko programskim sučeljima, potrebno je koristi requests [14] za slanje HTTP zahtjeva. Ova biblioteka olakšava proces izrade HTTP zahtjeva, jedan je od projekata koji se smatra gotovo dijelom standardne knjižnice. Kako bi mogli koristiti biblioteku requests potrebno ju je prvo instalirati u terminalu. Slika 3.1. prikazuje instalaciju requestsa.

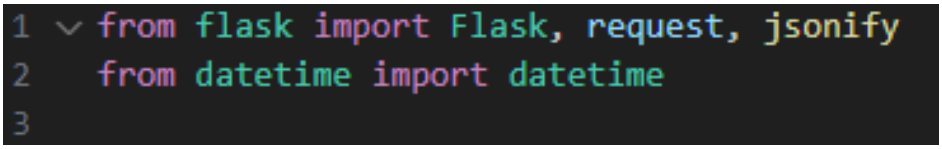
```
PS C:\Users\Šimun\Desktop\mateo> pip install requests
Requirement already satisfied: requests in d:\mateo\py\lib\site-packages (2.31.0)
Requirement already satisfied: charset-normalizer<4,>=2 in d:\mateo\py\lib\site-packages (from requests) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in d:\mateo\py\lib\site-packages (from requests) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in d:\mateo\py\lib\site-packages (from requests) (2.1.0)
Requirement already satisfied: certifi>=2017.4.17 in d:\mateo\py\lib\site-packages (from requests) (2023.11.17)

[notice] A new release of pip is available: 23.2.1 -> 23.3.2
[notice] To update, run: D:\mateo\py\python.exe -m pip install --upgrade pip
```

Slika 3.1: Prikaz instalacije requests pomoću pip

Sljedeći korak je kreiranje nove .py datoteke te uvoz request, flask i jsonify [15] biblioteka iz Flaska. Također uvozimo i datetime biblioteku [16] koja će nam kasnije biti potrebna za funkciju kojom računamo cijenu uvoza. Flask je web okvir za Python, a request i jsonify koriste se za

rukovanje HTTP zahtjevima i stvaranje JSON odgovora. Modul datetime se uvozi za rad s datumima i vremenom. Slika 3.2. prikazuje uvoz prethodno navedenih biblioteka.



Slika 3.2: Prikaz uvoza potrebnih biblioteka

Nakon toga stvara se instanca klase Flask, pri čemu se kao parametar koristi __name__. Ovo je uobičajeni obrazac u Flasku koji određuje osnovnu putanju aplikacije, te nakon toga blok koji osigurava da se aplikacija pokreće samo ako se skripta izvršava izravno (a ne ako se uvozi kao modul). Kada se skripta pokrene, poziva se metoda run na instanci Flask aplikacije s parametrom debug=True. Ovo pokreće razvojni server i omogućuje način rada za otklanjanje grešaka. Ukratko, ova skripta postavlja osnovnu Flask web aplikaciju. Ako pokrenete ovu skriptu, pokrenut će se razvojni server, a aplikaciju možete pristupiti na http://127.0.0.1:5000/. Aplikacija još uvijek nema definirane rute ili funkcionalnosti, to je samo osnovno postavljanje. Prikaz stvaranja instance klase Flask prikazan je na slici 3.3.

Slika 3.3: Prikaz stvaranja instance klase Flask

3.2. Izrada funkcija za računanje cijene uvoza

Glavna funkcija je podijeljena na više manjih funkcija kako bi kod bio pregledniji, a i kasnije bi mnogo lakše bilo napraviti izmjene ako bude bilo potrebno ili nadogradnju koda.

Slika 3.4: Prikaz koda glavne funkcije calculateExpenses.

Na slici *3.4.* vidimo kod za funkciju. Funkcija prima 6 vrijednosti: 2 bool varijable isNew i isPetrol koje nam govore informacije o stanju vozila, je li ono novo ili rabljeno, te tip motora automobila, benzinski ili dizelski motor. Sljedeće 4 varijable redom nam daju informacije o emisiji CO_2 izraženoj u gramima po kilometru, cijeni automobila kada je bio nov (u eurima), te godini i mjesecu prve registracije. Sljedećih 6 varijabli koje su postavljene na 0 će biti potrebne za daljnji rad funkcije. Također računamo i starost automobila u mjesecima s pomoću biblioteke datetime koju smo prethodno uvezli.

```
19
20      def calculatePriceComponents(carPrice):
21          priceRanges = [
22              (0, 19908.42, 0, 0),
23              (19908.42, 26544.56, 265.45, 0.03),
24              (26544.56, 33180.70, 464.53, 0.05),
25              (33180.71, 39816.84, 796.34, 0.07),
26              (39816.85, 46452.98, 1260.87, 0.09),
27              (46452.99, 53089.12, 1858.12, 0.11),
28              (53089.13, 59725.26, 2588.10, 0.13),
29              (59725.27, 66361.40, 3450.79, 0.14),
30              (66361.41, 72997.54, 4379.85, 0.15),
31              (72997.55, 79633.69, 5375.27, 0.16),
32              (79633.69, float('inf'), 6437.05, 0.17)
33          ]
34
35          for rangeMin, rangeMax, VN, PCrate in priceRanges:
36              if rangeMin <= carPrice <= rangeMax:
37                  PC = (carPrice - rangeMin) * PCrate
38                  return VN, PC
39
40      VN, PC = calculatePriceComponents(carPrice)
```

Slika 3.5: Primjer koda pomoćne funkcije calculatePriceCompotents.

Na slici 3.5. pomoćna funkcija prima cijenu automobila u eurima kada je bio nov, te s pomoću nje računa vrijednosnu naknadu u eurima (VN) i naknadu koja se utvrđuje način da se od prodajne cijene motornog vozila oduzme najniži iznos za skupinu kojoj motorno vozilo pripada i tako dobiveni iznos pomnoži s postotkom utvrđenim za skupinu kojoj motorno vozilo pripada. Definira se lista priceRanges koja sadrži n-torke s četiri vrijednosti: (donja granica raspona, gornja granica raspona, VN, PCrate). Svaka n-torka opisuje jedan raspon cijena.

