IZRADA WEB APLIKACIJE ZA DOHVAT AKTUALNIH PARAMETARA MODELA MREŽE ELEKTROENERGETSKOG SUSTAVA HRVATSKE

Šodan, Vice

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

VICE ŠODAN

Z A V R Š N I R A D

IZRADA WEB APLIKACIJE ZA DOHVAT AKTUALNIH PARAMETARA MODELA MREŽE ELEKTROENERGETSKOG SUSTAVA HRVATSKE

Split, rujan 2021.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Specijalistički diplomski stručni studij Informacijske tehnologije

Predmet: Sustavi za upravljanje resursima

Z A V R Š N I R A D

Kandidat: Vice Šodan

Naslov rada: Izrada web aplikacije za dohvat aktualnih parametara modela mreže elektroenergetskog sustava Hrvatske

Mentor: dr. sc. Igor Nazor

Split, rujan 2021

Sadržaj

Sažetak

U ovome radu predstavljena je izrada potpune (engl. *full-stack*) web aplikacije za dohvaćanje aktualnih parametara modela mreže EES-a (elektroenergetskog sustava) Hrvatske i izvoza željenih podataka. U radu je opisana arhitektura aplikacije, korištene baze podataka koje sadrže parametre modela mreže EES-a, hijerarhija prikaza te je pobliže objašnjeno koju funkciju svaka od komponenti aplikacije obavlja. Rad same aplikacije objašnjen je na prikazanom primjeru korištenja. Tehnologije korištene za izradu su Python biblioteka Django za *back-end* dio aplikacije, JavaScript biblioteka React za *front-end*, Docker kao spremnik za pokretanje aplikacije u drugim okruženjima i Active Directory za dodjelu prava pristupa korisnicima, a njihove prednosti i osobine predstavljene su u samome radu.

Ključne riječi: *Django, React, Docker, Active Directory, aplikacija*

Summary

Development of a web application for retrieving the current parameters of the power system network model

This paper presents the development of a full-stack web application for retrieving the current parameters of the Croatian power system network model and exporting the desired data. The paper describes the architecture of the application, the databases used that contain the parameters of the power system network model, the display hierarchy, and it is explained in more detail which function each of the components of the application performs. The operation of the application is explained on the shown example. Technology used as development tool for the back-end part of the application is Python library Django. JavaScript library React is used for building the front-end part. Docker is used as a container to make the application work in various environments and operating systems. Active Directory is used to grant access rights to users. Their advantages and features are presented in the paper itself.

Key words: *Django, React, Docker, Active Directory, application*

1. Uvod

Internet je preuzeo i odredio novu generaciju medija. Uveo nas je u novo moderno doba tehnologije i njegov se utjecaj odrazio na sve aspekte našeg života. Informacije su dostupnije i mnogobrojnije, te se do njih dolazi lakše i brže nego ikada prije. Razmjena informacija među računalima uz brzi razvoj tehnologije promijenila je način i odvijanje poslovanja. Poslovni procesi se odvijaju brže te se lakše dolazi do rješenja.

Informatizacija poslovnih procesa, digitalna transformacija i implementacija informacijskih sustava postali su esencijalni za uspješno poslovanje. Informatizacijom poslovnih procesa tvrtke lakše stječu prednost pred konkurentima, ali samo ukoliko se tom procesu pristupi sa jasno određenom vizijom, definiranim ciljevima, te adekvatnim planom i stručnim timom. Implementacijom informacijskih tehnologija poslovni procesi mogu se ubrzati i napraviti lakšim za korištenje.

Cilj ovoga rada je pokušati napraviti web-aplikaciju koja će dohvaćati aktualne elemente EES-a Hrvatske iz postojećih baza podataka i omogućiti izvoz željenih podataka u tablični format te na taj način prikazati primjer informatizacije određenog poslovnog procesa. Rad je podijeljen u sedam cjelina. Drugo i treće poglavlje iznose karakteristike korištenih tehnologija za izradu aplikacije i principe njihovog rada. Četvrto i peto poglavlje opisuju tehnologije koje su implementirane u rad aplikacije, a služe kao spremnik za korištenje na drugim operacijskim sustavima i za dodjelu prava pristupa korisnicima. Šesto poglavlje opisuje izradu aplikacije i pokazuje primjer njenog korištenja, njene sastavnice i pravila, te opisuje praktični rad kroz njegove komponente i funkcionalnosti koje one obavljaju. Zaključak rada donesen je u završnom poglavlju.

2. Django

Poslužiteljski dio aplikacije (engl. back-end) izrađen je korištenjem Djanga – besplatne web biblioteke otvorenog koda (engl. open source) temeljene na programskom jeziku Pythonu koji slijedi arhitektonski obrazac MTV (Model–Template–View), a potiče brzi razvoj i čist, pragmatičan dizajn [1]. Izgrađen je od strane iskusnih programera koji su se posvetili rješavanju velikog broja problema koji se javljaju pri razvoju weba, stoga se lako usredotočiti na izradu aplikacije bez potrebe za ponovnim „izmišljanjem kotača“. Django je osmišljen kako bi programerima pomogao u što bržoj izradi aplikacija od koncepta do završetka. Sigurnost je na iznimno visokoj razini, lako se izbjegavaju uobičajeni sigurnosni propusti i izuzetno je skalabilan. Neke od najprometnijih web stranica ga koriste zbog sposobnosti brzog i fleksibilnog skaliranja [2].

2.1. Povijest

Django je nastao u jesen 2003., kada su web programeri u novinama Lawrence Journal-World, Adrian Holovaty i Simon Willison počeli koristiti Python za izradu aplikacija. Jacob Kaplan-Moss započeo je s radom rano u razvoju Djanga nedugo prije završetka staža Simona Willisona [3]. Javno je objavljen pod licencom BSD-a (Berkeley Software Distribution-a) u srpnju 2005. BSD je operacijski sustav temeljen na Unixu, koji je razvila i distribuirala Computer Systems Research Group na Sveučilištu California u Berkeleyu. Django je dobio ime po gitaristu Djangu Reinhardtu. Adrian Holovaty je romski jazz gitarist i veliki obožavatelj Djanga Reinhardta [4].

U lipnju 2008. objavljeno je da će novoosnovana DSF (Django Software Foundation) održavati Django u budućnosti. Njegova trenutna stabilna verzija je 3.2 koja je objavljena 6. travnja 2021. godine. Instalira se jednostavno preko `pip` sustava za upravljanje paketima napisanom u Pythonu koji se koristi za instaliranje i upravljanje softverskim paketima. Primjer kôda za instaliranje dan je u Ispisu 1 [5]:

```
pip install Django == 3.2
```

Ispis 1: Instalacija Djanga

Za Django su organizirane konferencije i susreti diljem svijeta, a najznačajnija je organizacija *Django Girls* osnovana u Poljskoj, a do danas je susrete imala u 91 različitoj zemlji [5]. Neke od poznatijih tvrtki koje koriste Django su: Instagram, Disqus, Pinterest, Mozilla Firefox, Spotify i YouTube.

2.2. Princip rada

Primarni cilj Djanga je olakšati izradu složenih web stranica na temelju baze podataka. Django naglašava višekratnu upotrebu i "mogućnost uključivanja" komponenti, manje pisanog kôda, brzi razvoj i princip rada u kojem je bespotrebno ponavljati već izrađene dijelove [6]. Python se koristi svugdje, čak i za postavke, datoteke i podatkovne modele. Također nudi izborno sučelje za administrativno stvaranje, čitanje, ažuriranje i brisanje koje se dinamički generira introspekcijom i konfigurira putem administratorskih modela.

Kada koristiti Django i zašto? Prije početka učenja Djanga, preporuča se prvo savladati osnove Pythona. Django je razvojno okruženje temeljeno na Pythonu koji omogućuje brzo stvaranje web aplikacije bez čestih problema s instalacijom ili ovisnostima koje se inače pronalaze s drugim okvirima. Django se koristi za web razvoj u sljedećim slučajevima:

- razvoj web aplikacije ili *back-end* API-je (Application Programming Interface),
- brzi razvoj neke web aplikacije,
- brzo postavljanje aplikacije i njeno skaliranje prema traženim potrebama,
- rad s bazom podataka bez direktnih upita na bazu podataka,
- razvitak sigurne aplikacije za stranicu za dohvaćanje podataka ili objavljivanje podataka.

2.3. Karakteristike Djanga

2.3.1. Osnovni primjer

Nakon instalacije Djanga metodom spomenutom u poglavlju 2.1. potrebno je postaviti radnu okolinu. Svaka aplikacija napisana u Djangu sastoji se od Python paketa koji slijedi određenu konvenciju. Django dolazi s uslužnim programom koji automatski generira osnovnu strukturu direktorija aplikacije, tako da se može usredotočiti na pisanje kôda, a ne

na stvaranje direktorija. Postavljanje radne okoline obuhvaća automatski generiran kôd koji uspostavlja Djangov projekt – odnosno zbirku postavki za instancu Djanga, uključujući konfiguraciju baze podataka, opcije specifične za Django i postavke za aplikaciju. Nakon što je postavljanje gotovo i aplikacija pokrenuta, može se pokrenuti Python kôd u Djangovoj komponenti `View`.

```
from django.http import HttpResposne
def Poruka(request):
     return HttpResponse(„Hello, world!“)
```

Ispis 2: Osnovni primjer korištenja Djanga

Funkcija `Poruka` prima zahtjev (engl. *request*), a vraća HTTP-om (engl. *Hypertext Transfer Protocol*) poruku „Hello, world!“. Da bi ispis radio potrebno je uvesti Djangov modul `HttpResponse`, a da bi se pozvao `View` prvo ga se mora preslikati u URL (Uniform Resource Locator), odnosno kreirati URL na kojem će se ispis prikazati, a za to je potreban `URLconf`. `URLconf` je Python modul koji predstavlja preslikavanje URL izraza na Python funkcije, odnosno na `View`.

2.3.2. Views

`View` je komponenta u Djangovoj arhitekturi koja sadrži logiku koja se prikazuje korisniku. U tradicionalnoj MVC (Model-View-Controller) arhitekturi to se provodi usklađujući se i s modelom i s kontrolerom (engl. *controller*). Problem predstavlja to što `Views` ovise o modelu i kontroleru pa se generiraju pomoću dvije komponente. To krši filozofiju dizajna MVC-a jer jedna komponenta previše ovisi o drugim komponentama.

Dizajn web stranice mijenja se mnogo češće od poslovne logike, a `View` najčešće sadrži oboje, pa se koordinira i s kontrolerom i s modelom. To otežava izradu aplikacije programerima, odnosno usklađivanje podataka s komponentama za prikaz.

Te probleme riješila je Djangova MTV arhitektura gdje `View` sadržava samo poslovnu logiku, tj. objašnjava koju web stranicu treba generirati na koji ulaz, a također je postala posrednik između modela i predloška. Jedno od važnih pitanja koje se ovdje postavlja je - da li je `View` kontroler u Djangu? Odgovor je 'ne' jer Djangove `View` komponente odgovaraju samo određenom predlošku, tehnički ne biraju sami model ili prikaz. Ovo se pitanje javlja jer je kontroler u MVC-u odabir prikaza za određene podatke. Zapravo,

kontroler je sami razvojni okvir (engl. *framework*) Djanga. Kontroler je najsloženiji dio kôda koji mora učinkovito kontrolirati i komunicirati sa svakom komponentom. Stoga je potrebno precizno napisati kôd te se taj problem rješava Django okvirom [7].

2.3.2.1. Class-based View (CBV)

Django je prvo započeo s `View` komponentama temeljenim na funkcijama, ali je zatim dodao `View` komponente temeljene na klasama koje predstavljaju način izrade predloška funkcionalnosti. Iz tog razloga ne treba iznova pisati isti `View` (tj. isti kôd). CBV-ovi su način povećanja produktivnosti tako da se ne mora pisati nepotreban kôd. Organizacija kôda sa određenim HTTP metodama (GET, POST itd.) odvija se tako da se one prikazuju kao zasebne metode umjesto uvjetnog grananja. `Views` na temelju klase ne odvijaju se na temelju `Viewsa` zasnovanih na funkcijama, ali zahvaljujući nasljeđivanju, jednostavno se implementiraju te su optimalniji.

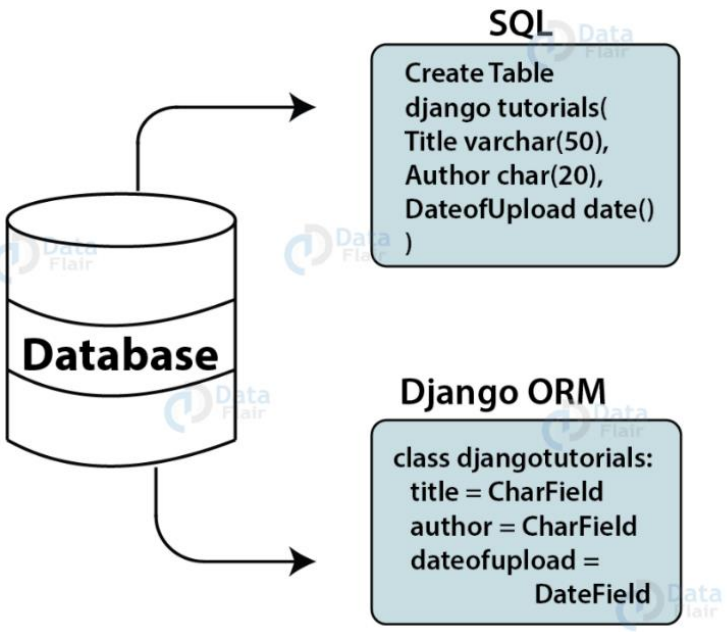
2.3.2.2. Function-based View (FBV)

Funkcije `View` su Python funkcije koja prihvaćaju web zahtjev i daju web odgovor. Taj odgovor je HTML sadržaj web stranice, XML dokument, preusmjeravanje (engl. *redirect*), greška „404“ (greška gdje stranica nije pronađena) ili slika. Sam prikaz `Viewsa` sastoji se od bilo koje proizvoljne logike koja zahtijeva da se vrati taj odgovor.

Ove funkcije su jednostavne za implementaciju i prilično su korisne, ali glavni nedostatak je što na velikom projektu Django općenito postoji mnogo sličnih funkcija. Jedan primjer je da svi objekti Django projekta imaju `create`, `read`, `update`, `delete` ili CRUD operacije, pa se ovaj kôd nepotrebno ponavlja. To je bio glavni razlog stvaranja `Views` zasnovanih na generičkoj klasi.

