Uporedna analiza različitih načina integracije klijentske i serverske strane aplikacije razvijene u skript jezicima

Marija Stepanović

***Sadržaj* — U ovom radu razmatraju se različite tehnike razvoja klijentske strane veb aplikacije i njeno povezivanje sa serverskom stranom razvijenom u programskom jeziku Python korišćenjem Django frejmvorka. Osnovnim funkcionalnostima serverske strane može se pristupiti preko REST poziva. Rađeno je pet različitih tehnika razvoja klijentske strane aplikacije, izvorni Javascript, tri različita veb frejmvorka, to su JQuery, Angular i React i Django templejti. Sve tehnike razvoja klijentske strane osim Django templejta komuniciraju sa serverskom stranom preko REST poziva. Cilj je bio da se uporede različite tehnike integracije sa serverskom stranom, da se prikažu njihove prednosti i nedostaci. Analizirana je kompleksnost određenog jezika ili frejmvorka, čitljivost koda, sigurnost, podržanost od strane zajednice. Analiza će se raditi na primeru razvoja veb aplikacije za rentiranje motornih vozila.**

***Ključne reči* — JavaScript, React, Angular, Django, JQuery**

I. UVOD

DANAS je razvoj veb aplikacija jedna od najvećih grana industrije informacionih tehnologija. Skoro svakom softverskom rešenju je potrebno pristupiti preko veb pretraživača ili telefona. Poslednjih godina se dosta ulaže u razvoj veb aplikacija koje su podržane i od strane mobilnih uređaja (mobilni telefoni i tableti), time se čuvaju resursi firme jer ne postoji potreba da se razvijaju Android ili iOS aplikacije. Ovaj rad se bavi analizom različitih jezika i frejmvorka za implementaciju klijentske strane veb aplikacije.

Kao primer za demonstraciju izabrana je veb aplikacija za rentiranje

P. A. Marija Stepanović, Računarski fakultet, Srbija (telefon: 060/0262692; e-mail: mstepanovic1118m@raf.edu.rs).

motornih vozila. Implementirane su osnovne funkcionalnosti, unos novih vozila i korisnika, kao i njihovo iznajmljivanje. Svi podaci se skladište u relacionu bazu podataka.

Poslednjih godina Django frejmvork dobija sve veću popularnost, zbog brzog i lakog razvoja veb aplikacija. Stoga je odlučeno da će on biti korišćen sa pisanje serverske strane. Django podržava i razvoj klijentskog dela veb aplikacije koji je takođe razmatran u radu kao jedan način integracije sa serverskom stranom.

Kada je reč o klijentskoj strani tu postoji pregršt različitih jezika i frejmvorka, koji se uglavnom zasnivaju na Javascript-u i zato su za poređenje izabrani frejmvorci koji se na njemu zasnivaju, to su JQuery, Angular i React. JQuery je jedan od prvih Javascript frejmvorka, a Angular i React su dva najveća frejmvorka iza kojih stoje kompanije kao što su Google i Facebook.

II. Opis korišćenih tehnologija za implementaciju klijentske strane veb apliakcije

U ovom poglavlju dajemo pregled svih korišćenih tehnologija, to su JavaScript kao osnovni jezik za implementaciju klijentske strane, zatim tri frejmvorka za razvoj veb aplikacija, to su JQuery, React i Angular, na kraju je opisan Django frejmvork.

A. Javascript

Javascript je dinamički tipiziran programski jezik visokog nivoa. Standardizovan je po ECMASkript (ECMAScript) specifikaciji jezika i spada u grupu skript jezika. Mogu se pisati direktno u HTML veb stranici i bez potrebe za kompajliranjem pokrenuti u pretrazivaču tj. na veb strani. Danas se Javascript izvršava ne samo u pretraživaču, već i na serveru. Pretraživač već ima ugrađeni mehanizam koji se naziva „Javascript virtuelna mašina“ [1].

Javascript se na strani klijenta sastoji od nekih uobičajenih programskih karakteristika, koje omogućavaju da se rade stvari poput:

- validacija korisničkog unosa,
- jednostavne kalkulacije i kontrola na klijentskoj strani (unutar brauzera),
- rad sa različitim složenim tipovima podataka poput stringova, nizova, vremena i datuma,
- generisanje HTML sadržaja.

Javascript je imperativni jezik i on uglavnom izvršava kod u redosledu koji je napisan, od početka do kraja. Javaskript interpreter koristi tehniku prevođenja na licu mesta (“just-in-time”), da bi poboljšao preformanse. Javascript kod se prevodi u binarni format, da bi se mogao brže izvršavati [2].

B. JQuery

JQuery je veoma popularna biblioteka napisana u Javascript-u, koja kao primarnu ulogu ima manipulaciju strukturom HTML dokumenta (HTML DOM). Omogućava da se pronađu delovi veb strane i da se nad njima izvrši izmena sa svega par linija koda. On takođe čini osnovu drugih većih frejmvorka. Moto JQuery-ja je "piši manje, uradi više" jer omogućava lako i brzo pisanje koda, koje bi inače zahtevalo više vremena koristeći druge jezike [5].