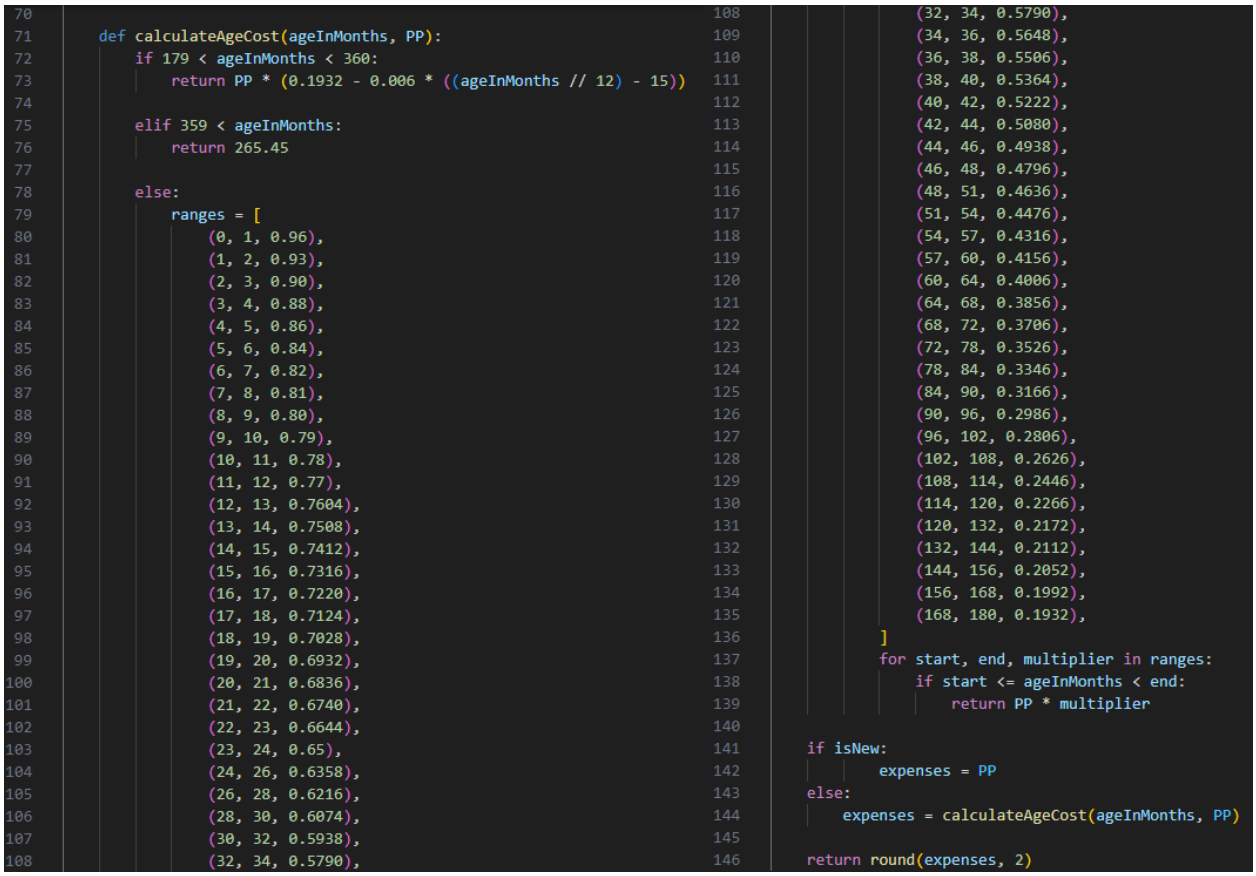
Zatim se prolazi kroz svaku n-torku u priceRanges, ako carPrice pripada trenutnom rasponu, izračunava se PC prema formuli (carPrice - rangeMin) * PCrate, te se vraćaju vrijednost VN i izračunata komponenta PC. [17]

```
41
42      def calculateEmissionCosts(co2Emission, isPetrol):
43          if isPetrol:
44              ranges = [
45                  (75, 90, 12.61, 4.65),
46                  (90, 120, 82.36, 17.92),
47                  (120, 140, 619.96, 59.73),
48                  (140, 170, 1814.56, 92.91),
49                  (170, 200, 4601.86, 159.27),
50                  (200, float('inf'), 9379.96, 172.54)
51              ]
52          else:
53              ranges = [
54                  (70, 85, 24.55, 7.30),
55                  (85, 120, 134.05, 23.23),
56                  (120, 140, 947.10, 152.63),
57                  (140, 170, 3999.70, 165.90),
58                  (170, 200, 8976.70, 179.18),
59                  (200, float('inf'), 14352.10, 192.45)
60              ]
61
62          for rangeMin, rangeMax, ON, ENrate in ranges:
63              if rangeMin <= co2Emission <= rangeMax:
64                  EN = (co2Emission - rangeMin) * ENrate
65                  return ON, EN
66
67      ON, EN = calculateEmissionCosts(co2Emission, isPetrol)
68
69      PP = (VN + PC) + (ON + EN)
```

Slika 3.6: Prikaz koda pomoćne funkcije calculateEmissionCosts

Na slici 3.6 pomoćna funkcija prima 2 argumenta co2Emission (emisija CO_2) i isPetrol (zastava koja označava je li vozilo na benzin), te izračunava osnovnu naknadu u eurima (ON) i naknadu koja se utvrđuje tako da se od iznosa prosječne emisije ugljičnog dioksida (CO_2) motornog vozila oduzme najniži iznos za skupinu kojoj motorno vozilo pripada i tako dobiveni iznos pomnoži s pripadajućim iznosom u eurima za jedan g/km CO_2. Također, vidimo da se sada može izračunati iznos posebnog poreza u eurima (PP).

Ako je vozilo na benzin (isPetrol je True), koriste se rasponi za benzin, inače se koriste rasponi za dizel. Zatim se prolazi kroz svaki tuple u ranges. Ako co2Emission pripada trenutnom rasponu, izračunava se EN prema formuli (co2Emission - rangeMin) * ENrate, te se vraćaju vrijednost ON i izračunati trošak EN. [18]



```
def calculateAgeCost(ageInMonths, PP):
    if 179 < ageInMonths < 360:
        return PP * (0.1932 - 0.006 * ((ageInMonths // 12) - 15))

    elif 359 < ageInMonths:
        return 265.45

    else:
        ranges = [
            (0, 1, 0.96),
            (1, 2, 0.93),
            (2, 3, 0.90),
            (3, 4, 0.88),
            (4, 5, 0.86),
            (5, 6, 0.84),
            (6, 7, 0.82),
            (7, 8, 0.81),
            (8, 9, 0.80),
            (9, 10, 0.79),
            (10, 11, 0.78),
            (11, 12, 0.77),
            (12, 13, 0.7604),
            (13, 14, 0.7508),
            (14, 15, 0.7412),
            (15, 16, 0.7316),
            (16, 17, 0.7220),
            (17, 18, 0.7124),
            (18, 19, 0.7028),
            (19, 20, 0.6932),
            (20, 21, 0.6836),
            (21, 22, 0.6740),
            (22, 23, 0.6644),
            (23, 24, 0.65),
            (24, 26, 0.6358),
            (26, 28, 0.6216),
            (28, 30, 0.6074),
            (30, 32, 0.5938),
            (32, 34, 0.5790),
            (34, 36, 0.5648),
            (36, 38, 0.5506),
            (38, 40, 0.5364),
            (40, 42, 0.5222),
            (42, 44, 0.5080),
            (44, 46, 0.4938),
            (46, 48, 0.4796),
            (48, 51, 0.4636),
            (51, 54, 0.4476),
            (54, 57, 0.4316),
            (57, 60, 0.4156),
            (60, 64, 0.4006),
            (64, 68, 0.3856),
            (68, 72, 0.3706),
            (72, 78, 0.3526),
            (78, 84, 0.3346),
            (84, 90, 0.3166),
            (90, 96, 0.2986),
            (96, 102, 0.2806),
            (102, 108, 0.2626),
            (108, 114, 0.2446),
            (114, 120, 0.2266),
            (120, 132, 0.2172),
            (132, 144, 0.2112),
            (144, 156, 0.2052),
            (156, 168, 0.1992),
            (168, 180, 0.1932),
        ]
        for start, end, multiplier in ranges:
            if start <= ageInMonths < end:
                return PP * multiplier

if isNew:
        expenses = PP
else:
    expenses = calculateAgeCost(ageInMonths, PP)

return round(expenses, 2)
```

Slika 3.7: Prikaz koda pomoćne funkcije calculateAgeCost

Na slici *3.7.* vidimo pomoćnu funkciju calculateAgeCost koja prima 2 argumenta: starost vozila u mjesecima te iznos posebnog poreza u eurima (PP). Ako je dob vozila između 15 i 30 godina (izražena u mjesecima), tada se za svaku dodatnu godinu oduzima 0.6 % od početnih 19.32 % , te se množi s iznosom posebnog poreza u eurima (PP).

Ako je dob vozila veća od 30 godina, vraća se fiksni iznos troškova od 265.45. Odnosno ako je vozilo oldtimer, tj starije od 30 godina, posebni porez u eurima (PP) će iznositi 265.45 eura, bez obzira na početnu cijenu ili emisiju CO_2. Ako dob vozila ne spada ni u jedan od prethodnih uvjeta, tada se koriste unaprijed definirani rasponi dobi (ranges). Svaki raspon ima svoj početak (start), kraj (end) i multiplikator (multiplier). Ako dob vozila pripada određenom rasponu, izračunava se trošak prema formuli PP * multiplier.