2.3.3. Models

Model u Djangu je klasa koja je uvezena iz biblioteke django.db te djeluje kao most između baze podataka i poslužitelja. Ova klasa predstavlja strukturu podataka koju koristi web stranica. Izravno će povezati ovu strukturu podataka s bazom podataka i ne zahtijeva znanje SQL-a (Structured Query Language-a) za bazu podataka.



Slika 1: Usporedba tablice baze podataka u SQL-u i Djangu [8]

Model je konačan izvor informacija o podacima. Sadrži bitna polja i pravila o ponašanju podataka koja se pohranjuju. Općenito, svaki se model preslikava u jednu tablicu baze podataka, prikazano na Slici 1. Svaki model je Pythonova klasa koja je podređena klasa komponente django.db.models.Model te Django daje automatski generirani API za pristup bazi podataka [8].

Također, svaka baza podataka implementira SQL na drugačiji način, jer su sve napravljene za izvršavanje različitog zadatka. To programerima stvara problem ukoliko se zadatak može riješiti samo na specifičan način. Tada bi se trebalo uložiti vrijeme u razumijevanje nove baze podataka, ali također pokušati smisliti način kako integrirati poslužitelj s bazom podataka. Djangov ORM je izvrstan alat koji pruža mogućnost pisanja Python struktura podataka umjesto jezika baze podataka te time znatno olakšava rad. To postaje jasno za nekoga tko dobro poznaje Python i pokazuje važnost Django ORM-a nad SQL-om.

Kod kreiranja modela, Django izvršava SQL kako bi dizajnirao odgovarajuću tablicu u bazi podataka bez potrebe za pisanjem ijednog retka SQL-a. Django prefiksira naziv tablice s imenom aplikacije. Također, model povezuje povezane podatke u bazi podataka. Nakon svake izmjene moraju se izvršiti dva ispisa, sa prvim se provjerava ima li promjena u odnosnu na trenutnu bazu podataka i on glasi:

```
python manage.py makemigrations
```

Ispis 3: Provjera izmjena nad bazom

A sa drugim pronađene promjene, ukoliko ih ima, primjenjuju se nad bazom podataka i on glasi:

```
python manage.py migrate
```

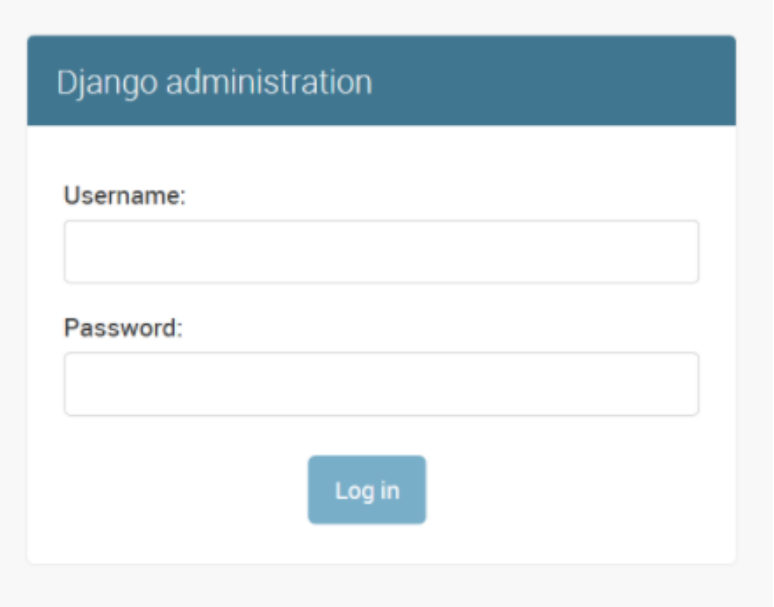
Ispis 4: Izvođenje izmjena nad bazom podataka

Ukoliko se pokuša pokrenuti poslužitelj bez da su registrirane nove izmjene program će na to upozoriti.

2.3.4. Admin i Users

Administratorsko sučelje web stranice je upravljačka ploča web stranice. Administratori web stranice koriste ga za dodavanje, brisanje i uređivanje sadržaja na web stranici, a također i za održavanje drugih funkcionalnosti koje su izgradili programeri.

Django Admin potpuno opremljeno sučelje, unaprijed je učitano i napravljeno da ispuni sve potrebe programera. Nakon pokretanja poslužitelja, URL 'localhost:8000/admin/' vodi na početnu stranicu Django Admin sučelja koje traži podatke za autentifikaciju, kao što je prikazano na Slici 2 [9].



Slika 2: Djangovo Admin sučelje

Prvi korak je kreiranje administratora, koji može izvesti bilo koju manipulaciju podataka ili sadržaja vezanog za web stranicu. Automatizirana izrada administratora je prikazana na Ispisu 5:

```
python manage.py createsuperuser
```

Ispis 5: Kreiranje admin korisnika

To su u osnovi vjerodajnice administratora za web stranicu i kada web stranica postane aktivna, one postaju vitalne informacije za održavanje i kontrolu web stranice. Nakon što se stvori super korisnik, može se prijaviti na administratorsku stranicu. Django Administration nudi opcije izrade korisnika i grupa korisnika. Korisnicima se dodaju podatci. Već definirani model *Users* ima polja: ime, prezime i e-mail, a po želji dodatna polja mogu se definirati u novom modelu korisnika. Lozinke svih korisnika kriptiraju se putem posebnog sigurnosnog ključa u datoteci settings.py. Korisnicima se mogu dodijeliti različita prava poput izmjena, dodavanja, brisanja i pregleda baza i modela. Može im se pridijeliti administrativne privilegije ili im dodati neaktivni status sa kojim se neće moći prijaviti u aplikaciju.

2.3.5. URL-ovi, URLConf i Redirect

URL označava Uniform Resource Locator. To je adresa koju poslužitelj koristi za traženje odgovarajuće web stranice. Svi poslužitelji uzimaju URL koji se pretražuje u pregledniku i putem tog poslužitelja pruža točan rezultat, a ako ne pronađu ništa što odgovara URL-u, obično se prikaže poruku `404 FILE NOT FOUND - ERROR` [9].

Django tumači URL-ove na prilično drugačiji način. URL-ovi u Djangu su u formatu regularnih izraza, koje ljudi lakše čitaju od tradicionalnih URL-ova PHP-a (Hypertext Preprocessor). Regularni izraz koji se također naziva RegEx format služi kao obrazac pretraživanja koji je mnogo čišći i lakši za čitanje i vrlo je logičan. To je važno jer znatno olakšava proces SEO-a (Search engine optimizationa) za razliku od tradicionalnog pristupa URL-om koji sadrži mnogo više posebnih znakova.

```
from django.contrib import admin
from django.urls import path
urlpatterns = [
     path('admin/', admin.site.urls),]
```

Ispis 6: Primjer datoteke urls.py

Kako se traži URL i dostavlja datoteka? Kad se traži URL u traci preglednika isti URL se šalje poslužitelju nakon što se poveže s odgovarajućim poslužiteljem. Dok je Djangov poslužitelj pokrenut, primit će URL i tražiti URL u datoteci urls.py ili vrijednost u varijabli `ROOT_URLCONF`. Varijabla `ROOT_URLCONF` sadrži adresu datoteke urls.py. Poslužitelj će odgovarati URL-ovima iz te datoteke. Unutar datoteke nalazi se kôd prikazan u Ispisu 6.

URL obrasci su popis u koji se spremaju URL-ovi, a `urlpatterns` konvencionalni je naziv. Na ovaj popis dodaju se svi URL-ovi i uzorci koji će biti dostupni za pretraživanje na web stranici. Datoteka urls.py u Djangu je poput adresara web stranice Django. Pohranjuje sve web adrese za web stranicu. To povezuje s nekom komponentom prikaza ili bilo kojom drugom `URLS-CONF` datotekom za određenu aplikaciju.

2.3.6. REST API

Django REST Framework (DRF) omogućuje programerima da brzo izgrade RESTful API-je. RESTful API sučelje je aplikacijskog programiranja koje je u skladu s ograničenjima REST arhitektonskog stila i omogućuje interakciju s web uslugama RESTful. REST predstavlja reprezentativni prijenos stanja, a stvorio ga je informatičar Roy Fielding. API je kratica za Application Programming Interface i koristi se za međusobnu komunikaciju. API koriste dvije aplikacije koje pokušavaju međusobno komunicirati putem mreže ili interneta [10].

Slika 3: Djangov REST API [10].

Slika 3 pomaže u vizualizaciji rada API-ja. API djeluje kao posrednik između Djanga i drugih aplikacija. Druge aplikacije mogu biti s Androida, iOS-a, izrađene u *front-end* bibliotekama React ili Angular i koriste se u raznim web-preglednicima poput Google Chromea, Internet Explorera, Mozilla Firefoxa i Safarija. Glavni zadatak API-ja je primiti podatke iz drugih aplikacija i dostaviti ih u „pozadinu“. Oni su odgovorni za pravilno čišćenje i oblikovanje podataka. Ti podaci su obično u JSON formatu.

REST (Representational State Transfer) je arhitektura na kojoj se razvijaju web usluge. Web usluge mogu se shvatiti kao povezivanje uređaja s internetom. Pretragom informacija na Googleu ili gledanja videa na YouTubeu, koriste se web usluge na kojima uređaj komunicira s poslužiteljem. RESTful web usluge koriste HTTP za prijenos podataka između strojeva.

RESTful API djeluje kao prevoditelj između dva stroja koji komuniciraju putem web usluge. Sličan je API-ju, ali radi na RESTful web usluzi. Web programeri programiraju REST API tako da poslužitelj može primati podatke iz aplikacija.

3. React

Za izradu *front-end* dijela rada korišten je React, biblioteka JavaScripta koja se koristi za izradu korisničkih sučelja (engl. *user interface*, UI). React ne spada u potpunosti u *application library*. Dizajniran je posebno za izradu korisničkih sučelja te kao takav ne sadrži mnoge alate koje većina programera smatra nužnim u izradi aplikacije. Ovo omogućava razvojnom programu slobodu izbora različitih biblioteka za rješavanje zadataka vezanih uz mrežni pristup, lokalno spremanje podataka i sl.

3.1. Povijest

Pojavljivanje Reacta uvelike je povećalo mogućnosti *front-end* razvojnih programera u dizajniranju sučelja koja su prilagođena korisnicima. Da bi se lakše razumio koncept Reacta i njegova uloga odnosno razlog Reactovog nastanka potrebno je proučiti nekoliko posljednjih godina tijekom kojih je sve započelo.

2010. godine tada najpopularnija društvena mreža Facebook i njena istoimena tvrtka započinje se susretati s problemima pri održavanju kôda [11]. Naime, njihova platforma za društvenu mrežu nudi veliki broj mogućnosti (engl. *feature*) stoga je rastao i broj ljudi potrebnih za održavanje kôda kako bi sve funkcioniralo kako treba. Upravo to se pretvorilo u glavni problem koji je zahtijevao rast broja zaposlenika kao i samu kompleksnost kôda što je postao primarni razlog usporenja same tvrtke. Rješenje je bilo nužno i hitno.

Jordan Walke, jedan od softverskih inženjera, bio je zaslužan za izgradnju prototipa današnjeg Reacta inspiriranog sintaksom XPH kojeg je uveo Facebook 2010. godine. XPH je sintaksa koja je ostvarila novi način pisanja *front-end kôd*a u PHP okruženju koje je omogućavalo kreiranje kompozitnih komponenti [12]. Upravo XPH je utjecao na kreiranje i razvijanje JSX sintakse koja se od samog početka koristi s Reactom. Prekretnica razvoja Reacta dogodila se 2012. godine kada je Facebook kupio Instagram koji je odlučio koristiti njihovu novonastalu tehnologiju [13]. Facebook pod pritiskom proglašava React *open-source* projektom 29. svibnja 2013. godine i jezik se počinje širiti programerskom domenom.

Prvi veći val širenja jezika započeo je 2014. godine. Od tada React raste i širi se eksponencijalno. Predstavljena je i mobilna verzija pod nazivom React Native. Neke od

poznatih tvrtki koje koriste React su: Amazon, Apple, Dropbox, Lyft, Netflix, Google, Microsoft, Facebook, Pinterest i Snapchat.

3.2. Princip rada

React je *open-source* projekt koji se koristi za izradu korisničkih sučelja, a koju održava Facebook i grupa različitih samostalnih programera i tvrtki. Nastao je u razdoblju kada se primarni pristup sastavljanja korisničkih sučelja obavljao u MVC okruženju. Naime, React se ne smatra još jednim Javascript MVC okruženjem već bibliotekom odgovornom za sastavljanje korisničkih sučelja preko komponenti. Potiče se izrađivanje ponovno iskoristivih komponenti koje prikazuju podatke na stranici koji se mijenjaju ukoliko se aplikacija koristi ili za prikaz skupa istih podataka sa različitim vrijednostima [15].

React se ne temelji na predlošcima. Većina korisničkih sučelja *web* stranica izrađivalo se korištenjem predložaka ili HTML direktive. Nedostatak predložaka je ograničenost seta apstrakcije koji se kombinira kako bi se sastavio željeni produkt. Upravo React nudi drugačiji pristup kreiranja sučelja dijeleći ih na komponente. Komponente složene na taj način prikazuju se na sučelju i pružaju sljedeće prednosti naspram predložaka [14]:

- JavaScript je veoma fleksibilan i snažan programski jezik koji omogućava izgradnje apstrakcija (važna pri izradi velikih i visoko prilagodljivih aplikacija),
- Ujedinjuje odgovarajuću logiku unutar komponente se prezentacijskim jezikom što olakšava daljnji razvitak, održavanje i testiranje samostalnih komponenti,
- Koristi JavaScript sintaksu koja nudi čitljivi oblik HTML sintakse.