JQuery omugućava da se krećemo kroz DOM, koristeći CSS sintaksu. Pošto je napisan u Javascript-u može da koristi sve njegove funkcionalnosti. Na JQuery mogu da se dodaju razni plagin-ovi koji proširuju njegovu mogućnost izvršavanja. JQuery koristi koncept ulančavanja metoda. JQuery obejkat ima niz DOM čvorova i funkcija koje mogu da ih modifikuju. Kada se pozove neka funkcija ona menja čvor koji je skladišten u JQuery objektu. Funkcija može da manipuliše sa više ili sa svim čvorovima istovremeno, i svaki poziv funkcije vraća JQuery objekat [3].

C. Angular

Angular je platforma i frejmvork koji se koristi za pravljenje SPA (single page application) aplikacija korišćenjem HTML i TypeScripta. Angular je napisan u Typescript-u. Angular je projekat otvorenog koda (open source) [10].

Organizovanjem koda u jasno definisane funkcionalne celine, omogućava lakši razvoj složenih aplikacija i lakšu upotrobu istog koga. Ova tehnika omogućava korišćenje lenjog učitavanja (lazy loading).

Svaka Angular aplikacija ima barem jednu komponentu, root komponetu koja povezuje sve ostale komponente u hijerarhiji sa DOM-om. Svaka komponenta definiše klasu, koja sadrži logiku i podatke i povezana je sa HTML templejtom koji definiše vju koji će biti prikazan.

Angular podržava koncept umetanja zavisnosti (Dependency injection - DI) koji obezbeđuje da komponenta dobija druge komponente (servise) koje su joj potrebne, umesto da ih sama pravi što omogućava da komponenta ostane čista i pregledna.

Angularov Router modul pruža servis koji omogućava da se definišu putanje između više stanja i view-ova unutar hijerarhije aplikacije. Router mapira URL putanje u view-ove a ne u stvarne strane. Kada korisnik izvrši neku akciju, kao na primer klik na link, to bi učitalo novu stranu u pretraživač, ali router presreće ovo ponašanje pretraživača, i prikazuju ili sakriva view [9].

D. React

React je Javascript biblioteka otvorenog koda koja obezbeđuje pregled podataka zapisanih preko HTML-a. React pregledi su obično obezbeđeni korišćenjem komponenti koje sadrže dodatne komponente definisane kao prilagođene HTML oznake. React obezbeđuje programeru model u kojem podkomponente ne mogu direktno da utiču na spoljašnje komponente, efikasno ažuriranje HTML dokumenta pri promeni podataka i jasno razdvajanje komponenti na današnjim jednostraničnim aplikacijama.

U ReactJS-u su komponente integrisane kako bi stvorile jednu veću i dinamičniju aplikaciju. Komponenta može da se promeni u bilo kom trenutku, a da to ne utiče na ostatak aplikacija. Ova funkcija je najefikasnija kada se primeni u većim aplikacijama u kojima se podaci često menjaju. Svaki put kada se dodaju ili ažuriraju podaci, ReactJS automatski ažurira određenu komponentu čije se stanje zapravo promenilo. Ovo štedi pretraživač jer nije potrebno ponovno učitavanje cele aplikacije kako bi se videle promene [11].

React se koristi za razvoj korisničkog interfejsa. React se sastoji iz JSX i virtuelnog DOM-a. JSX (Javascript Extension) je React-ov dodatak koji omogućava veb programerima da lako modifikuju DOM, koristeći kod koji liči na html. JSX koristimo za ažuriranje DOM-a, i pruža nam značajno poboljšanje performansi i efikasnosti. Da bi se poboljšale performanse i efikasnost koristi se virtuelno DOM. Virtualni DOM je kopija pravog HTML DOM-a, i React koristi ovu kopiju da odredi koje delove pravog DOM-a treba zameniti prilikom izvršavanje nekih akcija (kao na primer kliktanje na dugme). Ovaj tip selektivnog updata zahteva manje resursa i skraćuje se vreme učitavanja. Ovim se poboljšava efikasnost sajta i ubrzava učitavanje celog sajta [4].

III. REST API

REST API je standardizovani način za pružanje podataka drugim klijentskim aplikacijama koji se zasniva na HTTP protokolu. Ponekad API-ji nude i način da druge aplikacije izvrše promene u podacima.

API je interfejs aplikacije preko koga druge aplikacije ili klijenti mogu da komuniciraju sa aplikacijom. API serverske strane obrađuje HTTP zahteve klijenta i kreiranje odgovora, pri čemu se u tom procesu često vrše operacije nad bazom podataka na serverskoj strani. Ono što se dobije sa serverske strane je resurs, obično u JSON formatu.

Proces ispitivanja i pretvaranja tabelarnih vrednosti baze podataka u JSON ili drugi format naziva se serializacija. Kada se kreira API, glavni izazov je

ispravna serializacija podataka.

IV. DJANGO

Django je besplatan veb frejmvork otvorenog koda, napisan u Pajtonu (eng. Python), koji prati model-pogled-kontroler (MVC) arhitekturu. Glavni cilj Djanga je da olakša i ubrza razvoj kompleksnih veb apliakcija koje po pravilu mogu da koriste bilo koju bazu podataka [12].

Osnovni elementi Django frejmvorka su:

- Django modeli - mehanizam za mapiranje objekata na relacije, koriste se za perzistenciju i preuzimanje podataka iz baze,
- Django templejt - mehanizam za kreiranje HTML stranica klijentske apliakcije,
- Django views - predstavlja kontroler, koji prima zahteve od klijentske strane i izvršava traženu funkcionalnost,
- Django urls - koristi se za mapiranje url putanja sa view-ovima.