Za kraj glavne funkcije calculateExpenses se provjerava je li vozilo novo ili ne, ako je, prethodna funkcija se ne koristi te je iznos jednak početnom iznosu posebnog poreza u eurima (PP). Ako vozilo nije novo pozivamo prethodnu funkciju da nam izračuna iznos koji je ujedno i glavni iznos. [19]

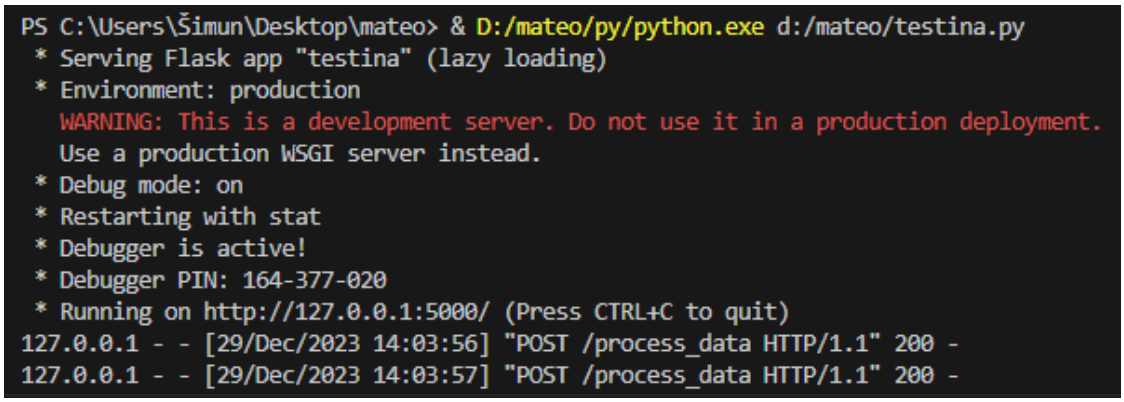
3.3. Izrada HTTP zahtjeva

Kako aplikacija još uvijek nema definirane rute ili funkcionalnosti, moramo dodati rute i definirati što se treba dogoditi kada se pristupi različitim krajevima.

```
147
148 @app.route('/process_data', methods=['POST'])
149 def calculate_expenses_route():
150     data = request.get_json()
151
152     required_keys = ['isNew', 'isPetrol', 'co2', 'cijenaAuta', 'godina', 'mjesec']
153     if not all(key in data for key in required_keys):
154         return jsonify({'error': 'Missing or invalid keys in the JSON payload'}), 400
155
156     isNew = data.get('isNew')
157     isPetrol = data.get('isPetrol')
158     co2 = data.get('co2')
159     cijenaAuta = data.get('cijenaAuta')
160     godina = data.get('godina')
161     mjesec = data.get('mjesec')
162
163     try:
164         result = calculateExpenses(isNew, isPetrol, co2, cijenaAuta, godina, mjesec)
165         return jsonify({'result': result})
166     except ValueError as e:
167         return jsonify({'error': str(e)}), 400
168
```

Slika 3.8: Prikaz koda koji obrađuje podatke poslane putem HTTP POST zahtjeva

Na slici 3.8. koristi se @app.route dekorator kako bi se stvorila ruta /process_data koja reagira samo na HTTP POST zahtjeve. Kada netko pošalje POST zahtjev na ovu rutu, izvršava se funkcija calculate_expenses_route. Dohvaćaju se JSON podaci poslani u tijelu POST zahtjeva s pomoću request.get_json(). Provjerava se jesu li svi obavezni ključevi prisutni u primljenim podacima. Ako nedostaju ili su neispravni, vraća se odgovor s odgovarajućom greškom. Dohvaćaju se vrijednosti pojedinih ključeva iz JSON podataka. Poziva se funkcija calculateExpenses s dobivenim podacima. Ako funkcija izračuna rezultat bez greške, rezultat se vraća u JSON formatu. Ako se dogodi ValueError (greška), vraća se odgovor s odgovarajućom greškom i status kodom 400. Greška sa status kodom 400 je HTTP statusni kod koji opisuje pogrešku uzrokovanu nevaljanim zahtjevom. Dakle, poslužitelj ga ne može razumjeti niti obraditi.



Slika 3.9: Prikaz pokretanja Flask servera

Programsko aplikacijsko sučelje završeno, pokreće se na adresi http://127.0.0.1:5000/. Ovo je lokalna adresa na računalu. Možemo otvoriti web preglednik i posjetiti ovu adresu kako bi vidjeli aplikaciju. Ukratko na slici 3.9., Flask server je pokrenut, i aplikacija trenutno prima POST zahtjeve na rutu /process_data. Zadnje dvije linije su log poruke koje pokazuju da su primljeni POST zahtjevi na rutu /process_data i da su ti zahtjevi uspješno obrađeni s status kodom 200. Status kod 200 je standardni odgovor za uspješne HTTP zahtjeve. Stvarni odgovor će ovisiti o korištenoj metodi zahtjeva. U GET zahtjevu, odgovor će sadržavati entitet koji odgovara traženom resursu. U POST zahtjevu, odgovor će sadržavati entitet koji opisuje ili sadrži rezultat akcije.

4. APLIKACIJA

Sada kada smo kreirali aplikacijsko programsko sučelje, sljedeći korak je izrada aplikacije.

Aplikacija se sastoji od:

1 aktivnosti (*MainActivity*)

1 sučelja (*ApiService*)

2 klase (*RequestData i ResponseData).*

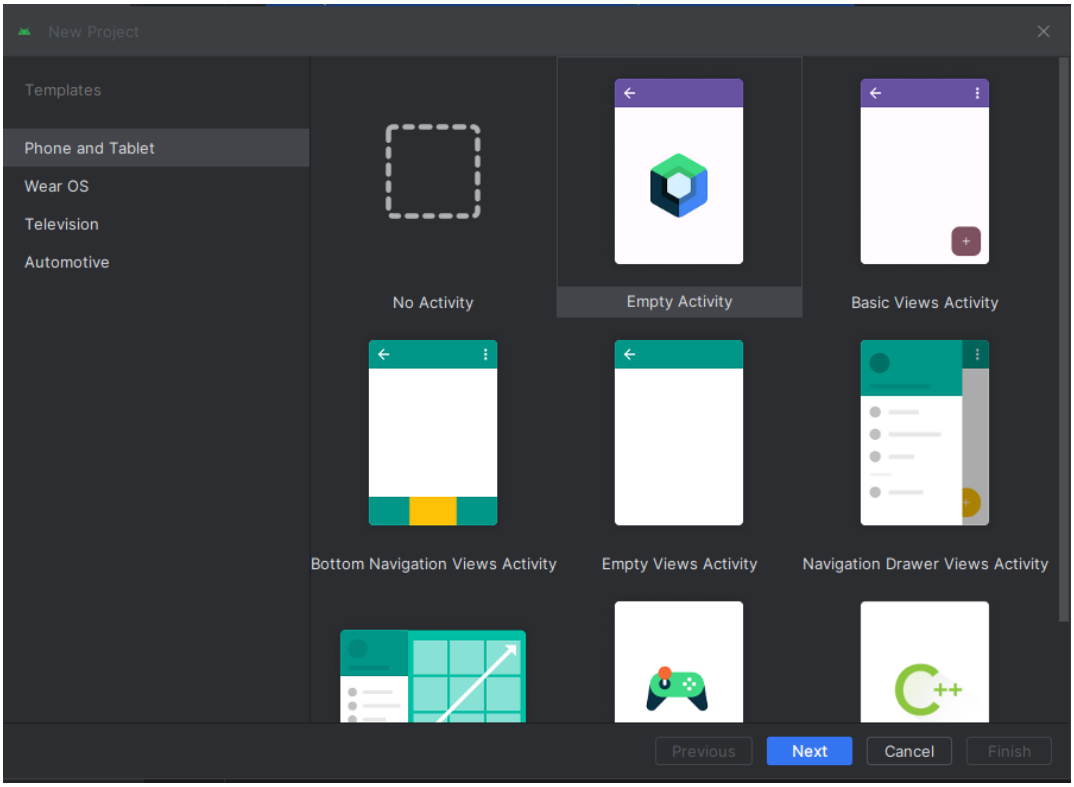
Aktivnost (engl. *Activity*) je klasa koja označava jedan zaslon aplikacije. Prva aktivnost se stvara prilikom kreiranja projekta i automatski se postavlja kao početni zaslon koji se prikazuje pri pokretanju aplikacije. Za razliku od klasičnih desktop aplikacija, Android aplikacije nemaju glavnu metodu koja služi kao ulazna točka programa. Umjesto toga, pojedine aktivnosti oslanjaju se na određene metode s povratnim rezultatom (engl. Callback). Ove metode se pozivaju tijekom stvaranja Activitya ili prilikom njegovog gašenja, a metode životnog ciklusa uključuju onCreate, onStart, onResume, onPause, onStop, onRestart i onDestroy. S obzirom na odvajanje sučelja od koda, nužno je kreirati objekte koji predstavljaju elemente korisničkog sučelja iz XML opisa. Ovo se postiže pozivanjem metode setContentView, kojoj se kao argument predaje resurs koji definira sučelje za aktivnost.