3.3. Karakteristike Reacta

3.3.1. Osnovni primjer

Osnovni primjer prikazan je u donjem ispisu kôda:

```
function Poruka(props) {
      return <h1>{props.pozdrav}</h1> }
var App = <Poruka pozdrav = „Hello, World!“ />;
ReactDOM.render(App, document.getElementById('root'));
```

Ispis 7: Osnovni primjer korištenja Reacta

Funkcija `Poruka` je React komponenta koja prima atribut `pozdrav`. Varijabla `App` je instanca komponente `Poruka` gdje je atribut `pozdrav` zadan kao funkcija koja ispisuje poruku „Hello World!“. Metoda `render` kreira zadanu komponentu `Poruka` unutar DOM elementa sa ID-jem `primjerReactApp` (Ispis 8).

```
<div class = „Poruka“>
      <h1>Hello World!</h1>
</div>
```

Ispis 8: Rezultat prikazan u browseru

React komponente pišu se korištenjem Javascript ili JSX sintakse. JSX ili JavaScript XML, je proširenje sintakse JavaScript jezika. Izgledom sličan HTML-u, JSX omogućava strukturiranja komponenti koje se prikazuju na način koji je poznat mnogim programerima (Ispis 9). JSX je sličan još jednom proširenju sintakse koju je smislio Facebook za PHP zvan XHP.

```
class App extends React.Component {
      render() {
            return(
            <div>
                  <p>Header</p>
                  <p>Header</p>
                  <p>Header</p>
            </div> );}}
```

Ispis 9: Primjer JSX sintakse

Korišteni elementi na najvišoj razini moraju biti sadržani unutar unikatnog spremnik elementa kao što je `<div>` element prikazan u kôdu ili kao niz. Reactu su pridruženi razni atributi pomoću elemenata dizajniranih na način da oponašaju one elemente kakve nudi HTML sintaksa. Također samostalno kreirani atributi također se mogu proslijediti komponenti. Svi atributi koje komponenta primi nalaze se unutar svojstva `props` [15].

JavaScript izrazi također se mogu koristiti unutar JSX sintakse uporabom vitičastih zagrada „{}“.Ispis 10. Rezultat kôda je suma operacije, te kao takva je prikazana na pregledniku.

```
// Kod pisan u JSX-u
     <h1>{10+5}</h1>
// Kod prikazan u browseru
     <h1>{15}</h1>
```

Ispis 10: Primjer JavaScript izraza

3.3.2. Komponente

React kôd sastavljen je od entiteta koje se nazivaju komponente. Komponente se prikazuju unutar određenog elementa u DOM-u korištenjem React DOM biblioteke. Tijekom generiranja komponente moguće joj je proslijediti vrijednosti koje se nazivaju `props` (svojstva engl. od *property*). Razlikuju se dva primarna načina definiranja komponenti u Reactu, a to su funkcionalne komponente i `Class-based` komponente [15].

3.3.3. Funkcionalne komponente

Funkcionalne komponente izrađuju se kao funkcije, zatim vračaju Javascript kôd (Ispis 11). Jednostavnije su od `class-based` komponenti i ne sadrže `state`. U praksi cilj je koristiti što više funkcionalnih komponenata a `class-based` samo gdje je nužno.

```
// Poziv funkcionalne komponente
<Poruka name={„John“} />
//Definicija
function Poruka (props) {
      return (
            <div className=“Poruka“>
                  Hello, {props.name}
            </div> );}
```

Ispis 11: Jednostavna funkcionalna komponenta

3.3.4. Class-based komponente

`Class-based` komponente izrađuju se pomoću ES6 klasa (Ispis 12). Također se nazivaju i `stateful` komponente jer sadrže `state`, varijablu nalik na objekt koja sadržava vrijednosti dohvatljive unutar cijelog prikaza. Vrijednosti iz `statea` također se mogu proslijediti `child` komponenti korištenjem `props` atributa.

```
class ParentComponent extends React.Component {
      constructor(props){
            super(props);
            this.state={
                  Name: „John“ }}
      render() {
            return (
                  <ChildComponent ime={this.state.name} /> );}}
```

Ispis 12: Jednostavna class-based komponenta

3.3.5. State

`State` sadrži određene podatke povezane s komponentom u kojoj je definiran. `Stateu` se ne može pristupiti izvan komponente niti ga se može izmijeniti, s obzirom da je dostupan samo unutar vlastite komponente te funkcionira na sličan princip kao lokalni doseg obične funkcije. U funkciji možemo definirati varijable koje je moguće koristiti jedino unutar njenog dosega.

3.3.6. Props

`Props` su jedan od bitnijih oblika podataka u Reactu bez kojih on ne bi imao smisla. React komponenta je prilagođena za višekratnu upotrebu (engl. *reusable*) koja se stalno koristi i prikazuje iznova u korisničkom sučelju, ali nije cilj uvijek prikazati komponentu sa istim podacima. Ponekad je potrebno izmijeniti podatke ili sadržaj komponente, te je upravo iz tog razloga u React uveden `props` (Ispis 13).

```
//Korištenje komponente props
<Person name="Mark" />
<Person name="John" />
Class Person extends React.Component {
     Render() {
          Return <div>Hello {this.props.name}</div>; }}
//Rezultatt prikaza
<div>Hello Mark</div>
<div>Hello John</div>
```

Ispis 13: Primjer korištenja `propsa`

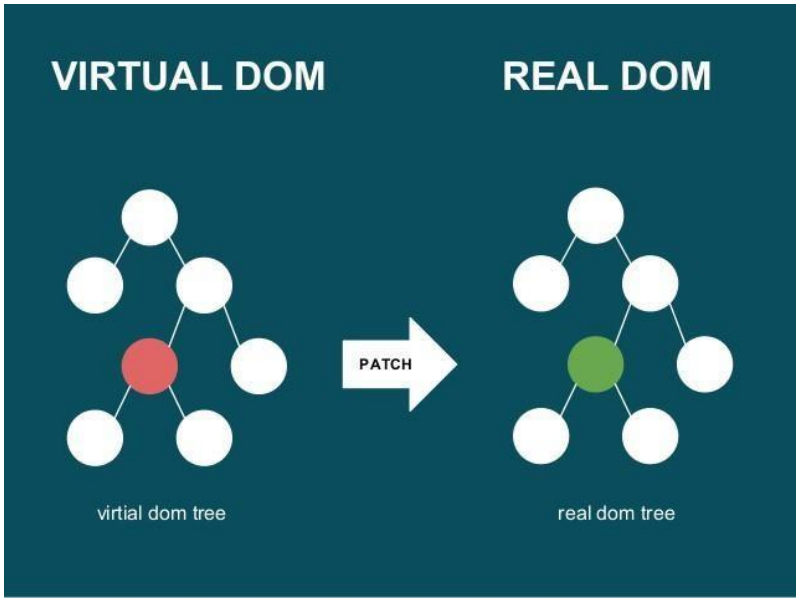
3.3.7. Virtual DOM

Jedna od bitnijih karakteristika je metoda korištenja *virtual Document Object Modela ili virtual DOM*. React ima različit pristup rukovanja s podacima od uobičajenog pristupa Javascript aplikacije. Obično se promatra način na koji se podaci mijenjaju te se zatim mijenja i sučelje kako bi bilo ažurno.

React prilikom otvaranja stranice, kada se komponenta inicijalizira, poziva metodu *render* koja prikazuje definirane komponente. Generirani prikaz nalazi se u memoriji kao tzv. *virtual* DOM. Na temelju tog prikaza generira se konačni HTML koji se prikaže u aplikaciji. Utjecajem promjena tj. izmjenom podataka opet se poziva `render` metoda koja uspoređuje utjecaj promjena na virtualni DOM.

Upravo to je razlog Reactove brzine [16]. Uspoređuje se DOM koji je spremljen u memoriji, tj. virtualni DOM što ubrzava proces provjere razlika. Zatim se samo te promjene spreme u stvarni DOM (Slika 4). Taj proces nazivamo *reconciliation* i događa se prilikom svake izmjene podataka. Aplikacija reagira (engl. *React*) na promjenu, odakle je potakao i

naziv ove biblioteke. Upravo ova brzina promjene (oko 1ms) omogućava programeru da izbjegne izričito vezanje podataka.



Slika 4: Virtual DOM [16]

3.3.8. Lifecycle metode

`Lifecycle` metode omogućavaju izvođenje kôda u određenim točkama tijekom životnog vijeka komponente, a te metode nazivaju se `hooks`. Pojam *hooking* u programiranju označava razne tehnike korištene za modificiranje i izmjenu operativnog sustava, aplikacije ili neke druge programske komponente, na način da presretne pozive funkcija, poruke ili događaje koji se prosljeđuju između komponenti [15].

`Lifecycle` metode mogu se koristiti unutar `class-based` komponenti. Razlikuju se sljedeće `Lifecycle` metode ovisno o specifičnim vremenima u procesu kad se pozivaju (Slika 4) [17]:

`Mounting (stavljanja elemenata u DOM):`

- `constructor`() – omogućava inicijalizaciju komponente i postavljanju njenog početnog stanja.
- `render`() – glavna metoda koja obavlja prikazivanje sadržaja komponente u sučelju.
- `componentDidMount`() – poziva se kad komponenta završi `mountanje` i služi

najčešće za dohvaćanje podataka i asinkrone pozive.

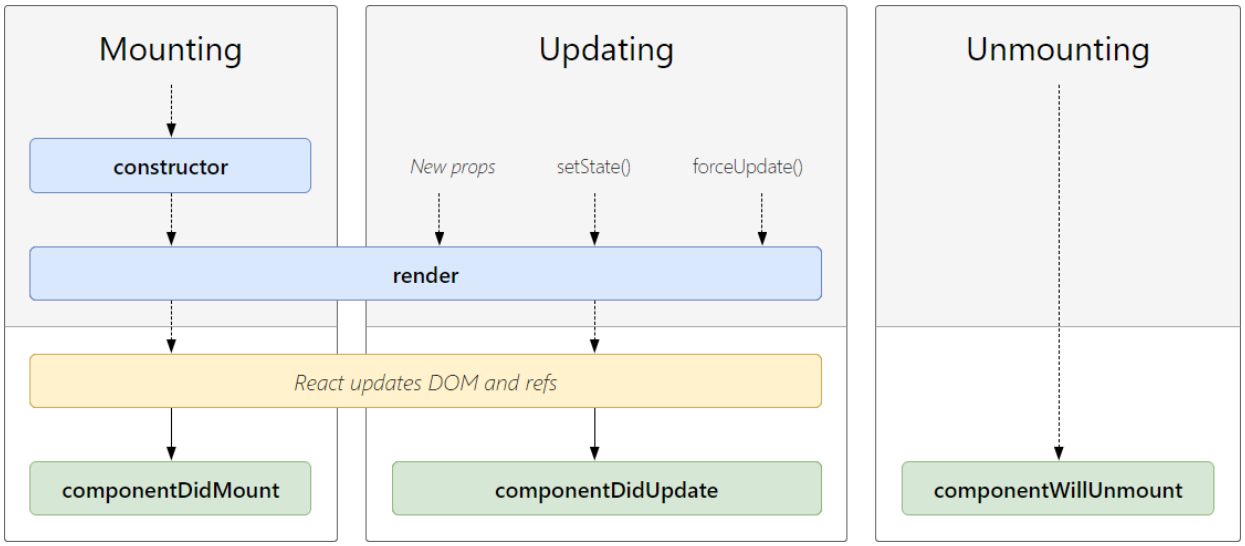
`Unmounting` (uklanjanje elemenata iz DOM-a):

- `componentWillUnmount`() – aktivira se točno prije nego li se komponenta uklanja sa DOM-a. Ovo je općeniti način za oslobađanje resursa vezanih zavisno o komponenti koji se ne mogu samostalno ukloniti ili zaustaviti prilikom uklanjanja njihove komponente(npr. `setInterval`()) instance koje su odnose na komponentu, ili bilo koje vrsta `eventListener` postavljena na `document` zbog prisustva komponente).

`Updating` (ažuriranje DOM-a):

- `render`() – opisano u `mounting` procesu.
- `compoonentDidUpdate`() – poziva se nakon što se završi ažuriranje komponente. Ova metoda se pri prvom prikazivanju komponente ne poziva, već isključivo kôd ažuriranja. Moguće je pozvati `setState`() metodu ali obavezno mora biti obuhvaćena `if` uvjetom kako bi se priječilo nastajanje beskonačne petlje.

Osim spomenutih postoji još `lifecycle` metoda, ali koristi se isključivo u određenim rubnim slučajevima kada je potrebno specifično rješenje.



Slika 5: Faze životnog ciklusa Reacta [17]

4. Docker

Docker je skup platformi kao usluga (PaaS) proizvoda koji koriste virtualizaciju na razini OS-a za isporuku softvera u paketima koji se nazivaju spremnici (engl. *container*). Kontejneri su međusobno izolirani i povezuju vlastiti softver, knjižnice i konfiguracijske datoteke; mogu međusobno komunicirati dobro definiranim kanalima. Budući da svi spremnici dijele usluge jedne jezgre operacijskog sustava, oni koriste manje resursa od virtualnih strojeva. Usluga ima besplatnu i premium verziju. Softver koji ugošćuje spremnike naziva se Docker Engine. Prvi put je pokrenut 2013. godine, a razvio ga je Docker, Inc [18].

Docker može zapakirati aplikaciju u virtualni spremnik koji se može izvoditi na bilo kojem Linux, Windows ili macOS računalu. To omogućuje izvođenju aplikacije na različitim lokacijama, kao što je lokalno, u javnom oblaku i/ili u privatnom oblaku. Prilikom rada na Linuxu, Docker koristi značajke izolacije resursa jezgre Linuxa (kao što su cgroups i jezgrini imenski prostori) i datotečni sustav sposoban za povezivanje (poput OverlayFS-a) kako bi omogućio pokretanje spremnika unutar jedne instance Linuxa, izbjegavajući tako pokretanje i održavanje virtualnih strojeva. Docker na macOS-u koristi virtualni stroj Linux za pokretanje spremnika.

Budući da su Docker spremnici lagani, jedan poslužitelj ili virtualni stroj mogu istodobno pokrenuti nekoliko spremnika. Analiza iz 2018. otkrila je da tipičan slučaj upotrebe Dockera uključuje pokretanje osam spremnika po *hostu*, te da četvrtina analiziranih organizacija radi 18 ili više po *hostu* [19].

Podrška jezgre Linuxa za imenske prostore uglavnom izolira pogled aplikacije na operativno okruženje, uključujući stabla procesa, mrežu, korisničke ID-eve i montirane datotečne sustave, dok jezgrine cgroups pružaju ograničenje resursa za memoriju i CPU. Od verzije 0.9, Docker uključuje vlastitu komponentu (engl. *libcontainer*) za korištenje mogućnosti virtualizacije koje pruža izravno jezgra (engl. *kernel*) Linuxa, uz korištenje apstraktnih sučelja za virtualizaciju.

Docker implementira API visoke razine za pružanje lakih spremnika koji pokreću procese izolirano. Docker spremnici su standardni procesi i moguće je koristiti značajke

jezgre za praćenje njihovog izvođenja uključujući promatranje i posredništvo u sistemskim pozivima.

4.1. Komponente Dockera

4.1.1. Docker Image

Docker slika (engl. *image*) je datoteka koja se koristi za izvršavanje kôda u Docker spremniku. Docker slike djeluju kao skup uputa za izgradnju Docker spremnika, poput predloška. Slike Dockera također služe kao početna točka pri korištenju Dockera. Slika je usporediva sa snimkom u okruženjima virtualnih strojeva [20].