Django koristi objekte HTTP zahteva i odgovora (request i response) za komunikaciju između klijenta i servera. HTTP objekti u Django-u mogu se prenositi upotrebom REST API poziva.

Django ima prikladan način za dinamičko generisanje HTML-a korišćenjem templejta. Templejt sadrži statičke delove željenog HTML koda, a posebnom sintaksom moguće je dinamički umetnuti sadržaj iz aplikacije.

V. BEZBEDNOST PRISTUPA VEB APLIKACIJAMA

JSON Web Token ili JWT je internet standard koji se koristi za kompaktan i kompletno bezbedan prenos informacija. Token je potpisan privatnim ili javnim ključem. JWT tokeni mogu biti i enkriptovani da obezbede tajnost u komunikaciji, ili da budu potpisani. Potpisani tokeni koriste se za verifikaciju integriteta zahteva, dok se enkriptovani tokeni koriste za sakrivanje zahteva od drugih strana. Tokeni se prave da budu kompaktni i da se mogu korisiti u SSO kontekstu [7].

JWT se sastoji iz tri dela razvojenih tačkom (.):

- Header
- Sadržaj (Payload)
- Potpis

Header obično sadrži dva dela: vrstu/tip tokena, u ovom slučaju to je JWT i algoritam koji se koristi za potpisivanje, na primer SHA256.

Payload se potom enkoduje, da bi napravio drugi deo JWT-a. Potpisani token, sprečava izmenu podataka, ali podaci u njemu su potpuno čitljivi. Ne

treba stavljati tajne podatke u header ili payload, sem u slučaju ako su enkriptovani.

Potpis se koristi da verifikuje da poruka nije izmenjena, a u slučaju da je token potpisan privanim ključem može se verifikovati i da je pošiljalac onaj za koga se i predstavlja [8].

VI. Implementacija serverske strane aplikacije

U ovom projektu izabrana je veb aplikacija za rentiranje motornih vozila kao primer na kome će se pokazati razlike između različitih implementacija klijentske strane

Na serverskoj strani aplikacije renta-kar koristi se relaciona baza podataka u kojoj se skladište podaci o korisnicima aplikacije, o automobilima i rentiranjima.

Za skladištenje podataka Django koristi tehniku mapiranja objekata na relacione tabele (ORM) preko koncepta modela. Model opisuje jedan entitet koji se skladišti u bazi, u njemu se navode atributi entiteta i veze sa drugim entitetima. Svaki model se mapira na jednu tabelu u bazi.

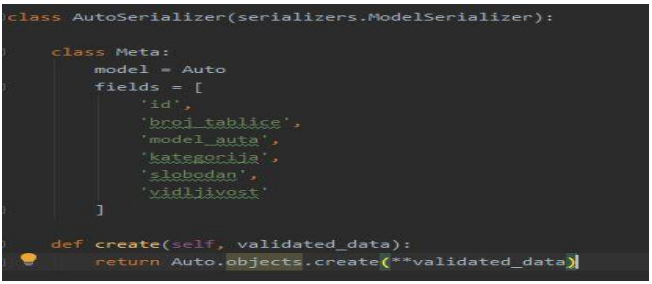
```
class Auto (models.Model):

    id = models.AutoField(primary_key='True')
    broj_tablice = models.CharField(max_length=25, unique='True')
    model_auta = models.CharField(max_length=25)
    kategorija = models.CharField(max_length=2)
    slobodan = models.BooleanField(default=1)
    vidljivost = models.BooleanField(default=1)

    class Meta:

        db_table = "polls_auto_novi"
```

Sl 1 Primer modela

Na Sl 1 prikazan je model klase Auto. Klasa Auto nasleđuje Django klasu models. Model, što označava da je ova klasa model klasa, odnosno da će se mapirati na tabelu u bazi. Zatim su navedene sve kolone iz tabele kao atributi. Na primer, atribut slobodan predstavlja istoimenu kolonu iz tabele koja ima podrazumevanu vrednost 1, što znači tačno. Metapodaci u modelu prosleđuju se korišćenjem unutrašnje Meta klase koja u datom primeru na Sl 1 sadrži ime tabele kao vrednost parametra db_table.

A. Serijalizacija Django modela

Da bi se obezbedio prenos podataka sa serverske na klijentsku stranu preko REST-a i obrnuto potrebno je da se Django model klase serijalizuju u JSON što se implementira korišćenjem Django klasa sa serijalizaciju (klase koje nasleđuju klasu serializers.ModelSerializers).

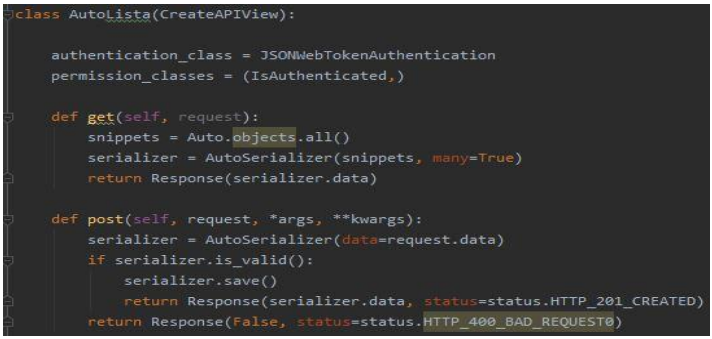


Sl 2 Primer Serializera

Na Sl 2 prikazana je serializer klasa AutoSerializer. Klasa AutoSerializer nasleđuje serializer.ModelSerializer da bi omogućila serializaciju modela. Ova klasa se sastoji iz unutrašnje Meta klase i metode create. Meta klasa sadrzi dva atributa model i fields. Atribut model nam sadrži model koji se serializuje. Atribut fields nam sadrži atribute modela koji serializujemo, ne moramo sve atribute navesti. Metoda create se poziva prilikom upisa u bazu, kreira objekat i vraća njegovu instancu. Ova metoda se poziva prilikom serializer.save().