Sučelje (engl. *interface*) predstavlja skup konstanti, metoda (apstraktnih, statičkih i zadanih) te ugniježđenih tipova. Klase koje implementiraju sučelje moraju naslijediti sve njegove metode i definirati ih unutar same klase. Sučelje je slično klasi, a ključna riječ "sučelje" koristi se za deklaraciju samog sučelja unutar određenih klasa.

Klasa podataka (engl. *data class*) je klasa koja uključuje samo svojstva i metode za pristup tim atributima. Ove klase djeluju kao jednostavni spremnici podataka koje druge klase koriste, a same po sebi nemaju vlastitu konkretnu funkcionalnost. Često se koriste za stvaranje objekata koji se pohranjuju u bazu podataka ili se čitaju iz nje.

4.1. Izrada i postavljanje aplikacije

Prvi korak je kreiranje novog projekta u Android Studiu. Odabran je *EmptyActivity* predložak koji nam kreira jednu aktivnost (*MainActivity*). Slika 4.1. prikazuje postupak kreiranja novog projekta.



Slika 4.1: Prikaz kreiranja novog projekta

Sljedeći korak je deklariranje dozvole za korištenje interneta u AndroidManifest.xml datoteci. To činimo dodavanjem jedne linije koda u istu datoteku. To je važno ako aplikacija treba dohvaćati ili slati podatke preko interneta, kao što je pristupanje web servisima, dohvaćanje podataka sa servera ili bilo koja druga komunikacija putem mreže. Bez ove dozvole, aplikacija može imati ograničen pristup internetu. U našem slučaju ovo nam koristi kako bi mogli komunicirati s prethodno izrađenim aplikacijsko programskim sučeljem. Na slici 4.2. deklarirana je dozvola pristupa internetu.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools">
4
5      <uses-permission android:name="android.permission.INTERNET" />
```

Slika 4.2: Prikaz deklariranja dozvole pristupa internetu

Još preostaje dodati dvije biblioteke koje su često korištene Android razvoju za rad s mrežnim zahtjevima, posebno za izradu HTTP zahtjeva i obradu odgovora u formatu JSON. Postignuto je isto dodavanjem 2 linije koda u build.gradle tj. sustavu za upravljanje Android ovisnostima.

Prva linija dodaje Retrofit [20] biblioteku verzije 2.9.0. Retrofit je popularna biblioteka za izradu HTTP zahtjeva u Android aplikacijama. Omogućuje lakšu komunikaciju s web servisima, definiranje API poziva i obradu odgovora. Sljedeća linija dodaje Gson konverter za Retrofit verzije 2.9.0. Gson [21] je biblioteka koja olakšava pretvaranje JSON formata podataka u objekte Java programa i obratno. U kombinaciji s Retrofitom, omogućuje jednostavno parsiranje JSON odgovora prilikom komunikacije s web servisima. Nakon toga samo još moramo sinkronizirati nove ovisnosti da bi aplikacija implementirala nove biblioteke. To je prikazano na slici 4.3.

```
54  dependencies { this: DependencyHandlerScope
55      implementation("com.squareup.retrofit2:retrofit:2.9.0")
56      implementation("com.squareup.retrofit2:converter-gson:2.9.0")
```

Slika 4.3: Prikaz Android ovisnosti koje su potrebne

4.2. Izrada klasa i sučelja

Kreirano je i sučelje za komunikaciju s aplikacijsko programskim sučeljem, za komunikaciju se koristi Retrofit biblioteka, koja se često koristi za rad s HTTP zahtjevima u Android aplikacijama koje je prikazano na slici 4.4.

```
1  package com.example.uvoz.ui.theme
2
3  import retrofit2.Call
4  import retrofit2.http.Body
5  import retrofit2.http.POST
6
7  interface ApiService {
8      @POST("/process_data")
9      fun processTheData(@Body requestData: RequestData): Call<ResponseData>
10 }
```

Slika 4.4: Prikaz koda ApiService sučelja

Ovo sučelje definira metodu za slanje HTTP zahtjeva u obliku POST zahtjeva na određeni URL, u ovom slučaju URL je "/process_data", pri čemu se podaci šalju u tijelu zahtjeva u obliku objekta RequestData, a očekuje se odgovor u obliku objekta ResponseData. Ove tipove podataka trebamo definirati prema specifikaciji web usluge s kojom komuniciramo.

Sada kreiramo i dvije podatkovne klase RequestData i ResponseData koje koristimo za komunikaciju s aplikacijsko programskim sučeljem. Klasa RequestData se koristi za predstavljanje podataka koji se šalju u zahtjevu, a klasa ResponseData predstavlja podatke koji se očekuju kao odgovor.

RequestData klasa predstavlja podatke koji se šalju u zahtjevu prema web usluzi, sadrži ove atribute: isNew (Boolean vrijednost koja označava je li automobil novi), isPetrol (Boolean vrijednost koja označava tip motora automobila), co2 (Integer vrijednost koja predstavlja emisiju CO_2), cijenaAuta (Integer vrijednost koja označava cijenu automobila), godina (Integer vrijednost koja označava godinu prve registracije) i mjesec (Integer vrijednost koja označava mjesec prve registracije). ResponseData klasa predstavlja podatke koji se očekuju kao odgovor od web usluge, sadrži 1 atribut: result (Float vrijednost koja predstavlja cijenu uvoza). Ove atribute definira konstruktor klase, a ključna riječ data označava da će Kotlin automatski generirati metode poput toString(), equals(), i hashCode() na temelju svojstava klase. Slika 4.5. prikazuje dana klasu.

```
4  data class RequestData(
5      val isNew: Boolean,
6      val isPetrol: Boolean,
7      val co2: Int,
8      val cijenaAuta: Int,
9      val godina: Int,
10     val mjesec: Int
11 )
12
13 data class ResponseData(
14     val result: Float
15 )
```

Slika 4.5: Prikaz koda data klasa

4.3. Aktivnost unutar aplikacije

Aplikacija sadrži samo jednu aktivnost, odnosno glavnu aktivnost. Detaljan prolazak koda na slici 4.6. nam omugućuje njegovo objašnjenje.