Docker se koristi za stvaranje, pokretanje i postavljanje aplikacija u spremnike. Docker slika sadrži kôd aplikacije, knjižnice, alate, ovisnosti i druge datoteke potrebne za pokretanje aplikacije. Kada korisnik pokrene sliku, ona može postati jedna ili više instanci spremnika.

Docker slike imaju više slojeva, svaki potječe iz prethodnog sloja, ali se razlikuje od njega. Slojevi ubrzavaju izgradnju Dockera, povećavajući ponovnu upotrebu i smanjujući upotrebu diska. Slojevi slike također su datoteke samo za čitanje. Nakon što se stvori spremnik, sloj koji se može upisivati dodaje se na nepromjenjive slike, dopuštajući korisniku unošenje promjena.

Reference na prostor na disku u Docker slikama i spremnicima mogu biti zbunjujuće. Važno je razlikovati veličinu od virtualne veličine. Veličina se odnosi na prostor na disku koji *writeable* sloj spremnika koristi, dok je virtualna veličina prostor na disku koji se koristi za spremnik i sloj za upisivanje. Slojevi slike samo za čitanje mogu se dijeliti između bilo kojeg spremnika pokrenutog s iste slike.

4.1.2. Docker Container

Docker Spremnik je standardna programska jedinica koja pakira kôd i sve njegove ovisnosti pa se aplikacija izvodi brzo i pouzdano iz jednog računalnog okruženja u drugo. Slika Docker spremnika lagan je, samostalni, izvršni paket softvera koji uključuje sve potrebno za pokretanje aplikacije: kôd, vrijeme izvođenja, sistemske alate, knjižnice sustava i postavke. Slike spremnika postaju spremnici za vrijeme izvođenja, a u slučaju Docker

spremnika - slike postaju spremnici kada se pokreću na Docker Engineu. Dostupan i za aplikacije temeljene na Linuxu i Windowsu, kontejnerski softver uvijek će raditi isto, bez obzira na infrastrukturu. Spremnici izoliraju softver od njegovog okruženja i osiguravaju da radi jednoliko unatoč razlikama, na primjer između razvoja i postavljanja.

Docker spremnici koji rade na Docker Engineu su:

- **Standardni spremnik**: Docker je stvorio industrijski standard za kontejnere kako bi mogli biti prenosivi bilo gdje,
- **Laki spremnik**: spremnici dijele jezgru OS-a stroja i stoga ne zahtijevaju OS po aplikaciji, povećavajući učinkovitost poslužitelja i smanjujući troškove poslužitelja i licenciranje,
- **Sigurni spremnik**: aplikacije su sigurnije u spremnicima, a Docker pruža najjače zadane mogućnosti izolacije u industriji.

4.1.3. Docker Hub

Docker Hub je web usluga spremnika koju nudi Docker za pronalaženje i dijeljenje slika spremnika s drugima. Ključne značajke uključuju [21]:

- **Privatna spremišta**: postavljanje i skidanje slika,
- **Automatske gradnje**: automatska izgradnja slika s GitHub-a i Bitbucket-a i postavljanje na Docker Hub,
- **Timovi i organizacije**: upravljanje pristupom privatnim spremnicima,
- **Službene slike**: korištenje visokokvalitetnih slika koje nudi Docker,
- **Slike izdavača**: korištenje visokokvalitetnih slika spremnika koje pružaju vanjski dobavljači. Certificirane slike također uključuju podršku i jamstvenu kompatibilnost s Docker Enterpriseom.

4.2. Implementacija Dockera

Docker se može implementirati na dva različita pristupa. U prvome aplikacija se od početka izrade „zapakira“ u Docker spremnik, a zatim kroz proces izrade ažurira skupa sa svim potrebnim komponentama. Drugi pristup je izrada Docker slike gotove aplikacije. Docker može pokrenuti bilo koju aplikaciju sve dok se može instalirati i izvršavati bez

nadzora, a osnovni operacijski sustav podržava aplikaciju. Stoga je ovo pristup koji se češće koristi, te je i način koji je korišten u ovome radu.

Nakon preuzimanja i instalacije, Dockeru prvo treba proslijediti instrukcije koje Djangove pakete i module treba instalirati prije nego izradi sliku aplikacije, to se postiže kôdom prikazanim na Ispisu 14, koji stvara listu potrebnih paketa koje Docker treba instalirati, a koji se već nalaze u projektu.

```
pip freeze > requirements.txt
```

Ispis 14: Popis korištenih paketa za instalaciju u Dockeru

Docker može automatski graditi slike čitajući upute iz *Dockerfile-a* (Docker datoteke). *Dockerfile* je tekstualni dokument koji sadrži sve naredbe koje korisnik može pozvati u naredbenom retku za sastavljanje slike. Unutar projekta kreirana je Docker datoteka (Ispis 15) u kojemu je postavljen Python kao jezik u kojemu je izrađena aplikacija i definirane naredbe: za ažuriranje i instaliranje potrebnih paketa za spajanje na bazu podataka, za definiranje radnog prostora Dockera, za kopiranje datoteke sa popisom Djangovih paketa potrebnih za rad, za kopiranje cijelog projekta i za pokretanje projekta.

```
FROM python:3
RUN apt-get update && apt-get install -y --no-install-recommends \
    unixodbc-dev \
    unixodbc \
WORKDIR /myproject
COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt
COPY . .
CMD ["python3", "myproject/manage.py", "runserver", "0.0.0.0:8000"]
```

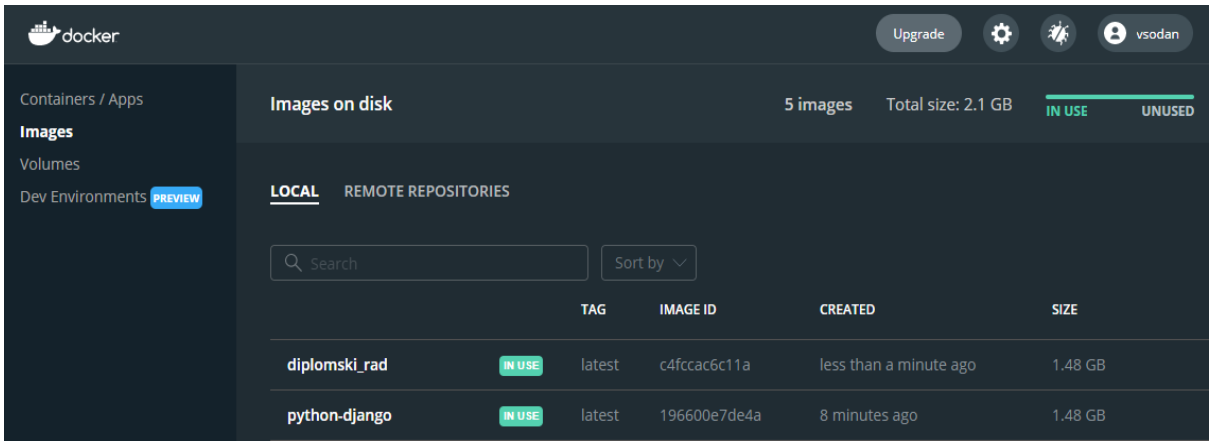
Ispis 15: Docker datoteka

Nakon izrade Docker datoteke napravljena je slika aplikacije naredbom prikazanom u Ispisu 16, Docker slici je dodano ime `diplomski_rad`, a točkom na kraju naredbe navodi se da se u sliku ugrade sve datoteke iz prostora na koji se prethodno pozicioniralo.

```
docker build -tag diplomski_rad .
```

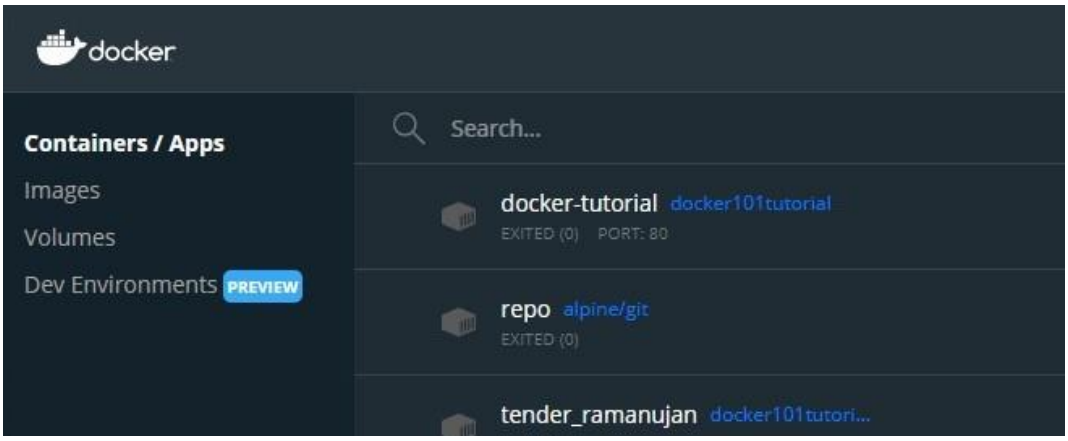
Ispis 16: Naredba za izradu Docker slike

Docker tada u terminalu izvršava sve prethodno navedene naredbe i ukoliko ih sve uspješno izvrši u Docker desktop aplikaciji se prikaže slika aplikacije u prostoru `Images`, kao što je prikazano na Slici 6, gdje se nakon izvršavanja naredbi izgradila slika `diplomski_rad`.



Slika 6: Prikaz izgrađene Docker slike

Docker slika tada se može pokrenuti naredbom `run`, nakon koje se u prostoru `Containers / Apps` u Docker aplikaciji prikaže pokrenuti Docker spremnik iz slike `diplomski_rad`, kao što je prikazano na Slici 7. Docker automatski generira imena spremnicima.



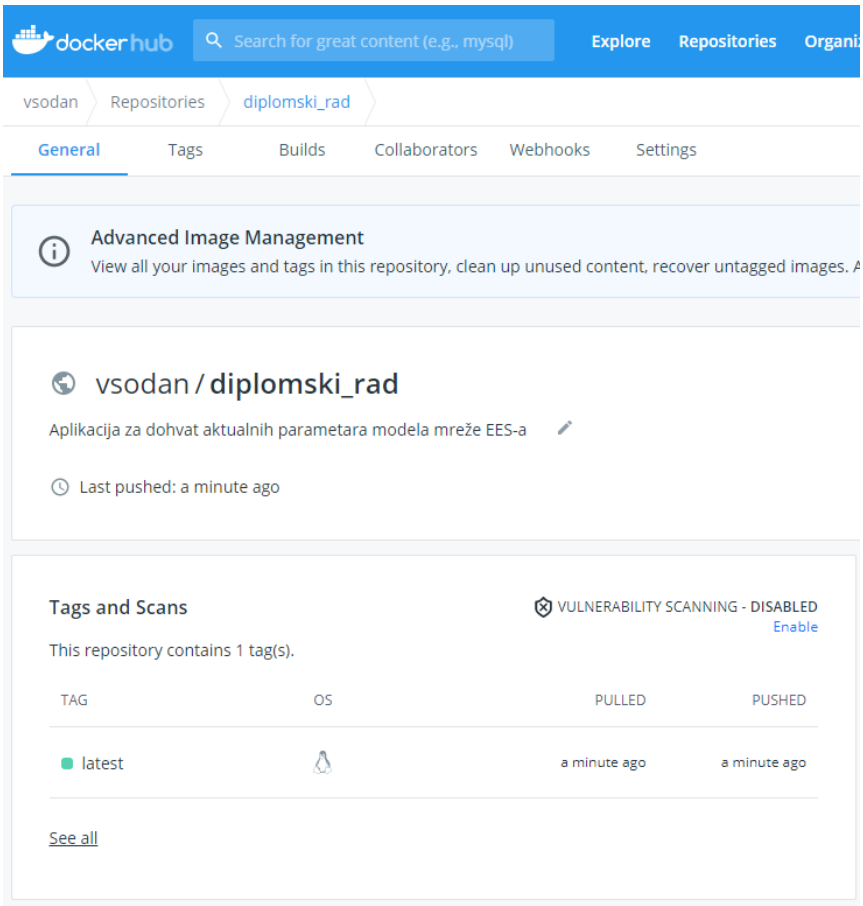
Slika 7: Prikaz slike pokrenute u Docker spremniku

Zadnji korak implementacije Dockera je objava Docker slike u Docker Hub-u kako bi bila spremna za skidanje i korištenje drugima. To se može napraviti kroz Docker aplikaciju ili ispisivanjem naredbe u terminalu kao što je prikazano u Ispisu 17.

```
docker push vsodan/diplomski_rad: diplomski_rad
```

Ispis 17: Naredba za objavljivanje Docker slike na Docker Hub

Slika izrađene aplikacije tada je objavljena na Docker Hubu kao što je prikazano na Slici 8. Na Docker Hubu također se može ograničiti tko sliku može vidjeti, a tko skinuti. Slika je postavljena kao „javna“ da bi bila dostupna drugim korisnicima.



Slika 8: Docker slika aplikacije objavljena na Docker Hubu

5. Active Directory

Active Directory (AD) usluga je imenika (engl. *directory*) koju je Microsoft razvio za mreže domene Windowsa. Uključen je u većinu operacijskih sustava Windows Server kao skup procesa i usluga [22]. U početku se Active Directory koristio samo za centralizirano upravljanje domenom. Međutim, Active Directory je na kraju postao naslov za širok raspon usluga vezanih uz identitet temeljenih na imeniku [23].

Poslužitelj koji izvodi ulogu usluge Active Directory Domain Service (ADDS) naziva se kontroler domene. Ovjerava i ovlašćuje sve korisnike i računala u mreži tipa Windows domene, dodjeljuje i provodi sigurnosna pravila za sva računala te instalira ili ažurira softver. Na primjer, kada se korisnik prijavi na računalo koje je dio Windows domene, Active Directory provjerava poslanu lozinku i utvrđuje je li korisnik administrator sustava ili normalan korisnik. Također, omogućuje upravljanje i pohranu informacija, pruža mehanizme provjere autentičnosti i autorizacije te uspostavlja okvir za implementaciju drugih povezanih usluga: usluga certifikata, federativnih usluga aktivnog imenika, laganih imeničkih usluga i usluga upravljanja pravima.