B. Implementacija obrade zahteva na serverskoj strani

Za čitanje HTTP zahteva, njihovu obradu i slanje HTTP odgovora koristi se Django view. View koristi serializer klasu za kreiranje odgovora i čitanja zahteva.



Sl 3 Primer View-a

Na Sl 3 prikazana je view klasa AutoLista. Klasa AutoLista nasleđuje CreateAPIView, da bi mogli da koristimo REST API frejmvork. Ova klasa se sastoji iz dva dela. Prvi deo je vezan za autentifikacija. U našem primeru koršćen je JWT token, i u koliko HTTP zahtev ne sadrži validan token neće biti u mogućnosti da izvrsi REST poziv. Drugi deo je vezan za tipove rest zahteva. Postoje dva tipa zahteva, i to su: post i get. Post se koriste da se podaci pošalju serveru za kreiranje ili ažuriranje resursa. Ovi podaci se nalaze u telu HTTP zahteva. Get se koristi da zatražimo neke podatke od servera. Parametri koji se šalju nalaze se u url putanji.

U našem primeru imamo post i get metodu. Get metoda sadrži dva artibuta, prvi atribut snippets dobijen odgovor od modela, u ovom slučaju model vraća listu svih auta. Drugi atribut serializer serializuje dobijeni odgovor. Na samom kraju vraćamo dobijeni odgovor u JSON formatu. Post metoda sadrži atribut serializer koji deserializuje poslati zahtev. Proverava se njegova validnost, i u koliko je validnost dobra poziva metodu save() koja služi za snimanja objekta u bazu, i vraća taj objekat u HTTP odgovoru. U koliko nije validno kao odgovor vraća grešku HTTP_400_BAD_REQUEST.

C. Rutiranje HTTP zahteva na serverskoj strani

Django koristi url putanje za mapiranje HTTP zahteva na određene view-ove ili druge url-ove.

```
urlpatterns = [
    path('polls/', include('polls.urls')),
    path('admin/', admin.site.urls),
    path('apiAdmin/', include('app_master_admins.urls')),
    path('apiAuto/', include('app_master_auto.urls')),
    path('apiKlijenti/', include('app_master_klijenti.urls')),
    path('apiIzvestaj/', include('app_master_izvestaj.urls'))
]
```

Sl 4 Primer glavnog url

Url putanje u projektu su implementirane kao hjerarhijski model, koji se sastoji od jednog roditelja i više dece. Na Sl 4 je prikazan jedan roditelj koji predstavlja skup glavnih url putanja za projekat. Svaka putanja ima dva parametra, od kojih je prvi parametar početni deo url putanje HTTP zahteva,

```
urlpatterns = [
    path('autoall', AutoLista.as_view()),
    path('auto', AutoActiveList.as_view()),
    path('autoSvi', AutoActiveListSvi.as_view()),
    path('autoZauzeti', AutoActiveListZauzeti.as_view()),
    path('autoId', AutoDetail.as_view()),
    path('autoNovoBrisanje', AutoNovoBrisanje.as_view()),
    path('vrati', AutoVrati.as_view()),
    path('izvestajAuto', AutoBrojTablice.as_view()),
    path('rentiranje', AutoRentiranje.as_view())
]
```

a drugi parametar predstavlja dete tj. url Django modul koji treba da se pozove. Ovaj pristup omogućava bolju preglednost, održavanje i bolju kontrolu pristupa resursima podataka.

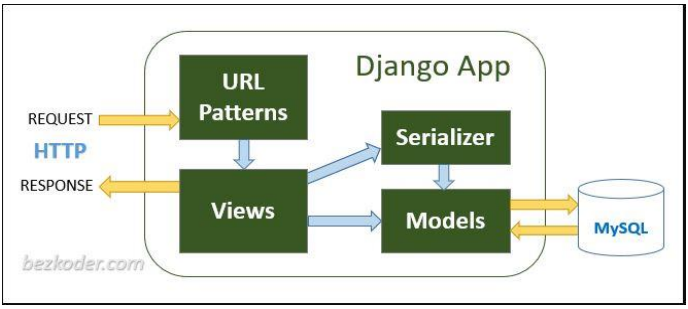
Sl 5 Primer url putanja za rad sa entitetom automobila

Na Sl 5 prikazana je url putanje za jedno dete tj. za jedan Django modul, i svaka putanja ima dva parametra. Prvi parametar je drugi deo url putanje HTTP zahteva, drugi parametar je mapirani view koji odgovora tom HTTP zahtevu. Sl 5 možemo da zamislimo kao rečnik, u kome ključ prestavlja url putanja, dok je vrednost tog ključa određena klasa u view-u.