```
class MainActivity : AppCompatActivity() {
    private lateinit var textView: TextView
    private lateinit var switch: Switch
    private lateinit var linearLayout: LinearLayout
    private lateinit var apiService: ApiService

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        textView = findViewById(R.id.textView)
        switch = findViewById(R.id.switch2isNew)
        linearLayout = findViewById(R.id.editlinearlayout)

        val retrofit = Retrofit.Builder()
            .baseUrl("http://10.0.2.2:5000/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        apiService = retrofit.create(ApiService::class.java)
```

Slika 4.6: Prikaz koda MainActivity

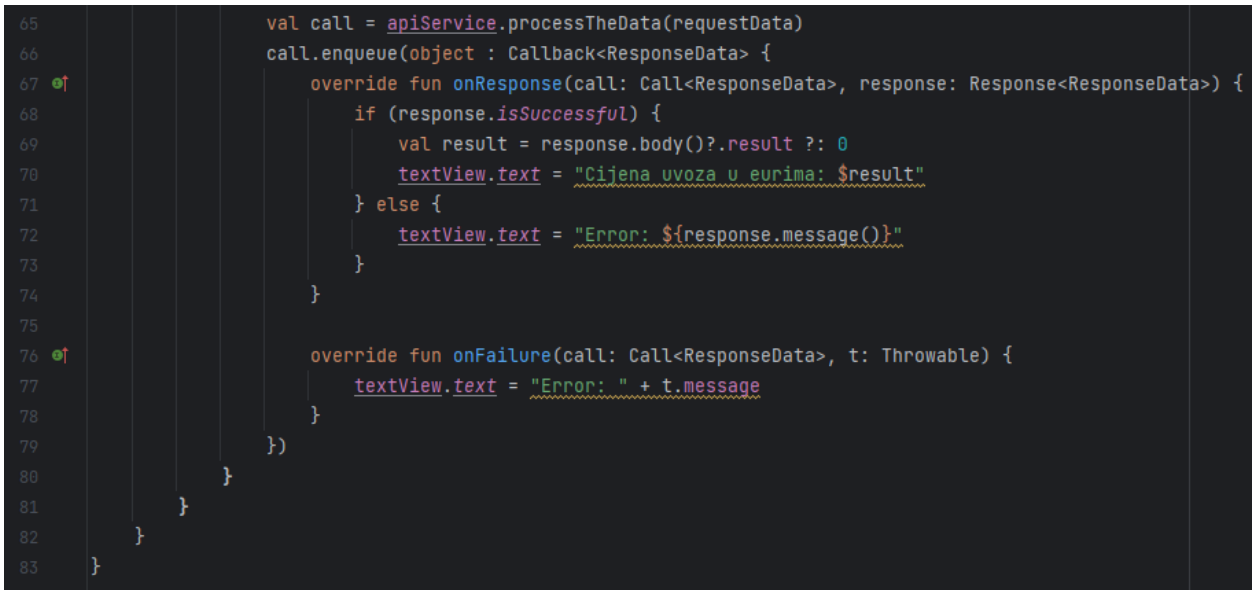
Definiramo klasu MainActivity koja nasljeđuje funkcionalnosti AppCompatActivity. AppCompatActivity je dio Android Jetpack biblioteke i pruža podršku za moderni dizajn aplikacija na starijim verzijama Androida. Nakon toga deklariramo 4 privatne varijable: textView (privatna varijabla textView koja će kasnije biti inicijalizirana kao objekt tipa TextView, TextView je komponenta za prikazivanje teksta na korisničkom sučelju), switch (privatna varijabla switch koja će kasnije biti inicijalizirana kao objekt tipa Switch, Switch je komponenta za prekidače na korisničkom sučelju), linearLayout (privatna varijabla linearLayout koja će kasnije biti inicijalizirana kao objekt tipa LinearLayout, LinearLayout je rasporeditelj koji organizira svoje dječje elemente u jedan red ili stupac) i apiService (privatna varijabla apiService koja će kasnije biti inicijalizirana kao objekt sučelja ApiService, ova varijabla će se koristiti za komunikaciju s web uslugom putem Retrofit biblioteke).

Postavlja se sadržaj aktivnosti koristeći layout definiran u activity_main.xml datoteci i inicijalizacija privatnih varijabli s pripadajućim XML elementima korisničkog sučelja putem njihovih identifikatora. Stvaranje objekta Retrofit koji će se koristiti za komunikaciju s web uslugom. Postavljena je baza URL-a, dodan je konverter za Gson kako bi se omogućila konverzija JSON podataka, i izgrađen je sam objekt Retrofit.

```
37      switch.setOnCheckedChangeListener { _, isChecked ->
38          linearLayout.visibility = if (isChecked) View.GONE else View.VISIBLE
39
40          val isNewsw = findViewById<Switch>(R.id.switch2isNew)
41          val isPetrolsw = findViewById<Switch>(R.id.switch3isPetrol)
42          val co2text = findViewById<EditText>(R.id.editTextco2)
43          val cijenaAutatext = findViewById<EditText>(R.id.editTextcijenaAuta)
44          val godinatext = findViewById<EditText>(R.id.editTextgodina)
45          val mjesectext = findViewById<EditText>(R.id.editTextmjesec)
46
47          val buttonSubmit = findViewById<Button>(R.id.button)
48          buttonSubmit.setOnClickListener { it: View!
49              val isNewValue = isNewsw.isChecked
50              val isPetrolValue = isPetrolsw.isChecked
51              val co2Value = co2text.text.toString().toIntOrNull() ?: 0
52              val cijenaAutaValue = cijenaAutatext.text.toString().toIntOrNull() ?: 0
53              val godinaValue = godinatext.text.toString().toIntOrNull() ?: 0
54              val mjesecValue = mjesectext.text.toString().toIntOrNull() ?: 0
55
56              val requestData = RequestData(
57                  isNew = isNewValue,
58                  isPetrol = isPetrolValue,
59                  co2 = co2Value,
60                  cijenaAuta = cijenaAutaValue,
61                  godina = if (isChecked) 2015 else godinaValue,
62                  mjesec = if (isChecked) 6 else mjesecValue
63              )
```

Slika 4.7: Prikaz nastavka koda MainActivity

Na slici 4.7 koristi se setOnCheckedChangeListener za praćenje promjena u stanju Switch komponente. Ova funkcionalnost koristi promjene stanja Switch komponente da dinamički postavlja vidljivost drugog dijela sučelja (linearLayout). Također, prikuplja podatke iz drugih komponenata (prekidača, tekstualnih polja, gumba) i stvara objekt RequestData koji se koristi u daljnjem dijelu koda. U linijama 61 i 62, ako je switch stavljen u stanje isChecked (znači stanje automobila je novo, te još nije bilo prve registracije), godina i mjesec su automatski postavljeni u 2015 i 6 samo kako bi upit bio poslan (potrebne su sve varijable za uspješan upit), glavna metoda u aplikacijsko programskom sučelju uopće ne uzima u obzir te 2 varijable kada je automobili novi.



```
                val call = apiService.processTheData(requestData)
                call.enqueue(object : Callback<ResponseData> {
                    override fun onResponse(call: Call<ResponseData>, response: Response<ResponseData>) {
                        if (response.isSuccessful) {
                            val result = response.body()?.result ?: 0
                            textView.text = "Cijena uvoza u eurima: $result"
                        } else {
                            textView.text = "Error: ${response.message()}"
                        }
                    }

                    override fun onFailure(call: Call<ResponseData>, t: Throwable) {
                        textView.text = "Error: " + t.message
                    }
                })
            }
        }
    }
}
```

Slika 4.8: Prikaz nastavka koda MainActivity

Dio koda prikazan na slici 4.8. koristi Retrofit biblioteku za slanje HTTP zahtjeva na određenu API rutu i obrađuje odgovor. Ovdje se koristi asinkrona metoda enqueue za slanje HTTP poziva. U onResponse se obrađuje uspješan odgovor, dok se u onFailure obrađuje situacija u kojoj poziv nije uspio. Ako je odgovor uspješan, prikuplja se rezultat iz tijela odgovora i postavlja u textView. U suprotnom, postavlja se tekst s porukom o grešci.

4.4. XML kod

Slike, izgled i ostali podaci su odvojeni od ostatka koda i nazivaju se resursima, smješteni su u posebnu mapu. Izgled aplikacije se definira putem layouta, koji su XML datoteke opisnog jezika. Svaki layout mora imati korijenski element, koji je View ili ViewGroup objekt. U taj korijenski element se dodaju ostali elementi, stvarajući hijerarhiju. View predstavlja temeljni blok korisničkog sučelja, oblikovan kao mali pravokutni okvir. Prilikom pokretanja aplikacije otvara se početni zaslon, odnosno MainActivity aktivnost. Na slici 4.9. prikazan je početak XML koda.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:background="@color/teal_700">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="match_parent"
        android:layout_height="172dp"
        app:srcCompat="@drawable/dizajn_bez_naslova" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_marginTop="16dp"
        android:layout_height="40dp"
        android:background="@color/teal_700"
        android:orientation="horizontal">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_gravity="center_vertical"
            android:layout_marginEnd="8dp"
            android:layout_weight="1"
            android:text="Rabljeni Automobil"
            android:textColor="#333"
            android:textSize="16sp"/>
```

Slika 4.9: Prikaz XML koda

TextView je klasa koja služi za prikazivanje tekstualnog sadržaja koji nije podložan promjenama ili uređivanju od strane korisnika. Kako biste dodali TextView unutar vašeg izgleda, potrebno je u XML datoteci ubaciti oznaku <TextView>. Visina i širina elementa definiraju se s pomoću svojstava layout_width i layout_height, dok svojstvo text određuje sadržaj unutar elementa.