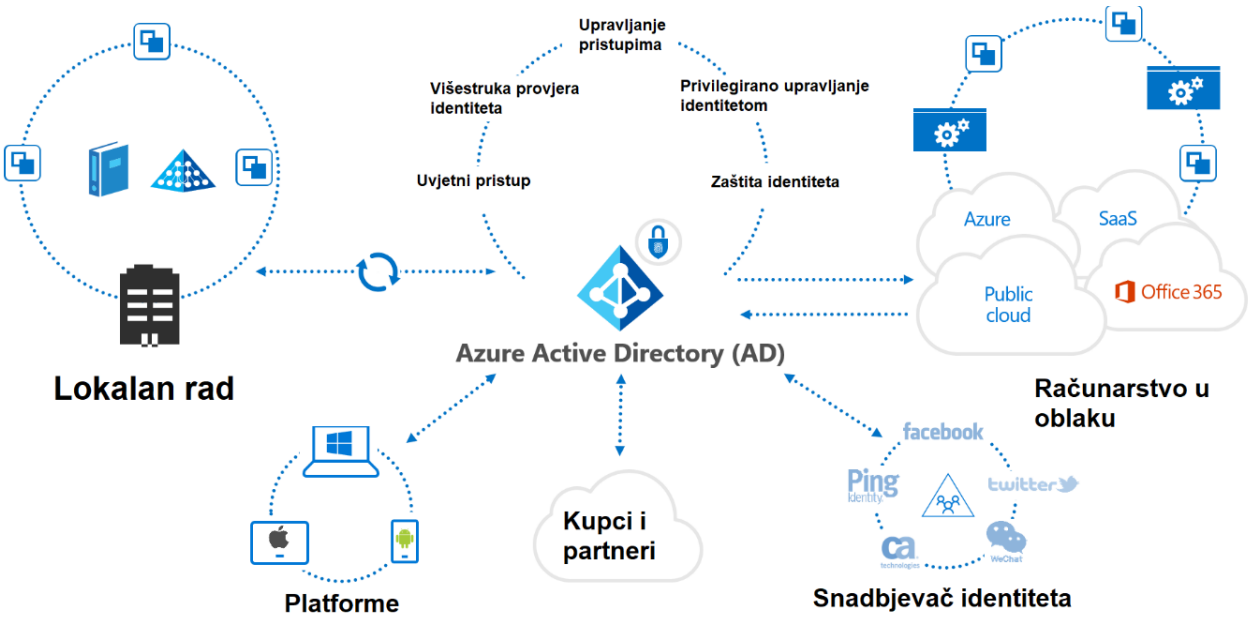
Active Directory pomaže organizirati korisnike tvrtke, računala i još mnogo toga. IT administrator koristi AD za organiziranje potpune hijerarhije tvrtke od toga koja računala pripadaju kojoj mreži, do toga kako izgledaju profilne slike korisnika ili koji korisnici imaju pristup skladišnoj prostoriji.

5.1. Azure Active Directory

Azure AD nije samo *cloud* verzija AD-a kako bi naziv mogao sugerirati. Azure Active Directory sigurni je internet poslužitelj za provjeru autentičnosti koji može sadržavati korisnike i grupe. Korisnici imaju korisničko ime i lozinku koji se koriste kada se prijavljuju u aplikaciju koja koristi Azure AD za provjeru autentičnosti. Tako na primjer sve Microsoft Cloud usluge koriste Azure AD za provjeru autentičnosti: Office 365, Dynamics 365 i Azure [24].

Osim upravljanja korisnicima i grupama, Azure AD upravlja i pristupom aplikacijama koje rade sa suvremenim mehanizmima provjere autentičnosti poput SAML -a

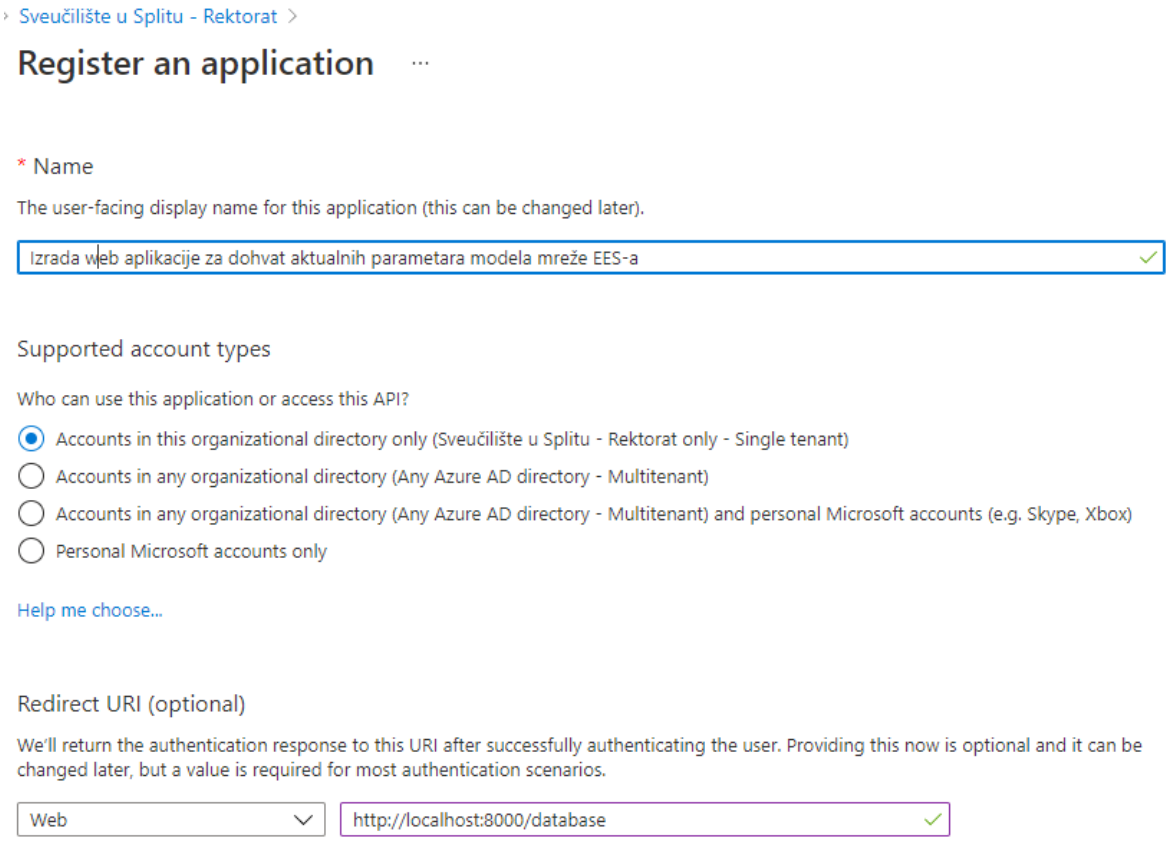
i OAuth-a. Aplikacije su objekt koji postoji u Azure AD -u, a to omogućuje stvaranje identiteta za svoje aplikacije (ili aplikacije treće strane) kojima možete odobriti pristup korisnicima. Osim besprijekornog povezivanja s bilo kojim Microsoftovim mrežnim uslugama, Azure AD može se povezati s tisućama SaaS aplikacija (npr. Salesforce, Slack, ZenDesk itd.) pomoću jedne prijave. AD izvrsno upravlja tradicionalnom lokalnom infrastrukturom i aplikacijama. Azure AD izvrstan je u upravljanju korisničkim pristupom aplikacijama u oblaku. Rade različite stvari, a područje preklapanja je upravljanje korisnicima. Detalji funkcioniranja Azure Active Directorya prikazani su na Slici 9.



Slika 9: Detalji funkcioniranja Azure Active Directorya

5.2. Implementacija Active Directorya

Za potrebe rada korišten je Azure Active Directory rektorata Sveučilišta u Splitu. Prvi korak nakon prijave sa korisničkim podacima Sveučilišta u Splitu je registriranje aplikacije u Azure AD sustavu. Kroz registraciju aplikacije u sustavu postavlja se tko će aplikaciji imati pristup te domena aplikacije na koju će AD sustav proslijediti nakon uspješne autentifikacije, kao što je prikazano na Slici 10. Po završetku dobije se Id aplikacije koji služi za konfiguraciju u Djangu.



Slika 10: Registracija aplikacije u Azure AD sustavu

Zatim su dodani opsezi dopuštenja (engl. *scope of permissions*) u novo registriranu aplikaciju na način da je u izborniku nakon biranja stavke „Expose API“ na bočnoj traci odabrana metoda „add Scope“ te je u njoj izrađeno pravo pristupa sa sljedećim svojstima:

- ***Delegated permissions*:** API-u aplikacije imaju pristup samo prijavljeni korisnici,
- ***Admin and users:*** aplikaciji mogu pristupiti svi korisnici i administratori,
- ***Read-only:*** aplikaciji se može pristupiti, ali ne i mijenjati je.

Sljedeći korak je u Django datoteku `settings.py` dodati konfiguraciju kao što je prikazano u Ispisu 18.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.BasicAuthentication',

'django_auth_adfs.rest_framework.AdfsAccessTokenAuthentication',
),}
client_id = '81818a59-107d-49c1-a30b-8344f5e9c496'
client_secret = '6eac75f9-91ef-4885-a811-bf0e751c185c'
tenant_id = 'ba2fa2d1-38b6-47ce-806c-0bce7f022431'
AUTH_ADFS = {
    'AUDIENCE': client_id,
    'CLIENT_ID': client_id,
    'CLIENT_SECRET': client_secret,
    'CLAIM_MAPPING': {'first_name': 'given_name',
                      'last_name': 'family_name',
                      'email': 'upn'},
    'GROUPS_CLAIM': 'roles',
    'MIRROR_GROUPS': True,
    'USERNAME_CLAIM': 'upn',
    'TENANT_ID': tenant_id,
    'RELYING_PARTY_ID': client_id,}
```

Ispis 18: Konfiguracija AD u Djangu

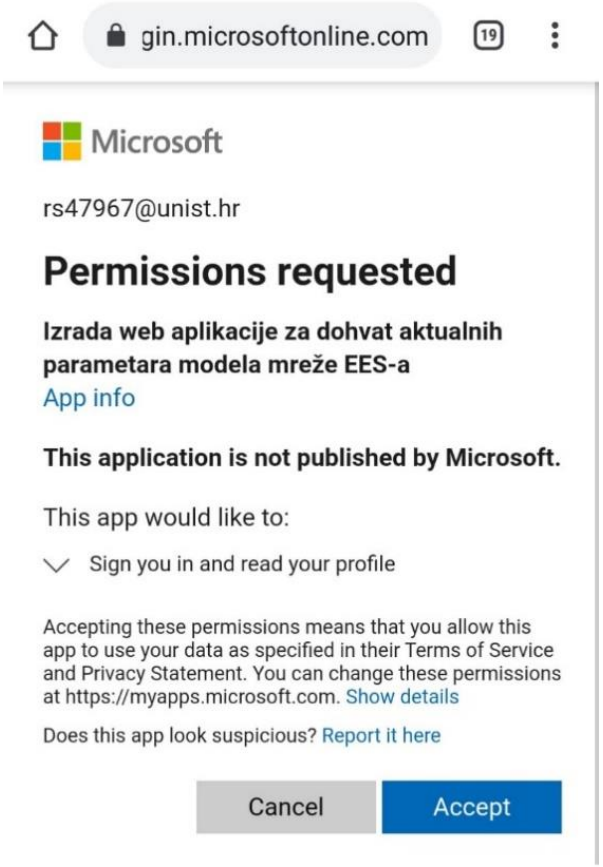
U `REST_FRAMEWORK` dodane su *back-end* klase u kojima Djangu definira se koju će klasu koristiti za autentifikaciju. `client_id` označava ID aplikacije koji je dodijeljen po završetku registriranja aplikacije u AD sustavu, a `client_secret` i `tenant_id` su ID korisnika koji je registrirao aplikaciju.

Zadnji korak uključuje definiranje URL putanje prema OAuth protokolu za autentifikaciju pristupa, kao što je prikazano u Ispisu 20.