D. Django REST arhitektura aplikacije renta-kar

Dijagram na Sl 6 prikazuje osnovnu arhitekturu Django-a REST API-ja, ona se sastoji iz 4 osnovna dela koji su prethodno opisana, to su:

1. url - koristi se za mapiranja HTTP zahteva za odgovarajući view,
2. views - koristi se za obradu HTTP zahteva i slanje odgovora,
3. serializer - koristi se za serializaciju i deserializaciju objekta modela
4. models - koristi se za komunikaciju sa bazom.



Sl 6 Arhitektura Django REST API [13]

E. Podrška za Django templejte na serveru

Korišćenje Django templejta zahteva da se drugačije implementira view funkcija u odnosu na REST, jer se umesto JSON-a vraćaju html stranice koje pored samog HTML-a sadrže i podatke iz aplikacije. U okviru HTML za kreiranje templejta koristimo Django templejt jezik.

```
def get(self, request, *args, **kwargs):

    form = AuthenticationForm()
    return render(request, 'templates/login.html', {"form": form})
```

Sl 7 View klasa templejta

Na Sl 7 prikazan je view klasa za templejt. U ovoj klasi imamo atribut form. Ovaj atribut se koristi za kreiranje autorizacione forme. Za razliku od REST-a gde bi vratili HTTP odgovor, u templejtu vraćamo HTML stranicu, i atribut koji umećemo u nju.

F. Bezbednost pristupa na serverskoj strani

Za bezbednost komunikacije koristimo JWT token, koji nam omogućava kontrolu pristupa serverskoj strani.

```
try:
    payload = JWT_PAYLOAD_HANDLER(user)
    jwt_token = JWT_ENCODE_HANDLER(payload)
    update_last_login(None, user)
except Admins.DoesNotExist:
    raise serializers.ValidationError(
        'User with given email and password does not exists'
    )
return {
    'email': user.email,
    'token': jwt_token
}
```

Sl 8 Kreiranje JWT tokena

Na Sl 8 je prikazano kreiranje JWT tokena. Kreiranje tokena se vrši iz dva dela. U prvom delu pravimo telo tokena u kome se nalaze inicijali uspešno ulogovanog korisnika. U drugom delu kreiramo ceo token, kome dodajemo i telo koje smo već kreirali u prvom delu. U okviru kreiranja JSON odgovora, dodajemo i JWT token u parametar token.

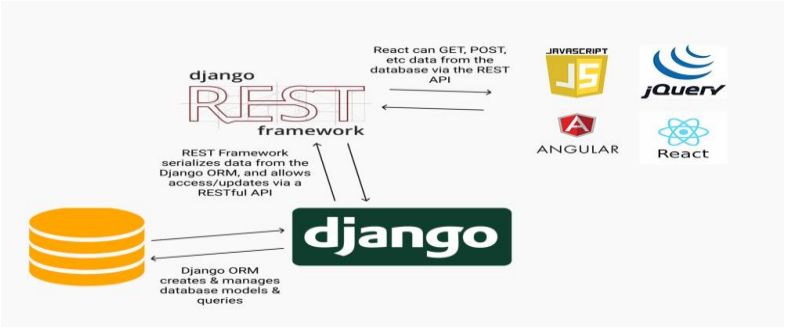
```
class AutoLista(CreateAPIView):

    authentication_class = JSONWebTokenAuthentication
    permission_classes = (IsAuthenticated,)
```

Sl 9 Bezbednost pristupa

Na Sl 9 prikazana je provera samog tokena, koja se vrsi prilikom svakog REST poziva. Ova provera se nalazi u svim klasama u view-ovima. Proveravamo da li postoji JWT token, i ukoliko postoji to znači da je korisnik autentifikovan i dajemo mu pristup našem resursima. Ako neko pokušava da koristi url putanju, a nije ulogovan, neće dobiti nikakve podatke od našeg servera.

VII. IMPLEMENTACIJA KLIJENTSKE STRANE I KOMUNIKACIJA SA SERVEROM

Glavni cilj ovog rada je da se testiraju različiti načini implementacije poziva serverske strane aplikacije od strane klijentske aplikacije. Četiri tehnologije zasnovane na JavaScript-u pozivaju serversku stranu putem REST-a prema arhitekturi koja je prikazana na Sl 10, dok peto rešenje koristi direktnu integraciju sa serverom preko templejta koji se zasniva na istoj tehnologiji kao i serverska strana aplikacije.



Sl 10 Arhitektura projekta [6]

A. *Klijent implementiran u Javascript-u*

Za REST pozive u javascript-u se koristi XMLHTTP funkcija. XMLHTTP funkcija nam omogućava da radimo slanje zahteva preko post i get-a, da stavimo parametre u hederu i možemo da šaljemo i JSON objekte.

```
function dodajKlijenta(){
    var ime = document.getElementById("ime").value;
    var prezime = document.getElementById("prezime").value;
    var jmbg = document.getElementById("jmbg").value;
    var kategorija = document.getElementById("kategorijaK1").value;
    var requestKl = new XMLHttpRequest();

    if (kategorija != "A" && kategorija != "B" && kategorija != "C" && kategorija != "D") {
        window.location.reload();
    }

    requestKl.open('POST','http://127.0.0.1:8000/apiKlijenti/klijentiall');
    requestKl.setRequestHeader('Content-Type', 'application/json');
    requestK1.setRequestHeader( 'Authorization', 'Bearer '+token);
    requestKl.onload = function(){
        window.location.reload();
    };
    requestKl.send(JSON.stringify({
        ime: ime,
        prezime : prezime,
        jmbg : jmbg,
        kategorija : kategorija,
    }));
};
```