Klasa ImageView koristi se za prikaz slika na zaslonu. Da biste je dodali unutar izgleda, koristite oznaku <ImageView> u XML datoteci, pri čemu su sva korištena svojstva prethodno opisana i objašnjena.

LinearLayout klasa se koristi u Android razvoju za organizaciju elemenata korisničkog sučelja. To je jedan od osnovnih rasporeda u Androidu, koji postavlja svoju djecu u jedan red ili stupac, ovisno o odabranoj orijentaciji.s postavljenim atributima za širinu, visinu, razmak od vrha ekrana, pozadinsku boju i orijentaciju rasporeda djece. Kako biste dodali LinearLayout unutar vašeg izgleda, potrebno je u XML datoteci ubaciti oznaku <LinearLayout>.

```
33
34          <Switch
35              android:id="@+id/switch2isNew"
36              android:layout_width="wrap_content"
37              android:layout_height="wrap_content"
38              android:layout_centerHorizontal="true"
39              android:textOff="Used"
40              android:textOn="New"
41              android:layout_weight="1"
42              android:thumbTint="#F19595" />
43
44          <TextView
45              android:layout_width="wrap_content"
46              android:layout_height="match_parent"
47              android:layout_gravity="center_vertical"
48              android:layout_marginStart="8dp"
49              android:fontFamily="sans-serif"
50              android:layout_weight="1"
51              android:text="Novi Automobil"
52              android:textColor="#333"
53              android:textSize="16sp"/>
54      </LinearLayout>
55
56      <LinearLayout
57          android:layout_width="match_parent"
58          android:layout_height="40dp"
59          android:background="@color/teal_700"
60          android:orientation="horizontal">
```

Slika 4.10: Prikaz XML koda

Klasa Switch je grafički element korisničkog sučelja u Android razvoju koji omogućuje korisnicima da prebacuju između dvije opcije - uključeno ili isključeno. To je interaktivni prekidač koji korisnicima omogućuje jednostavno odabiranje između dvije mogućnosti s pomoću klizača. Da bi bio dodan unutar izgleda, potrebno je koristiti oznaku <Switch> u XML datoteci.

```
61
62          <TextView
63              android:layout_width="wrap_content"
64              android:layout_height="match_parent"
65              android:layout_gravity="center_vertical"
66              android:layout_marginEnd="8dp"
67              android:layout_weight="1"
68              android:text="Dizel"
69              android:textColor="#333"
70              android:textSize="16sp"/>
71
72          <Switch
73              android:id="@+id/switch3isPetrol"
74              android:layout_width="wrap_content"
75              android:layout_height="wrap_content"
76              android:layout_centerHorizontal="true"
77              android:textOff="Diesel"
78              android:layout_weight="1"
79              android:textOn="Petrol"
80              android:thumbTint="#F19595" />
81
82          <TextView
83              android:layout_width="wrap_content"
84              android:layout_height="match_parent"
85              android:layout_gravity="center_vertical"
86              android:layout_marginStart="8dp"
87              android:layout_weight="1"
88              android:text="Benzin"
89              android:textColor="#333"
90              android:textSize="16sp"/>
91      </LinearLayout>
```

Slika 4.11: Prikaz XML koda

```
94      <EditText
95          android:id="@+id/editTextco2"
96          android:layout_width="match_parent"
97          android:layout_height="wrap_content"
98          android:hint="CO2 Emisija Automobila"
99          android:ems="10"
100         android:inputType="number"
101         android:layout_marginTop="8dp" />
102
103     <EditText
104         android:id="@+id/editTextcijenaAuta"
105         android:layout_width="match_parent"
106         android:layout_height="wrap_content"
107         android:hint="Cijena Novog Automobila"
108         android:ems="10"
109         android:inputType="number"
110         android:layout_marginTop="8dp" />
111
112     <LinearLayout
113         xmlns:android="http://schemas.android.com/apk/res/android"
114         android:layout_width="match_parent"
115         android:layout_height="wrap_content"
116         android:orientation="horizontal"
117         android:background="@color/teal_700"
118         android:id="@+id/editlinearlayout">
119
120         <TextView
121             android:layout_width="wrap_content"
122             android:layout_height="wrap_content"
123             android:text="Godina i Mjesec Prve Registracije:"
124             android:layout_gravity="center_vertical"
125             android:textColor="#333"
126             android:textSize="15sp"/>
```

Slika 4.12: Prikaz XML koda

Klasa EditText se koristi za unos teksta od strane korisnika u Android aplikacijama. To je interaktivni grafički element koji omogućuje korisnicima da unesu ili urede tekst. Korisnici mogu dodirnuti polje EditText, unijeti tekst s pomoću virtualne tipkovnice na zaslonu i aplikacija može zatim koristiti uneseni tekst u skladu s potrebama. Da biste ga dodali unutar izgleda, koristite oznaku <EditText> u XML datoteci.

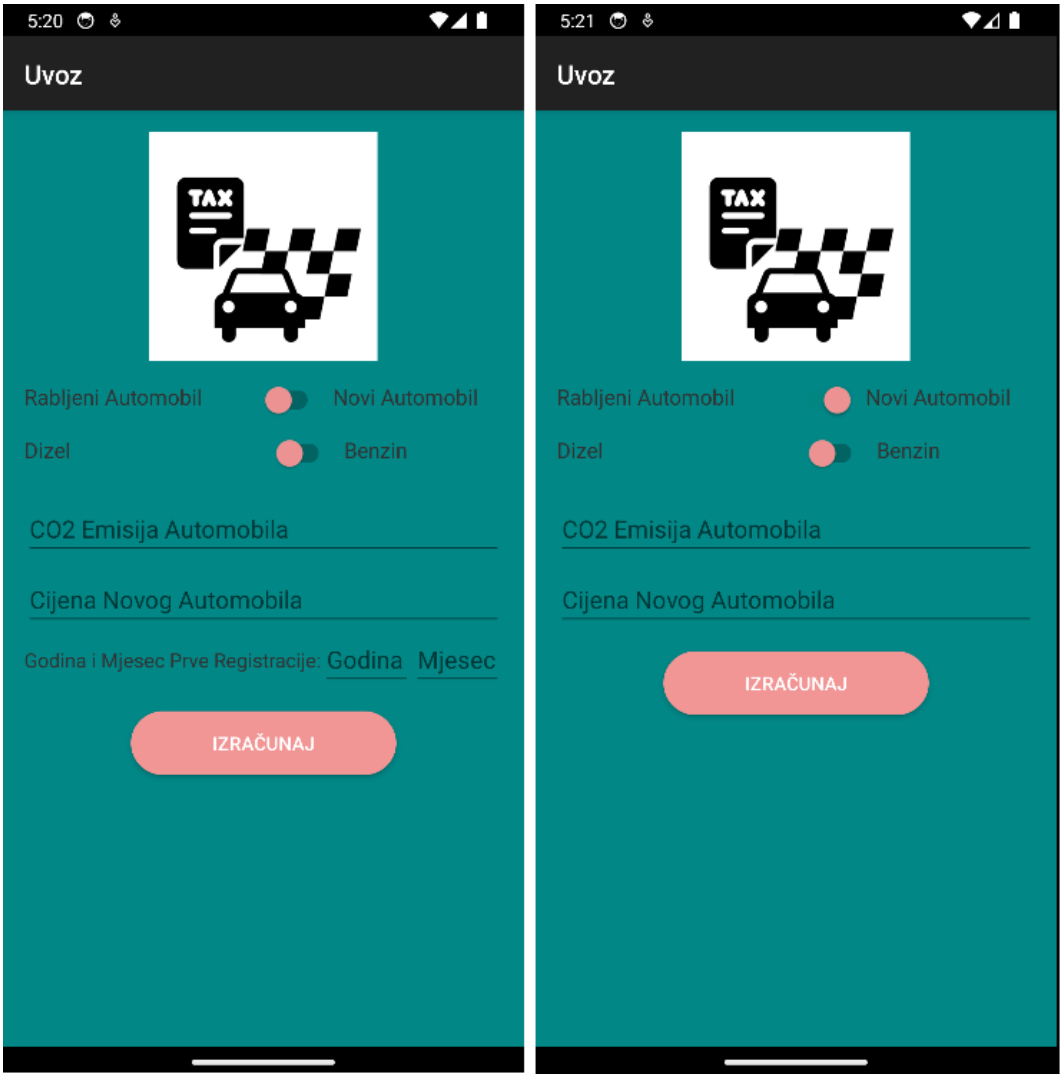
```
128         <EditText
129             android:id="@+id/editTextgodina"
130             android:layout_width="0dp"
131             android:layout_height="wrap_content"
132             android:layout_weight="1"
133             android:hint="Godina"
134             android:ems="10"
135             android:inputType="number"/>
136
137         <EditText
138             android:id="@+id/editTextmjesec"
139             android:layout_width="0dp"
140             android:layout_height="wrap_content"
141             android:layout_weight="1"
142             android:hint="Mjesec"
143             android:ems="10"
144             android:inputType="number"/>
145     </LinearLayout>
146
147     <Button
148         android:id="@+id/button"
149         android:layout_width="200dp"
150         android:layout_height="48dp"
151         android:layout_marginTop="16dp"
152         android:backgroundTint="#F19595"
153         android:background="@drawable/rounded_button"
154         android:layout_gravity="center_horizontal"
155         android:fontFamily="sans-serif-medium"
156         android:gravity="center"
157         android:text="Izračunaj"
158         android:textColor="#fff"
159         android:textSize="14sp" />
160
161     <TextView
162         android:id="@+id/textView"
163         android:layout_width="match_parent"
164         android:layout_height="88dp"
165         android:layout_marginTop="16dp"
166         android:gravity="center" />
167
168 </LinearLayout>
```