```
urlpatterns = [
    path('oauth2/', include('django_auth_adfs.urls')),]
```

Ispis 20: Definiranje URL-a protokola za autentifikaciju

Pristup aplikaciji je omogućen za sve članove Active Directorya rektorata Sveučilišta u Splitu, a primjer prijave studenta Roka Švenjaka prikazan je na Slici 11. Nakon uspješne prijave preko AAI@EduHr sustava (Autentifikacijska i autorizacijska infrastruktura sustava znanosti i visokog obrazovanja u Republici Hrvatskoj), preko kojeg su korisnici Sveučilišta u Splitu registrirani u Microsoftovom AD sustavu, korisnik je preusmjeren u izbornik web aplikacije za dohvat aktualnih parametara modela mreže EES-a.



Slika 11: Primjer prijave u aplikaciju

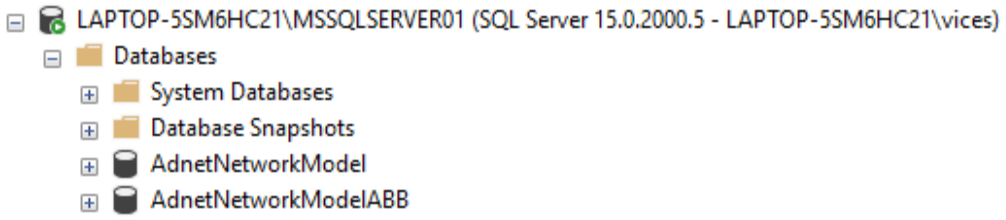
6. Aplikacija

6.1. Inspiracija teme

Temelj same aplikacije je pokušaj da se informacijskim tehnologijama ubrza i olakša izvođenje jednog poslovnog procesa. Tema je inspirirana trenutnim načinom dohvaćanja elemenata EES-a u tvrtki HOPS. Trenutno programeri u tvrtki ručno dohvaćaju podatke na upite radnika i na zahtjeve klijenata s kojima posluju. Problem je što korisnici koji ne posjeduju znanje SQL-a i/ili nisu upoznati sa hijerarhijom baza podataka nemaju pristup potrebnim podacima. Ideja aplikacije je riješiti taj problem.

6.2. Baze Podataka

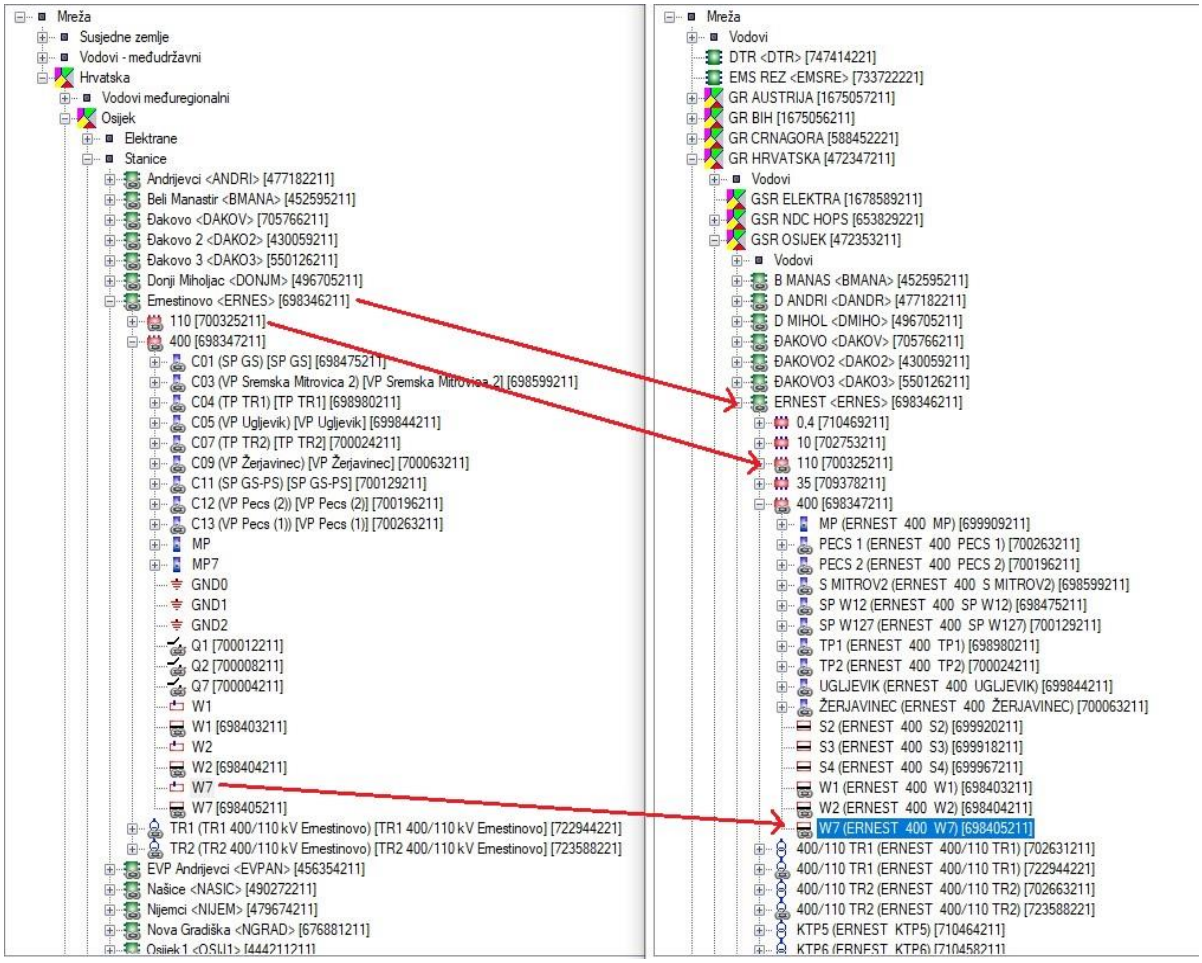
Za potrebe rada korištene su dvije baze podataka sa parametrima modela mreže EES-a Hrvatske: `AdnetNetworkModel` i `AdnetNetworkModelABB`. Baze podataka su vraćene (engl. *restore*) na MS SQL poslužitelju (prikazano na Slici 12).



Slika 12: Izbornik MS SQL-a s korištenim bazama podataka

Podaci su u bazama složeni u strukturu stabla. Na Slici 13 prikazana je struktura i povezanost obje baze podataka, lijevo se nalazi `AdnetNetworkModel`, a desno `AdnetNetworkModelABB`. Svaki element, odnosno mrežni objekt, u bazi ima svoje identifikacijsko polje (`Id`) koje je ujedno i roditeljsko polje člana iznad u hijerarhiji. `Id` svakog člana sadrži tablica `PowerSystemResource` u polju `PowerSystemResourceId`, a isti `Id` je i `ParentId` svim članovima kojima je taj član nadređen. Najviši član u hijerarhiji je objekt `Mreža`. Objekti `Susjedne zemlje`, `Vodovi – međudržavni` i `Hrvatska` su sljedeći u hijerarhiji, te je njihov `ParentId` zapravo `PowerSystemResourceId` od polja `Mreža`. Konkretno, u prikazanom primjeru odabran je objekt `W7` i to slijedom odabira: `Mreža` → `Hrvatska` → `Osijek` → `Stanice` → `Ernestinovo` → `400` → `MP7` → `W7`.

Svaki objekt sadrži `PowerSystemResourceId` naređenog objekta u svome `ParentId` polju. Pomoću navedenih polja povlače se i prikazuju elementi iz baza podataka te daju na odabir korisniku u aplikaciji.

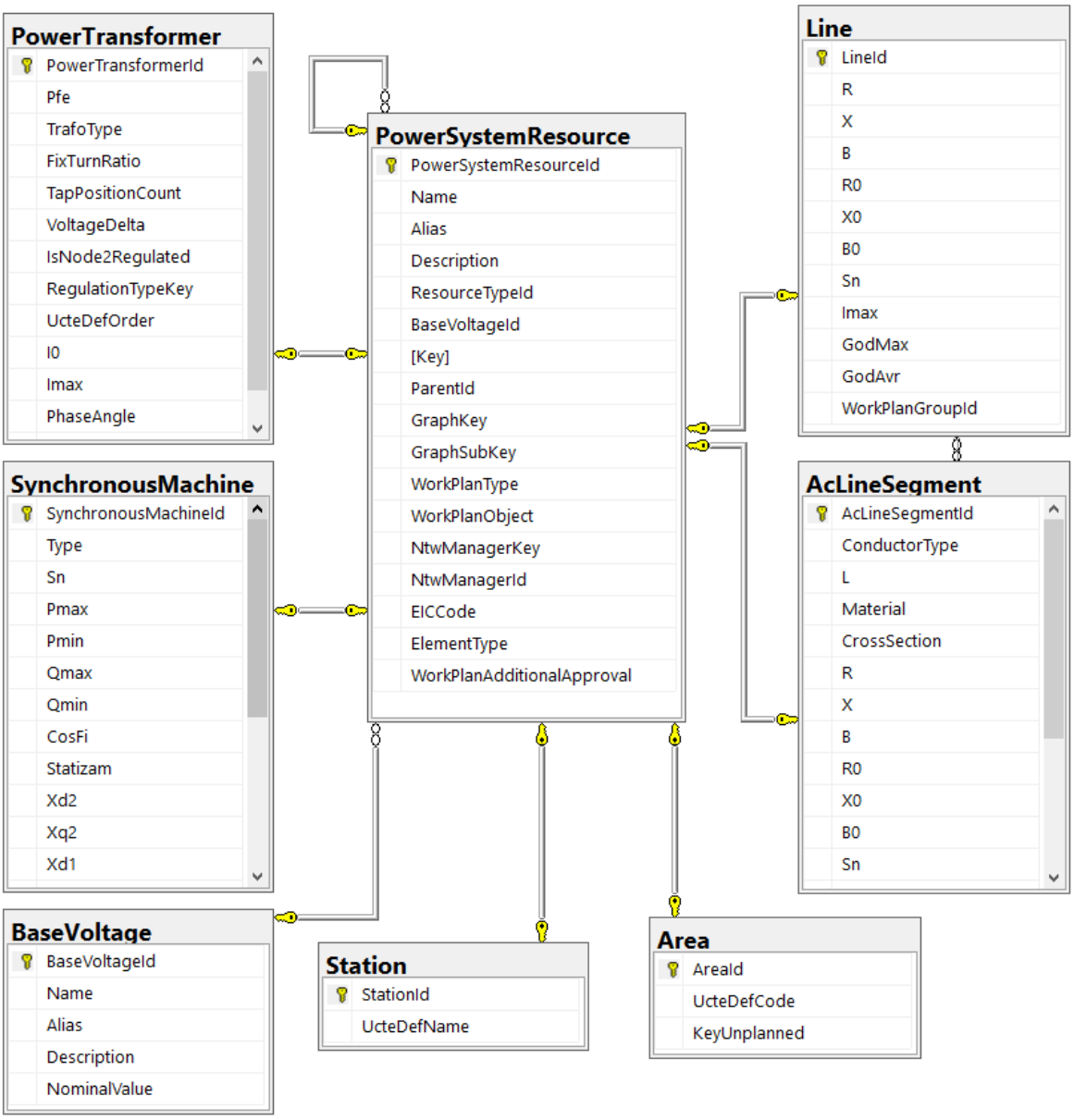


Slika 13: Elementi strukture modela podataka koji su međusobno povezani

Obje baze podataka sadrže iste tablice, razlika je što je model baze `AdnetNetworkModel` izrađen u NetVision DAM sustavu, a baze `AdnetNetworkModelABB` u SCADA Network Manager sustavu. Tablice korištene u ovome radu i njihova međusobna povezanost prikazane su na Slici 14., podaci ovih tablica se naposljetku izvoze u Excel datoteku.

Tablica `PowerSystemResource` sadrži podatke o mrežnom objektu. Preko nje su povezani svi elementi poljima `PowerSystemResourceId` i `ParentId`. Tablica `PowerSystemResource` je povezana s tablicama `PowerTransformer`, `AcLineSegment`, `Line`, `BaseVoltage` , `Station`, `Area` i `SynchronousMachine`. U svim navedenim tablicama se nalaze podaci za svaku vrstu mrežnog objekta. `Line` i

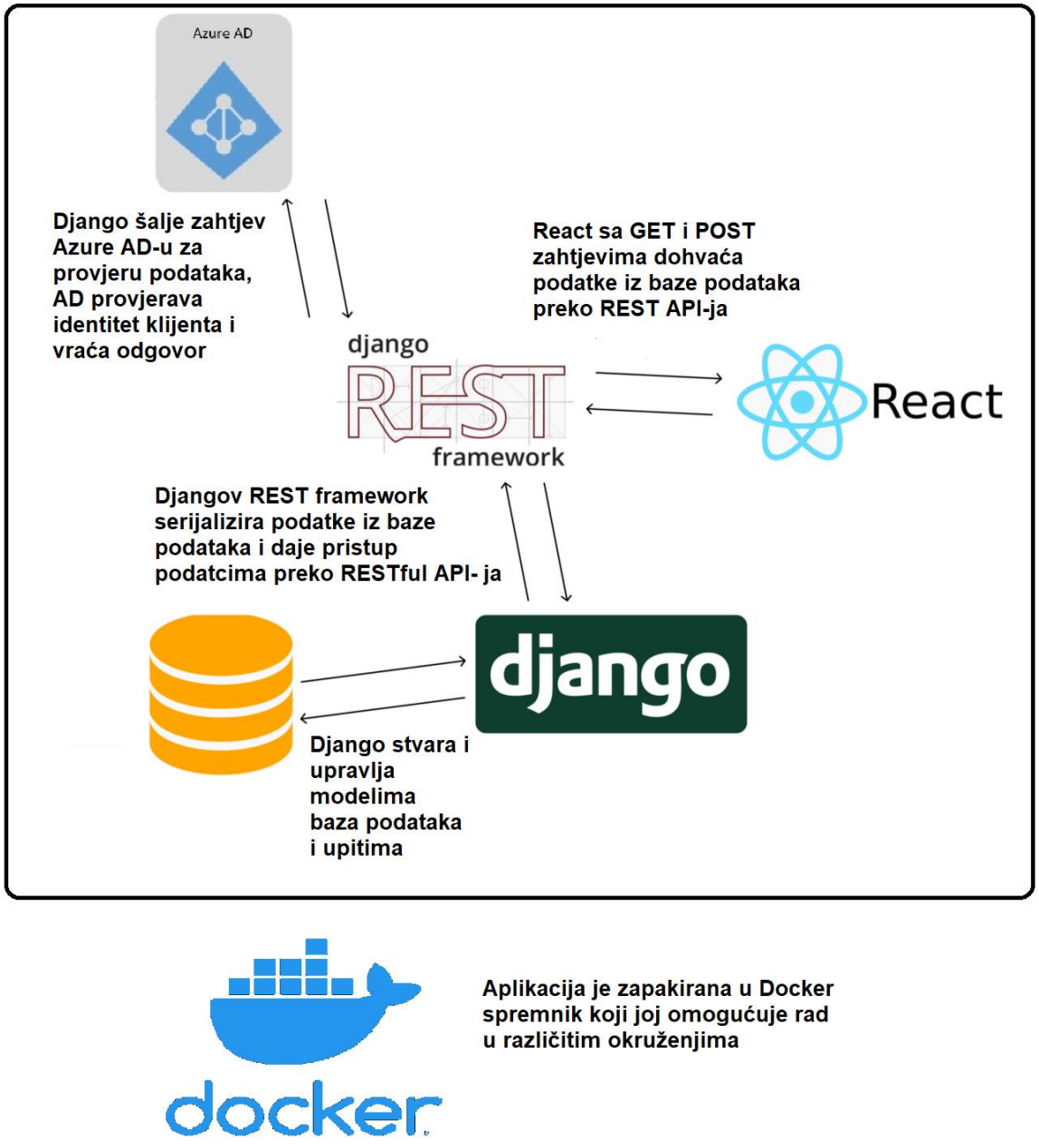
`AcLineSegment` su također povezani međusobno i zajedno opisuju generator kao vrstu mrežnog objekta. Tablice `Area` i `Station` sadrže područja gdje se nalaze mrežni objekti.



Slika 14: Relacijski model podataka

6.3. Arhitektura aplikacije

U nastavku slijedi shema na Slici 15 na kojoj je prikazan način komunikacije i međusobni odnos različitih komponenti korištenih u ovome radu:



Slika 15: Prikaz sheme načina komunikacije i međusobnog odnosa Djanga, Reacta, Active Directoryja i Dockera

Podatci su spremljeni u bazi podataka. Django stvara i upravlja modelima baza podataka i upitima prema bazi podataka. Modeli mogu biti automatski stvoreni kao model `Users`, ali

isto se odnosi i na samostalno kreirane modele. Da bi React mogao dohvatiti te podatke, treba ih serijalizirati. Podatke se uglavnom serijalizira u JSON format kako bi ih React mogao koristiti. Djangov REST Framework je proširenje Djanga koje serijalizira podatke iz baze podataka i daje pristup podatcima preko RESTful API-ja. React tada serijalizirane podatke može preko GET i POST metoda dohvatiti i koristiti, npr. sa GET metodom će iz modela *Users* dohvatiti u JSON formatu korisničko ime, lozinku, e-mail i ostale unesene podatke. Azure Active Directory služi za autentifikaciju korisnika. Klijent šalje zahtjev, a Azure Active Directory nakon provjere identiteta vraća odgovor. Docker tehnologija služi da bi se omogućio rad u različitim okruženjima. Aplikacija je u cijelosti spremljena u Docker spremnik da bi to bilo moguće.

Aplikacija se sastoji od tri glavne komponente: `api`, `frontend` i `myproject`.

`Api` je Django komponenta i u njoj su definirani URL-ovi za API pozive, jedan Model za korisnike, i šest `View` komponenti: `Popis`, `Izbor`, `Export`, `PopisABB`, `IzborABB` i `ExportABB`.

`Frontend` komponenta izrađena je u Reactu. U njoj su definirani URL-ovi za preusmjeravanje između prikaza i pet glavnih prikaza: `Homepage`, `Database`, `Adnet`, `AdnetABB` i `Endpage`.

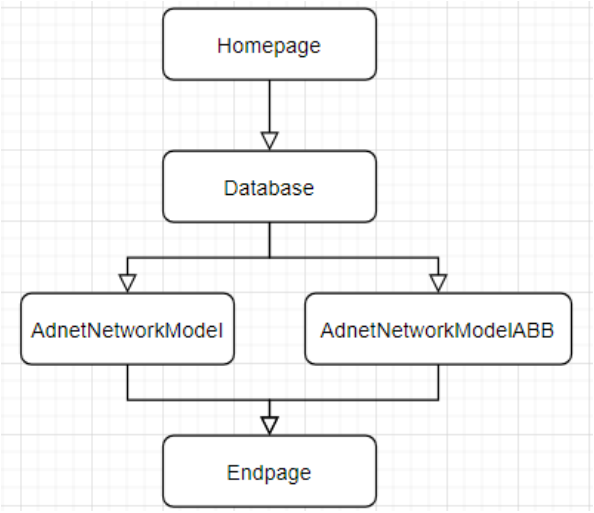
`Myproject` je Django komponenta koja sadrži konfiguracijske podatke i datoteke od kojih su najvažniji: URL-ovi prema servisima za autentifikaciju `rest-auth` i `oauth2` te aplikacijama `api` i `frontend`, kao što je prikazano u Ispisu 21, i podatci o instaliranim aplikacijama, predlošcima i bazama podataka.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('rest-auth/', include('rest_auth.urls')),
    path('oauth2/', include('django_auth_adfs.urls')),
    path('api/', include('api.urls')),
    path('', include('frontend.urls'))]
```