Sl 11 Slanje POST zahteva iz Javascript

Na Sl 11 prikazan je primer slanja POST zahteva serveru. Imamo četiri

promenljive: ime, prezime, jmbg, kategorija. Ove promenljive služe za kreiranje JSON objekta koji stoji u telu POST zahteva. Promenljiva requestK1 nam služi za slanje HTTP upita i za primanje odgovora. Prvo pozivamo metodu open i šaljemo joj dva parametra. Prvi parametar predstavlja tip HTTP upita. Drugi parametar predstavlja url putanju. Sledeća metoda koju pozivamo je setRequestHeader, i ona prima dva parametra. Prvi parametar je koje HTTP zaglavlje hoćemo da podesimo, a drugi parametar je njegova vrednost. U našem slučaju sadržaj poziva je JSON, i prosleđujemo autorizacioni token. Metoda onload mora da stoji ispred metode send, da bi znala šta da uradi kada se izvrši metoda send. U našem slučaju ona služi za ponovno učitavanje stranice. Metoda send služi da se izvrsi REST poziv.

```
function opcijaFunction() {
    var opcija = document.getElementById("opcije").value;
    if (opcija == 'slobodni'){
        putanja = 'http://127.0.0.1:8000/apiAuto/auto';
    } else if (opcija == 'zauzeti') {
        putanja = 'http://127.0.0.1:8000/apiAuto/autoZauzeti';
    } else if (opcija == 'svi') {
        putanja = 'http://127.0.0.1:8000/apiAuto/autoSvi';
    }

    // console.log(opcija);
    const node = document.getElementById("listaFilter");
    node.innerHTML = '';

    request1.open('GET', putanja, true);
    request1.setRequestHeader( 'Authorization', 'Bearer '+token);
    console.log(opcija);
    request1.onload = function () {

        var data = JSON.parse(this.response)
        console.log(data);

        data.forEach(tmp => {
            var node = document.createElement("LI");
            var textnode = document.createTextNode(tmp.broj_tablice + " "+tmp.model_auta+ " "+tmp.kategorija);
            node.append(textnode);
            document.getElementById("listaFilter").appendChild(node);
        })
    };
    request1.send();
}
```

Sl 12 Slanje i primanje GET zahteva iz Javascript

Na slici 12 prikazan je primer slanja GET zahteva serveru. Slanje GET zahteva se radi slično kao slanje POST zahteva. Jedina razlika je u tome što se ne šalje JSON objekat. Metoda send nam je prazna.

B. Klijent implementiran u JQuery-u

Za REST pozive u jquery-ju se koristi AJAX biblioteka. Ona nam omogućava da u pozadini učitavamo nove podatke i prikazujemo na veb strani.

```
$("#vracanje").click(function () {
    var vrati_broj_tablice = $("#vrati_broj_tablice").val();

    $.post("http://127.0.0.1:8000/apiAuto/vrati",{
        vrati_broj_tablice: vrati_broj_tablice,
    },
    function(data,status){
        window.location.reload();
    });

});
```

Sl 13 Slanje POST zahteva iz JQuery

Na Sl 13 prikazan je primer slanja POST zahteva serveru. Imamo jednu promenljivu koja nam služi za kreiranja tela POST zahteva u JSON formatu. Post zahtev se vrši pozivanjem $.post funkcije i prosleđuju se tri parametra. Prvi parametar je url koji hoćemo da gađamo. Drugi parametar je JSON objekat, u našem slučaju koristimo promenljivu koju smo već pre toga kreirali. Treći parametar je funkcija koja treba da se izvrši posle pozivanja post metode. Funkcija sadrži dva parametra, prvi je odgovor koji smo dobili od servera, a drugi je kod za status. Mi ovde radimo ponovno učitavanje strane.

```
var listaA = '';


$.get("http://127.0.0.1:8000/apiAuto/auto"
, function (data, status) {
    //console.log(data);
    listaA = data;
    //console.log(lista);
    $("#boundOnPageLoaded").remove();
    $.each(listaA, function (key, value) {
        $("#prikaz").append("<li>"+value.broj_tablice+" "+value.model_auta+
        +" "+value.kategorija+"<input type='button' id='boundOnPageLoaded' value='click' name='"+value.id+"' /></li>");
        //console.log(value.username);
    })
});
```

Sl 14 Slanje i primanje GET zahteva iz JQuery

Na Sl 14 prikazan je primer slanja GET zahteva serveru. Slanje GET zahteva se radi slično kako slanje post zahteva. Jedina razlika je što get metoda prima dva parametra. Prvi parametar je url, a drugi parametar je funkcija.