Slika 4.13: Prikaz XML koda

Klasa Button označava gumb koji omogućuje izvršavanje određenih radnji pritiskom na njega. Za dodavanje Buttona u izgled potrebno je unutar XML datoteke koristiti oznaku <Button>. Dimenzije visine i širine elementa postavljaju se svojstvima layout_width i layout_height kako bi bile jednake veličini sadržaja unutar gumba.

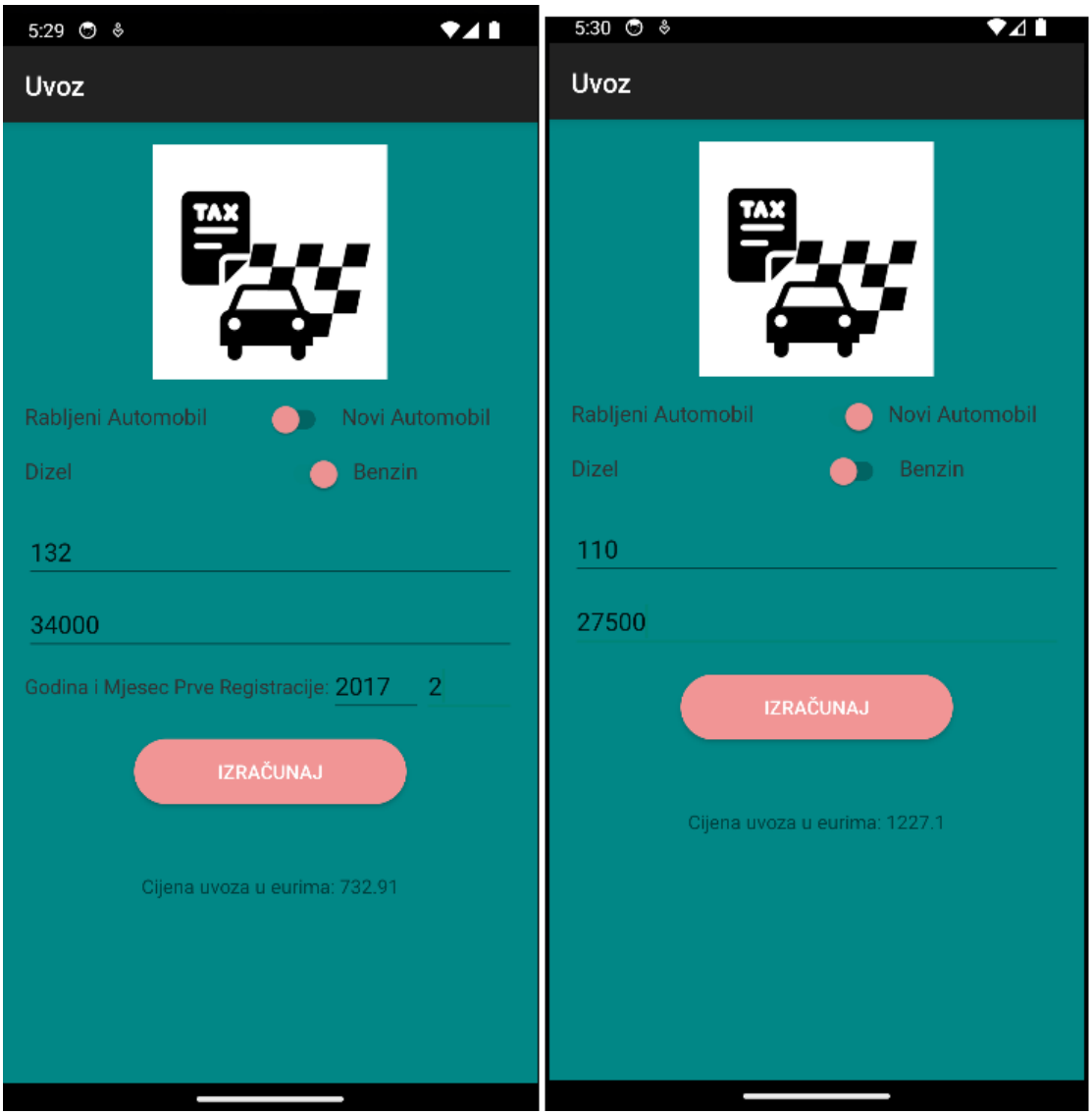
4.5. Testiranje aplikacije

U ovom poglavlju prikazan je izgled gotove aplikacije, te je testirana njena funkcionalnost. Da bi aplikacija radila, prethodno spomenuto aplikacijsko programsko sučelje mora biti postavljeno na poslužitelj kako bi aplikacija mogla komunicirati s istim. Početni zaslon sadrži ImageView, 5 TextViewa, Button, 2 Switcha i 4 EditTexta.



Slika 4.14: Prikaz početnog zaslona aplikacije

Aplikacija od korisnika traži unos podataka s pomoću 2 Switcha i 4 EditTexta. Na slici *4.14* uočavamo da kada korisnik promjeni stanje prvog Switcha, zadnji TextView i EditText postanu nevidljivi korisniku. Na ovaj način aplikacija ne zbunjuje korisnika pri unosu potrebnih podataka, jer ako je automobil novi, tada još nije bio prvi puta registriran.



Slika 4.15: Prikaz rada aplikacije

Aplikacija se smatra uspješnom ako pokaže točnu cijenu s unesenim podatcima, kao što slika *4.15* pokazuje. Korisnik s pomoću Switcheva može mijenjati stanje i tip automobila, te mijenjati cijenu

i CO_2 emisiju unosom tipkovnice. Pritiskom na Button aplikacija nam izračunava cijenu uvoza automobila u eurima, te je prikazuje na zaslonu.