Ispis 21: Datoteka url.py u komponenti myproject

6.3.1. Hijerarhija prikaza

Hijerarhija prikaza prikazana je na Slici 16. Homepage je najviši sloj hijerarhije, u njemu se nalazi izbornik za prijavu. `Database` je razinu ispod, u njemu se bira jedna od dvije baze podataka. Ovisno o izboru, prikazuje se prikaz `AdnetNetworkModel` ili `AdnetNetworkModelABB`. Oba sadrže izbornike za izbor mrežnih objekata te vode na posljednji prikaz u hijerarhijskom nizu – `Endpage`. Uz `Endpage` pokreće se izvoz podataka.



Slika 16: Hijerarhija prikaza

6.4. Izrada aplikacije

U ovome poglavlju opisana je izrada aplikacije i pobliže je objašnjeno koju funkciju svaka od komponenti obavlja.

6.4.1. Prikaz Homepage

Prikaz Homepage nalazi se na vrhu hijerarhije. U `class` komponenti Homepage definirani su `state` objekti `username`, `password`, `error` i `isLoggedIn`, kao što je prikazano u Ispisu 22, te putanje za sve ostale prikaze, metoda `render` i tri funkcije: `renderHomepage`, `handleFieldInput` i `authenticate`. Funkcija `renderHomepage` poziva se unutar metode `render`, u njoj je napravljen prikaz izbornika za prijavu. Prilikom promjene unosa podataka u tekstualna polja poziva se funkcija `handleFieldInput`, kao što je prikazano u Ispisu 23.

```
export default class HomePage extends Component {
  constructor(props) {
    super(props);
    this.state = {
      username: '',
      password: '',
      error: false,
      isLoggedIn: false, };}
```

Ispis 22: `Class` komponenta Homepage

```
renderHomepage(){
      return(
          <Typography component="h4" variant="h4">
            Prijava: </Typography>
        <Grid item xs={12} align="center">
          <TextField
            label="Korisničko ime"
            variant="outlined"
            onChange={event     =>     this.handleFieldInput(event,
'username')}/>
        </Grid>
        <Grid item xs={12} align="center">
          <TextField
            label="Lozinka"
            variant="outlined"
            color="secondary"
            type="password"
            onChange={event     =>     this.handleFieldInput(event,
'password')}
            error={this.state.error}
            helperText={this.state.error ? 'Netočno korisničko ime
ili lozinka' : ''} />  </Grid>
```

Ispis 23: Funkcija `renderHomepage`/1

Drugi dio iste funkcije sadrži tri `Button` polja: prvo poziva funkciju `authenticate` za provjeru unesenih podataka, drugi preusmjerava na sljedeći prikaz, ovisno o stanju objekta `isLoggedIn`, a treće preusmjerava na stranicu za prijavu Microsofta, kao što je prikazano u Ispisu 24.

```
<Grid item xs={12} align="center">
        <Button
            color="primary"
            variant="contained"
            onClick={() => this.authenticate()}> Provjeri podatke
          </Button>
          <Button
            color="secondary"
            variant="contained"      to={this.state.isLoggedIn   ?
'/database' : '/' }
            component={Link} > Prijavi se!
          </Button>
        </Grid>
        <Grid item xs={12} align="center">
          <Button
            variant="outlined"
            color="primary">
            <MailIcon />
            <a class="linkm"
               href="https://login.microsoftonline.com/...">
               Prijava pomoću Microsoft Accounta</a>
          </Button>
        </Grid>
```

Ispis 24: Funkcija `renderHomepage`/2

Zadnji dio prikaza Homepage su dvije funkcije: `handleFieldInput` i `authenticate`, kao što je prikazano u Ispisu 25. `handleFieldInput` pri unosu u tekstualna polja korisničko ime i lozinka mijenja stanje objekata `username` i `password`, a funkcija `authenticate` POST metodom preko API-a provjerava jesu li uneseni `username` i password točni podaci i mijenja stanje varijabli `isLoggedIn` i `error`, a ovisno o njihovom

stanju aplikacija preusmjerava na novi prikaz ili vraća poruku „Netočno korisničko ime ili lozinka“.

```
handleFieldInput = (e, fieldType) => {
    const inputValue =  e.target.value;
    if(fieldType === 'username') {
      this.setState({
        username: inputValue
      })
      this.setState({error: true})
    }
    if(fieldType === 'password') {
      this.setState({
        password: inputValue
      })
    }
  }
  authenticate = () =>{
    axios.post('/rest-auth/login/' , {
      username: this.state.username,
      password: this.state.password
    })
    .then(res => {
      if(res.data.key) {
        this.setState({isLoggedIn: true})
        this.setState({error: false}) } } )
```

Ispis 25: Funkcije `handleFieldInput` i `authenticate`

6.4.2. Prikaz Database

Prikaz `Database` sadrži istoimenu klasu sa metodom `render`. Metoda `render` prikazuje izbor željene baze podataka, te nas ovisno o izboru aplikacija preusmjerava na prikaze `AdnetNetworkModel` ili `AdnetNetworkModelABB`, kao što je prikazano u Ispisu 26.

```
export default class Database extends Component {
    render(){
      return(
        <Grid container spacing={1}>
        <Grid item xs={12} align="center">
          <Typography component="h4" variant="h4">
            Odaberite bazu podataka:
          </Typography>
        </Grid>
        <Grid item xs={12} align="center">
          <FormControl component="fieldset">
            <FormHelperText>
              <div align="center">(Odabir vodi na izbornik podataka
za export!)</div>
            </FormHelperText>
            <ButtonGroup    disableElevation    variant="contained"
color="primary">
            <Button color="primary" to="/adnet" component={Link}>
              AdnetNetworkModel
            </Button>
            <Button         color="secondary"         to="/adnetabb"
component={Link}>
              AdnetNetworkModelABB
            </Button>
          </ButtonGroup>
          </FormControl>
        </Grid>
      </Grid> ); }
```

Ispis 26: `Class` komponenta `Database` sa metodom `render`

6.4.3. Prikazi AdnetNetworkModel i AdnetNetworkModelABB

Kao što je prethodno objašnjeno, `api` je Django komponenta i ona sadrži *back-end* dio rada. U aplikaciji se nalaze URL-ovi preko kojih se podaci šalju i primaju preko API-a i `View` komponente Djanga u kojima su definirana spajanja na baze podataka, kao što je prikazano u Ispisu 27, te `class` komponente (klase) `Popis`, `Izbor` i `Export`. Nakon spajanja na bazu podataka preko GET funkcija klase izvršavaju radnje poput prikupljanja

podatke, vraćanja rezultata i stanja preko API-a i na kraju zapisivanja podataka u Excel datoteku.

```
conn1=pyodbc.connect(
            "Driver={ODBC Driver 17 for SQL Server};"
            "Server=LAPTOP-5SM6HC21\MSSQLSERVER01,1434;"
            "Database=AdnetNetworkModel;"
            "Trusted_Connection=yes;")
conn2=pyodbc.connect(
            "Driver={ODBC Driver 17 for SQL Server};"
            "Server=LAPTOP-5SM6HC21\MSSQLSERVER01,1434;"
            "Database=AdnetNetworkModelABB;"
            "Trusted_Connection=yes;")
```

Ispis 27: Spajanja na baze podataka

URL-ovi su definirani u datoteci urls.py, i prikazani su u Ispisu 28.

```
urlpatterns = [
    path('popis', Popis.as_view()),
    path('popisabb', PopisABB.as_view()),
    path('get-izbor', Izbor.as_view()),
    path('get-izborabb', IzborABB.as_view()),
    path('excel', Export.as_view()),
    path('excelabb', ExportABB.as_view()),]
```

Ispis 28: URL-ovi aplikacije `api`

Klasa `Popis` iz baze dohvaća elemente kojima je `ParentId` mrežni objekt `Mreža`. To su mrežni objekti koji se prikazuju kao prvi odabir korisniku, a njihovo ime i Id se u JSON formatu vraćaju, odnosno šalju preko API-a, kao što je prikazano u Ispisu 29.

```
class Popis(APIView):
     def get(request, self):
        Q1=("SELECT      Name,      PowerSystemResourceId      FROM
PowerSystemResource  where  ParentId  =  'ef680b6e-fac6-423d-8312-
84da454d56b9'")
        cursor=conn1.cursor()
        cursor.execute(Q1)
        rows=cursor.fetchall()
        result = []
        keys = ('name','id')
        for row in rows:
            result.append(dict(zip(keys,row)))
        return Response(result, status=status.HTTP_200_OK)
```

Ispis 29: Klasa `Popis`

Klasa `Izbor` dohvaća odabrani mrežni objekt, iz baze prikuplja sljedeće objekte u hijerarhijskom nizu te ih vraća kao rezultat, kao što je prikazano u Ispisu 30.

```
class Izbor(APIView):
    lookup_url_kwarg = "mreza"
    def get (self, request):
        mreza = request.GET.get(self.lookup_url_kwarg)
        Izbor.ime = "'" + mreza + "'"
        Q2=("SELECT      Name,      PowerSystemResourceId      from
PowerSystemResource WHERE ParentId=" + Izbor.ime)
        cursor=conn1.cursor()
        cursor.execute(Q2)
        rows=cursor.fetchall()
        result = []
        keys = ('name', 'id')
        for row in rows:
            result.append(dict(zip(keys,row)))
        return Response(result, status=status.HTTP_200_OK)
```

Ispis 30: Klasa `Izbor`

Klasa `Export` dohvaća iz baze parametre odabranog mrežnog objekta, te ih zapisuje u Excel datoteku, kao što je prikazano u Ispisu 31.

```
class Export(Izbor, APIView):
    def get(self, request):
        ime = Izbor.ime
        Q3=("SELECT AcLineSegmentId, KeyUnplanned, UcteDefName,
Name, PowerSystemResourceId, Alias, BaseVoltageId, L, R, X, B, R0,
X0, B0, Sn, Imax FROM PowerSystemResource, AcLineSegment, Area,
Station where PowerSystemResourceId=" + ime)
        cursor=conn1.cursor()
        cursor.execute(Q3)
        result=cursor.fetchall()
        response = HttpResponse(content_type='text/csv')
        response['Content-Disposition'] = 'attachment;
filename="Parametri.csv"'
        writer = csv.writer(response)
        writer.writerow(['AcLineID','State','Station','Name',
'PowerSystemResourceId', 'Alias',
'BaseVoltageId','L','R','X','B','R0','X0','B0','Sn','Imax'])
        for user in result:
            writer.writerow(user)
        return response
```

Ispis 31: Klasa `Export`

Ovisno o izboru baze podataka, otvaraju se istoimeni prikazi `AdnetNetworkModel` ili `AdnetNetworkModelABB`. Prikazi sadrže: metode `componentDidMount`, `componentDidUpdate` i `render`, funkcije `mapMenuitems`, `mapSelectedChoices`, `exportEL` i `sendEl` i objekte `data` i `choice`. Metoda `componentDidMount` poziva se pri pokretanju prikaza, odnosno kad prikaz završi `mountanje` i preko API poziva dohvaća podatke metodom GET, ti su podaci početni izbor mrežnih objekata iz baze podataka, a sprema ih u objekt `data`, kao što je prikazano u Ispisu 32. Metoda `componentDidUpdate` poziva se pri ažuriranje komponente, a ona poziva funkciju `mapMenuitems`.

```
componentDidMount(){
    const endpoint = "/api/popis"
    let lookupOptions = {
      method: "GET",
      headers: {
        'Content-Type' : 'application/json' }}
    fetch(endpoint, lookupOptions)
      .then(res => res.json())
      .then(
        (result) => {
          this.setState({
            data: result
          });
  componentDidUpdate() {
    this.mapMenuitems()
  }
```

Ispis 32: Metode `componentDidMount` i `componentDidUpdate`

Metoda `render` prikazana je u dva dijela. U prvom dijelu prikazuje se padajući izbornik koji korisniku na odabir daje mrežne objekte i lista za pregled odabranih objekata, kao što je prikazano u Ispisu 33.

```
render() {
    return (
      <Grid container spacing={1}>
        <div style={{display: 'block'}}>
          <Grid item xs={12} align="center">
            <Typography component="h4" variant="h4">
              Odaberite podatke iz baze AdnetNetworkModel za export:
            </Typography>
          </Grid>
          <Grid container spacing={5}>
            <Grid item xs={12} align="center">
            <List style={{float: 'right', width:300, height:400}}>
              <ListItem>
                <ListItemText primary={<h3>Odabrano:</h3>}/>
              </ListItem>
              {this.mapSelectedChoices()}
            </List>
            {this.state.data.length    >    0    ?    <FormControl
style={{minWidth: 400,margin:10}}>
              <InputLabel      id="demo-simple-select-label">Mrežni
objekti:</InputLabel>
              <Select
                labelId="demo-simple-select-label"
                id="demo-simple-select"
               onChange={(event)=>this.sendEl(event.target.value)}
                value={this.state.choice[0]} >
              {this.mapMenuitems()}
              </Select>
              </FormControl> : null}
```

Ispis 33: Metoda `render`/1

Pozivaju se funkcije `mapMenuitems`, `mapSelectedChoices`, `sendEl` i. `mapMenuitems` prikazuje imena prethodno dohvaćenih elemenata mreže i prikazuje ih u padajućem izborniku, `mapSelectedChoices` odabrane elemente sprema u listu, a `sendEl` API pozivom šalje Id odabranog elementa i sprema rezultate u objekte `data` i `choice` (Ispis 34).