C. Klijent implementiran u Angular-u

Za REST poziv u angular-u koristimo HttpClient servis. On se koristi za slanje HTTP upita i obradu HTTP odgovora.

```
vracanje(stiglo){
  this.http.post('http://127.0.0.1:8000/apiAuto/vrati',{
    vrati_broj_tablice: stiglo.broj_tablice,
  },
  {
    headers: new HttpHeaders(
    {
      'Authorization': 'Bearer '+ this.token
    }
    )
   }).subscribe(data =>{
    console.log(data);
});
this.onChange("svi");
}
```

Sl 15 Slanje POST zahteva iz Angular

Na Sl 15 prikazan primer slanja POST zahteva serveru. Prima tri parametra. Prvi parametar je url, a drugi parametar je telo post zahteva koji sadži JSON. Treći parametar je heder koji šaljemo, u našem slućaju je to autorizacioni token. Posle izvršene post metode poziva se subscribe metoda, u kojoj dobijamo odgovor od servera. U našem slučaju ispisujemo u konzoli.

```
putanja = '';
listaA: Object;
onChange(opcija){
  console.log(opcija);
  if(opcija == "svi"){
    this.putanja = 'http://127.0.0.1:8000/apiAuto/autoSvi';
  }else if(opcija == "slobodni"){
    this.putanja = 'http://127.0.0.1:8000/apiAuto/auto';
  }else {
    this.putanja = 'http://127.0.0.1:8000/apiAuto/autoZauzeti';
  }

  this.http.get(this.putanja,
    {
      headers: new HttpHeaders(
      {
        'Authorization': 'Bearer '+ this.token
      }
      )
     }).subscribe(data =>{
      this.listaA = data;
  });
}
```

Sl 16 Slanje i primanje GET zahteva iz Angular

Na Sl 16 prikazan je primer slanja GET zahteva serveru. Slanje GET zahteva se radi slično kako slanje post zahteva. Jedina razlika je što get metoda ima dva parametra. Prvi parametar je url putanja, a drugi parametar je heder. U našem slućaju je to autorizacioni token koji šaljemo. Posle izvršavalja get metode poziva se metoda subsribe. U njoj se radi obrada podataka koji smo dobili od servera. U našem primeru odgovor od servera pamtimo u promenljivoj koja je tipa liste.

D. Klijent implementiran u React-u

Za REST poziv u react-u koristimo Fetch API. On se koristi za slanje

HTTP upita i obradu HTTP odgovora.

```
kliknuto(id) {
    fetch('http://127.0.0.1:8000/apiAuto/autoId', {
        method: 'PUT',
        headers: {
            'Content-Type': 'application/json',
            'Authorization': 'Bearer '+ this.token,
        },
        body: JSON.stringify({
            id: id,
            vidljivost: 0,
        })
    });
    //this.loaddata();
    window.location.reload();

}
```

Sl 17 Slanje POST zahteva iz React

Na Sl 17 prikazan primer slanja POST zahteva serveru. Post metoda nam se sastoji iz dva parametra. Prvi parametar je url putanja. Dok je drugi parametar objekat, koji se sastoji iz tri dela. Prvi deo je tip HTTP zahteva. U našem primeru je to PUT. Drugi deo čine parametri zaglavlja. U našem primeru to je objekat u JSON formatu i šaljemo autorizacioni token. Treći deo je telo HTTP zahteva. U našem primeru to je JSON objekat koji pravimo. Posle izvršavanja PUT zahteva, izvršava se ponovno učitavanje stranice.

```
loaddata() {
    fetch('http://127.0.0.1:8000/apiAuto/auto', {
        method: 'GET',
        headers: {
            'Authorization': 'Bearer '+ this.token,
        }
    })
        .then((responce) => responce.json())
        .then((responseJson) => {
            //console.log(responseJson);
            this.setState({data: responseJson })
            //console.log(this.state.rezultal);
        });
}
```

Sl 18 Slanje i primanje GET zahteva iz React

Na Sl 18 prikazan je primer slanja GET zahteva serveru. Slanje GET zahteva se radi slično kako slanje post zahteva. Jedina razika u get metodi je što se objekat koji prosleđujemo u drugom parametru sastoji se iz dva dela. Prvi deo je tip HTTP zahteva. U našem primeru je to get. A drugi deo je heder. U našem primeru je to autorizacioni token. Posle izvršavanje get upita, poziva se then metoda. U kojoj se piše funkcionalsnost koju hoćemo da izvršimo na stranici.

E. Klijent implementiran u Django templejt-u

Templejt koristi Django templejt jezik, koji je podrazumevani u Djangu za razvoj klijentske strane. Templejt predstavlja most između html, css i javascript. Templejt se u pozadini kompajlira na html-u.

```
<body>
    <form action="/apiAdmin/signin" method="post">
        {% csrf_token %}
        {{form}}
        <input type="submit" value="LogIn">
    </form>
</body>
```

Sl 19 Poziv prema View

Na Sl 19 je prikazan poziv prema view-a iz HTML forme. U formi se nalaze dva atributa, prvi atribut action sadrži url, a drugi parametar method tip HTTP zahteva. Zatim imamo prikaz vrednosti poslate sa servera koji se skladišti u promenljivu {{form}}. Sintaksa za promenljivu u Django-u templejt jeziku je unutar dve vitičaste zagrade (“{{promenljiva}}”). Sintaksa za tag u Django-u templejt jeziku je unutar vitičaste zagrade i unutar dva procenta (“{%tag%}”). U okviru taga može da se nalazi for petlja, if i drugi tagovi karakteristični za Django templejt.