5. ZAKLJUČAK

U okviru ovog završnog rada je razvijena i opisana implementacija mobilne aplikacije za Android platformu čiji je cilj izračunavanja cijene uvoza automobila. Korištenjem programskog jezika Kotlin i Python u kombinaciji s aplikacijsko programskim sučeljem postavljenim na poslužitelj, aplikacija je velika pomoć pri uvozu automobila. Korisnici moraju unijeti tražene parametre kako bi znali točnu cijenu uvoza automobila izraženu u eurima.

Daljnji rad na aplikaciji će doprinijeti njezinoj konkurentnosti na tržištu, ali će također služiti i kao koristan alat za usavršavanje znanja u izradi Android aplikacija. Tijekom testiranja su primijećeni neki nedostaci, što je uobičajeno za svaku aplikaciju. Postoji mogućnost prilagodbe funkcionalnosti i dodavanja novih mogućnosti tijekom daljnjeg razvoja projekta. Iako su prvobitno zamišljene funkcionalnosti ostvarene, neke dodatne nisu bile moguće zbog ograničenog vremena izrade završnog rada.

Glavni nedostatak aplikacije je potrebnost pristupa internetu, jer bez toga pristup aplikacijsko programskom sučelju nije moguć, kao ni slanje HTTP zahtjeva. Obavezno korištenje interneta ograničava korisnike da koriste aplikaciju samo kada su online, smanjujući predviđenu praktičnost. Još jedan nedostatak su česte izmjene zakona u našoj državi, što zahtjeva učestalo održavanje i nadogradnju sučelja i aplikacije.

Za poboljšano korisničko iskustvo, aplikaciju bi trebalo nadograditi s virtualnim popunjavanjem formulara, te online predaju istih Carinskoj upravi Republike Hrvatske, radi lakšeg kompletnog procesa uvoza automobila. Smatram da bi i izrada korisničkih računa u aplikaciji također bilo potrebno, kako bih Carinska uprava znala tko je poslao formulare za prijavu uvoza, te bi im slanje uplatnica istima bilo olakšano.

LITERATURA

[1] „Meet Android Studio | Android Developers“. https://developer.android.com/studio/intro (pristupljeno 15. prosinac 2023.).

[2] „IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains“, JetBrains. https://www.jetbrains.com/idea/ (pristupljeno 15. prosinac 2023.).

[3] „Kotlin Programming Language“, Kotlin. https://kotlinlang.org/ (pristupljeno 15. prosinac 2023.).

[4] Šimec, A., i Filipec, M. (2021). 'Kotlin compared to Java for development on JVM and Android platforms', *Polytechnic and design*, 9(3), str. 202-207 (pristupljeno 16. prosinac 2023.).

[5] „XML Introduction“. https://www.w3schools.com/xml/xml_whatis.asp (pristupljeno 16. prosinac 2023.)

[6] „Flask“https://palletsprojects.com/p/jinja/ (pristupljeno 22. prosinac 2023.).

[7] T. P. Trappenberg, „Scientific programming with Python“, u Fundamentals of Computational Neuroscience: Third Edition, T. P. Trappenberg, Ur., Oxford University Press, 2022, str. 0. doi: 10.1093/oso/9780192869364.003.0002. (pristupljeno 22. prosinac 2023.).

[8] Goldberg, Kevin (2016-05-09). "An Introduction to Python WSGI Servers for Performance | AppDynamics". *Application Performance Monitoring Blog | AppDynamics*. (pristupljeno 19. prosinac 2023.).

[9] „Jinja2“https://flask.palletsprojects.com/en/3.0.x/ (pristupljeno 18. prosinac 2023.).

[10] „Autentifikacija“http://struna.ihjj.hr/naziv/autentifikacija/51557/#naziv/ (pristupljeno 20. prosinac 2023.).

[11] Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine. (pristupljeno 17. prosinac 2023.).

[12] Trstenjak, Bruno, Željko Knok i Jurica Trstenjak. "SOAP komunikacijski protokol." *Zbornik radova Međimurskog veleučilišta u Čakovcu* 1, br. 1 (2010): 59-68. https://hrcak.srce.hr/55559 (pristupljeno 19. prosinac 2023.).

[13] Richardson, Leonard; Ruby, Sam (2007). *RESTful Web Services*. Sebastopol, California: O'Reilly Media. ISBN 978-0-596-52926-0 (pristupljeno 17. prosinac 2023.).

[14] „Requests“ https://pypi.org/project/requests/ (pristupljeno 20. prosinac 2023.).

[15] „Jsonify“https://www.fullstackpython.com/flask-json-jsonify-examples.html (pristupljeno 20. prosinac 2023.).

[16] „Datetime“https://docs.python.org/3/library/datetime.html (pristupljeno 20. prosinac 2023.).

[17] Ustav Republike Hrvatske („Narodne novine“, br. 15/2013, 108/2013, 115/2016, 127/2017 i 121/2019); Zakon o posebnom porezu na motorna vozila i članak 2. Uredbe o načinu izračuna i visinama sastavnica za izračun posebnog poreza na motorna vozila („Narodne novine“, broj 148/2020) (pristupljeno 22. prosinac 2023.).

[18] Ustav Republike Hrvatske („Narodne novine“, br. 15/2013, 108/2013, 115/2016, 127/2017 i 121/2019); Zakon o posebnom porezu na motorna vozila i članak 2. stavak 2.

Uredbe o načinu izračuna i visinama sastavnica za izračun posebnog poreza na motorna vozila („Narodne novine“, broj 148/2020) (pristupljeno 22. prosinac 2023.).

[19] Ustav Republike Hrvatske („Narodne novine“, br. 15/2013, 108/2013, 115/2016, 127/2017 i 121/2019); Zakon o posebnom porezu na motorna vozila i članak 9. stavak 3. Uredbe o načinu obračuna i plaćanje posebnog poreza („Narodne novine“, broj 148/2020) (pristupljeno 22. prosinac 2023.).

[20] „Retrofit“https://guides.codepath.com/android/consuming-apis-with-retrofit (pristupljeno 29. prosinac 2023.).

[21] „Gson“ https://github.com/google/gson/blob/main/UserGuide.md (pristupljeno 29. prosinac 2023.).

SAŽETAK

Cilj ovog rada bio je napraviti aplikaciju i razviti aplikacijsko programsko sučelje bazirano na REST protokolu za jednostavno izračunavanje cijene uvoza automobila. Aplikacija omogućava korištenje aplikacijsko programskog sučelja s pomoću spajanja na poslužitelj. U aplikaciji se korisniku omogućuje jednostavan i efikasan izračun cijene uvoza automobila unosom potrebnih parametara. Aplikacija je napravljena u Android okruženju te je dostupna svim korisnicima android uređaja. U radu su opisane osnove rada s Flaskom, koji je web framework za programski jezik Python koji omogućava jednostavno izrađivanje web aplikacija, fokusira na minimalizam i lakoću korištenja, pružajući osnovne alate za brzo razvijanje web projekata, kao i osnove izrade Android aplikacija sa slikama ispod teksta kao primjer koda.

Ključne riječi: aplikacija, aplikacijsko programsko sučelje, Flask, REST protokol, uvoz automobila

ABSTRACT

Title: Mobile application with internet interface for calculating the price of car imports

The main goal of this project was to create an application and develop an application programming interface based on the REST protocol for simple calculation of the price of car imports. The application enables the use of an application programming interface using a connection to the server. The application enables the user to easily and efficiently calculate the price of importing a car by entering the necessary parameters. The application was created in the Android environment and is available to all users of Android devices. This project describes the basics of working with Flask, which is a web framework for the Python programming language that enables easy creation of web applications, focuses on minimalism and ease of use, providing basic tools for fast development of web projects, as well as the basics of creating Android applications with images below the text as an example code.

Keywords: application, application programming interface, car import, Flask, REST protocol

ŽIVOTOPIS

Mateo Košturjak rođen je 9. lipnja 2000. godine u Našicama. Završava Osnovnu školu Dore Pejačević u Našicama, te se 2015. godine upisuje u Srednju školu Isidora Kršnjavog. Nakon završene prirodoslovno-matematičke gimnazije i položene mature upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, preddiplomski sveučilišni studij računarstva.