VIII. Poređenje različitih pristupa implementacije klijentske strane veb aplikacije

U ovoj sekciji analiziraćemo pet opisanih načina implementacije klijentske strane i njihove integracije sa serverskom stranom. Rešenja zasnovana na JavaScript jeziku komuniciraju sa serverskom stranom preko REST poziva, dok rešenje koje koristi Django templejte ne koristi REST. U nastavku dajemo neke prednosti i nedostatke opisanih rešenja.

Prednost rešenja zasnovanih na JavaScriptu i REST-u u odnosu na tempjelt je taj što su nezavisni od tehnologije u kojoj je razvijena serverska strana. Promenom serverske strane kod ovakvih rešenja nije potrebno menjati kod na klijentskoj strani, sve dok REST API ostaje nepromenjen. U slučaju templejta, imamo tehnologiju koja se zasniva na tehnologiji kojom je implementirana serverska strana aplikacije i promena serverske strane zahteva značajne promene na klijentskoj strani. Razlog za ovo je što je templejt u Django-u jako uvezan za serverskom stranom aplikacije i promena

tehnologije zahteva izmene na klijentskoj strani, u većini slučajeva i ponovno pisanje klijentske strane. JQuery, Angular, React se kompajliraju u Javascript i mogu da rade na bilo kom serveru, dok Django templejt mora da se izvršava na Django serveru.

Za kreiranje REST poziva svaki skript jezik ima svoj primaran način kreiranja. Javaskript koristi XMLHTTP funkciju, dok JQuery koristi AJAX biblioteku, a Angular koristi HttpClient servis i React koristi Fetch API.

Za kreiranje tipa poziva skript jezici se međusobno razlikuju u sintaksi. Jednostavnost pisanja sintakse se najbolje ogleda kod Angular-a i JQuery-ja, dok je kod Javaskript-a i React-a sintaksa malo složenija. Kada bi poredili Angular i JQuery, sa aspekta lakoće savladavanja koji je bolji i lakši, blagu prednost bi imao JQuery, zato što je kriva učenja manja. Kriva učenja je manja kod JQuery-ja zato što je sintaksa vrlo slična JavaScript-u, dok Angular ima potpuno novu sintaksu.

Kreiranje JSON-a u skript jezicima je svuda ista. Jedina razlika je u sintaksi, kako se JSON upisuje. U Angular-u i JQuery-ju je jednostavnije pakovanja podataka u JSON format. Samo parsiranje kod ovih biblioteka radi isto, tako što JSON podatak parsira u Javascript objekat. Svaka biblioteka koristi svoju metodu za čitanje, ali princip funkcionisanja metoda je isti.

React-om i JQuery-jem možemo da pravimo samo dizajn i može se nazvati da su to biblioteke a ne frejmvork. Angular je pravi frejmvork. Zato što podržava kreiranje interfejsa, klasa, zatim implementaciju, nasleđivanje i DI (dependency injection). JavaSkript je čist jezik i on nije ni frejmvork i biblioteka.

Angular ima dvosmernu vezu, ako se korisnički interfejs promeni onda se automatski menja i stanje modela. React i JQuery imaju jednosmernu vezu, da izemena modela može da utiče na izmenu korisničkog interfejsa, ali ne i obrnuto.

LITERATURA

[1] (An introduction to JavaScript Available: https://javascript.info/intro)
[2] (What is JavaScript? -Learn web development | MDN Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScrip)
[3] (JQuery Book Available: http://nicholasjohnson.com/jquery-book/)
[4] (ReactJS--What is it? What is Used For? Why Should You Learn it Available: https://skillcrush.com/blog/what-is-react-js/)
[5] (What is JQuery: An Intro for Beginners | Course Report Available https://www.coursereport.com/blog/what-is-jquery)

[6] (Build a REST API in 30 minutes with Django REST Framework | by… Available: https://medium.com/swlh/build-your-first-rest-api-with-django-rest-framework-e394e39a482c)
[7] (JSON Web Token Introduction - jwt.io Available: https://jwt.io/introduction/)
[8] (So what the heck is JWT or JSON Web Token? | by Uday Hiwarale |... Available: https://medium.com/jspoint/so-what-the-heck-is-jwt-or-json-web-token-dca8bcb719a6)
[9] (Angular - Introduction to Angular concepts Available: https://angular.io/guide/architecture)
[10] (AngularJS and Angular 2+: a Detailed Comparison - SitePoint Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/)
[11] (What And Why React.js Available: https://www.c-sharpcorner.com/article/what-and-why-reactjs/)
[12] (Django's Structure - A Heretic's Eye View - Python Available: https://djangobook.com/mdj2-django-structure/)
[13] (] Django: POST, PUT, GET, DELETE requests example | Rest Apis - … Available: https://bezkoder.com/django-rest-api/)

ABSTRACT

This paper discusses various techniques for developing the client-side web application and connecting it to the server-side developed in the Python programming language using the Django framework. The basic functionalities of the server-side can be accessed via REST calls. Five different client-side application development techniques were implement, the original Javascript, three different web frameworks, JQuery, Angular, React, and Django templates. All client-side development techniques except the Django template communicate with the server-side via a REST call. The aim was to compare different integration techniques with the server-side, to show their advantages and disadvantages. The complexity of a particular language or framework, code readability, security, and community support were analyzed. The analysis will be done on the example of the development of a web application for renting motor vehicles.

Comparative analysis of different ways of integrating client and server side applications developed in scripting languages

Marija Stepanovic