U data se sprema popis elemenata koji se prikazuju u padajućem izborniku, a u `choice` odabrani element na osnovu kojega se dohvaća taj popis.

```
mapMenuitems = () => {
      return this.state.data.map((element, index) => {
        const value = {
          id: element.id,
          name: element.name }
          return (
            <MenuItem value={value}>{element.name}</MenuItem>)})}
  mapSelectedChoices = () => {
    return this.state.choice.map((element, index) => {
        return (
          <ListItem>
            <ListItemIcon>
              <SendIcon />
            </ListItemIcon>
            <ListItemText                    primary={element.name}
secondary={"Odabir: " + (index+1)}/>
          </ListItem>)})}
  sendEl = (mreza) => {
    const endpoint = `/api/get-izbor?mreza=${mreza.id}`
    let lookupOptions = {
      method: "GET",
      headers: {
        'Content-Type' : 'application/json'}
    }
    fetch(endpoint, lookupOptions)
      .then(res => res.json())
      .then(
        (result) => {
          const selected = this.state.choice.slice();
          selected.push(mreza);
          this.setState({
            data: result,
            choice: selected });
```

Ispis 34: Funkcije `mapMenuitems`, `mapSelectedChoices`, `sendEl`

Drugi dio metode `render` sadrži dvije `Button` komponente, prva poziva funkciju `exportEl`, a druga ponovno pokreće, odnosno osvježava prikaz, kao što je prikazano u Ispisu 35.

```
<Grid item xs={12} align="center">
  <Button
    style={{margin:5}}
    variant="contained"
    color="secondary"
    onClick={() => this.exportEL()} >
    <a class="link" href="api/excel">Izvezi</a>
  </Button>
  <Button
    variant="contained"
    color="primary"
    onClick={() => window.location.reload(true)} >
    Reset
  </Button>
</Grid>
```

Ispis 35: Metoda `render`/2

Funkcija `exportEl` preusmjerava na zadnji prikaz aplikacije i poziva funkciju za izvoz parametara u Excel datoteci, kao što je prikazano u Ispisu 36.

```
exportEL = () =>{
    this.props.history.push('/endPage')
    const endpoint = '/api/excel'
      let lookupOptions = {
        method: "GET",
        headers: {
          'Content-Type' : 'application/json'}}
      fetch(endpoint, lookupOptions)}
```

Ispis 36: Funkcija `exportEl`

6.4.4. Prikaz Endpage

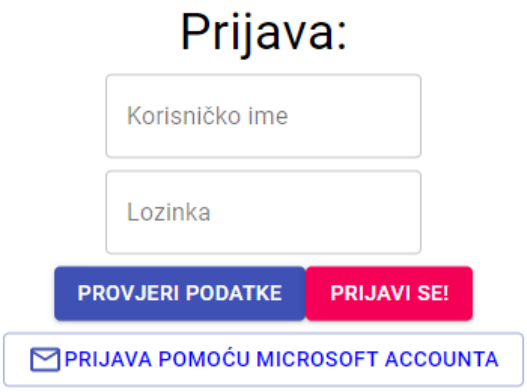
Prikaz `Endpage` sadrži istoimenu klasu sa metodom `render`. Metoda `render` prikazuje poruku „Podatci su uspješno izvezeni u excel datoteku!“ i nudi povratak na izbornik baze podataka, odnosno prikaz `Database`, kao što je prikazano u Ispisu 37.

```
export default class Endpage extends Component {
  render() {
    return (
      <Grid container spacing={1}>
        <Grid item xs={12} align="center">
          <Typography variant="h4" component="h4">
            Podatci su uspješno izvezeni u excel datoteku!
          </Typography>
        </Grid>
        <Grid
            item xs={12} align="center">
          <Button
            variant="contained"
            color="secondary"
            to="/database"
            component={Link}>
            Natrag na početni izbornik
          </Button>
        </Grid>
      </Grid>);}}
```

Ispis 37: `Class` komponenta `Endpage` sa metodom `render`

6.5. Primjer korištenja

Pokretanjem aplikacije otvara se izbornik za prijavu. Prijava u aplikaciju može se izvršiti na dva načina: upisivanjem točne kombinacije korisničkog imena i lozinke ili prijavom pomoću Microsoft računa (engl. *account*). (Slika 17)

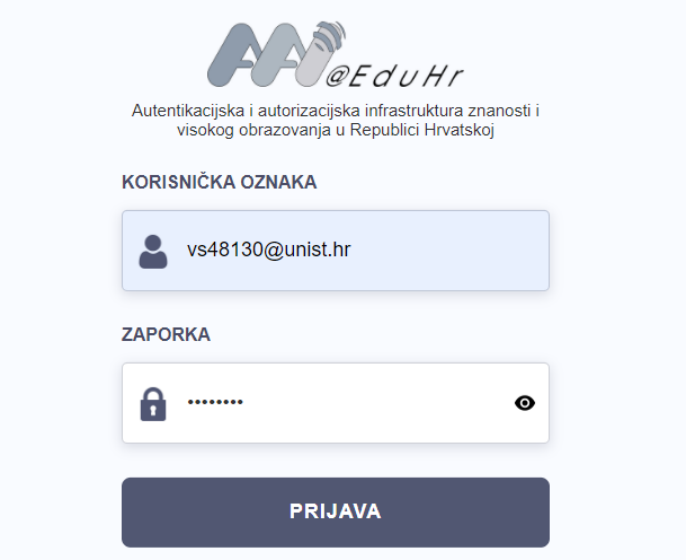


Slika 17: Zaslon za prijavu

Klikom na gumb „Provjeri podatke“ provjerava se jeli unesena kombinacija korisničkog imena i lozinke točna, ukoliko nije program nas na to upozorava. Ukoliko su podaci točni, klikom na gumb „Prijavi se!“ aplikacija preusmjerava na sljedeći prikaz.

Za prijavu preko Microsoft računa, klikom na gumb „Prijava pomoću Microsoft accounta“ aplikacija preusmjerava na stranicu za prijavu tvrtke Microsoft.

Zatim sustav preusmjerava na stranice prijave organizacije s kojom je račun registriran u Microsoftovom Active Directory sustavu, u ovome slučaju to je AAI@EduHr sustav, kao što je prethodno objašnjeno u poglavlju „5.2.“. Stranica za prijavu prikazana je na Slici 18.

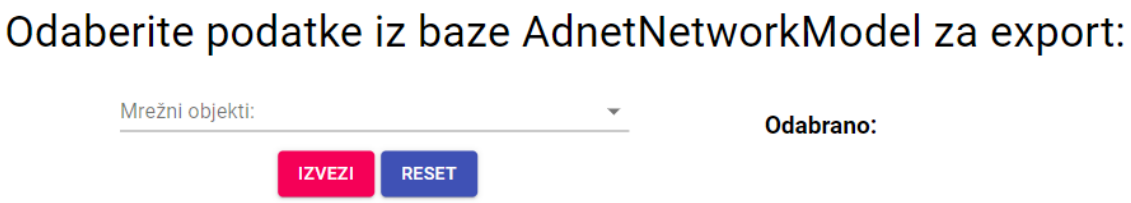


Slika 18: Stranica za prijavu AAI@EduHr sustava

Po uspješnoj prijavi, aplikacija nas preusmjerava na sljedeći prikaz aplikacije, izbornik baze podataka. (Slika 19)

Slika 19: Prikaz izbornika baze podataka

Odabirom odgovarajuće baze podataka, otvara se novi prikaz aplikacije, izbornik mrežnih objekata modela mreže EES-a. (Slika 20)



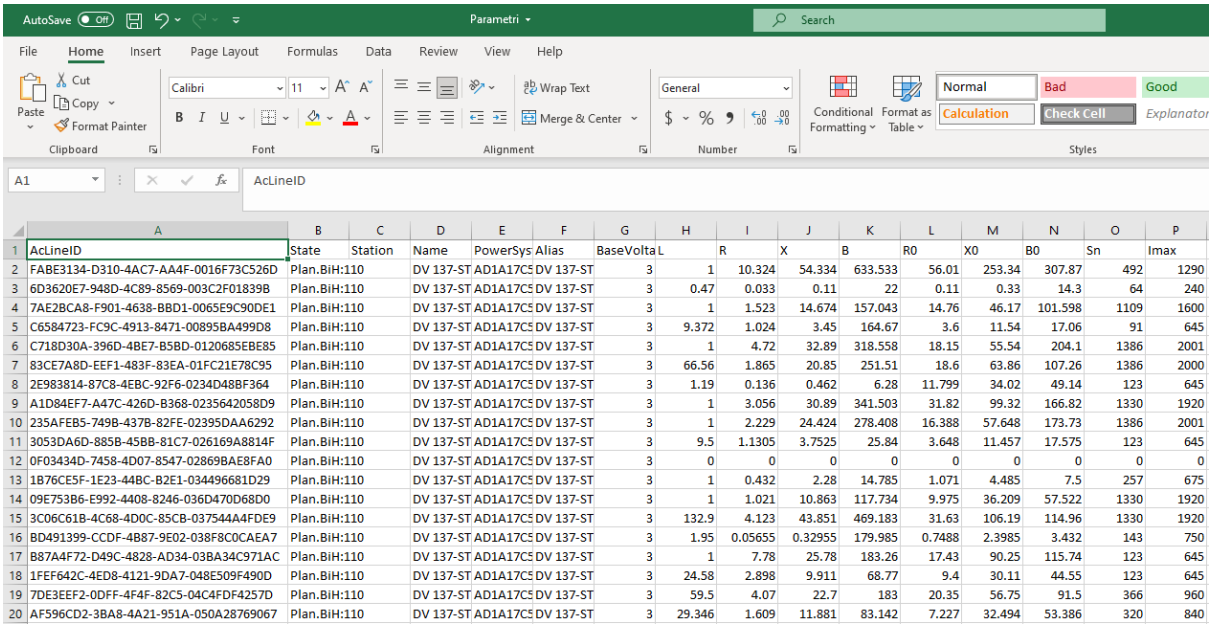
Slika 20: Prikaz izbornika parametara mreže

Sa lijeve strane nalazi se padajući izbornik iz kojeg redom biramo objekte, a nestaje kada dođemo do posljednjeg objekta u hijerarhijskom nizu. Sa desne strane nalazi se lista odabranih objekata, kao što je prikazano na Slici 21.

Slika 21: Lista odabranih mrežnih objekata

Zatim, klikom na gumb „Reset“ izbornik se osvježi i sa biranjem mrežnih objekata se može ponovo započeti, a klikom na gumb „Izvezi“ aplikacija preusmjerava na posljednji prikaz te se uz njega otvara izbornik željene lokacije za spremanje Excel datoteke sa parametrima odabranog mrežnog objekta.

Klikom na gumb „Natrag na početni izbornik“ aplikacija preusmjerava na izbornik baze podataka, a izgled izvezenih parametara u Excel datoteci prikazan je na Slici 22.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	AcLineID	State	Station	Name	PowerSys	Alias	BaseVolta	L	R	X	B	R0	X0	B0	Sn	Imax
2	FABE3134-D310-4AC7-AA4F-0016F73C526D	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	10.324	54.334	633.533	56.01	253.34	307.87	492	1290
3	6D3620E7-948D-4C89-8569-003C2F01839B	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	0.47	0.033	0.11	22	0.11	0.33	14.3	64	240
4	7AE2BCA8-F901-4638-BBD1-0065E9C90DE1	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	1.523	14.674	157.043	14.76	46.17	101.598	1109	1600
5	C6584723-FC9C-4913-8471-00895BA499D8	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	9.372	1.024	3.45	164.67	3.6	11.54	17.06	91	645
6	C718D30A-396D-4BE7-B5BD-0120685EBE85	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	4.72	32.89	318.558	18.15	55.54	204.1	1386	2001
7	83CE7A8D-EEF1-483F-83EA-01FC21E78C95	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	66.56	1.865	20.85	251.51	18.6	63.86	107.26	1386	2000
8	2E983814-87C8-4EBC-92F6-0234D48BF364	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1.19	0.136	0.462	6.28	11.799	34.02	49.14	123	645
9	A1D84EF7-A47C-426D-B368-0235642058D9	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	3.056	30.89	341.503	31.82	99.32	166.82	1330	1920
10	235AFEB5-749B-437B-82FE-02395DAA6292	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	2.229	24.424	278.408	16.388	57.648	173.73	1386	2001
11	3053DA6D-885B-45BB-81C7-026169A8814F	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	9.5	1.1305	3.7525	25.84	3.648	11.457	17.575	123	645
12	0F03434D-7458-4D07-8547-02869BAE8FA0	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	0	0	0	0	0	0	0	0	0
13	1B76CE5F-1E23-44BC-B2E1-034496681D29	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	0.432	2.28	14.785	1.071	4.485	7.5	257	675
14	09E753B6-E992-4408-8246-036D470D68D0	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	1.021	10.863	117.734	9.975	36.209	57.522	1330	1920
15	3C06C61B-4C68-4D0C-85CB-037544A4FDE9	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	132.9	4.123	43.851	469.183	31.63	106.19	114.96	1330	1920
16	BD491399-CCDF-4B87-9E02-038F8C0CAEA7	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1.95	0.05655	0.32955	179.985	0.7488	2.3985	3.432	143	750
17	B87A4F72-D49C-4828-AD34-03BA34C971AC	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	1	7.78	25.78	183.26	17.43	90.25	115.74	123	645
18	1FEF642C-4ED8-4121-9DA7-048E509F490D	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	24.58	2.898	9.911	68.77	9.4	30.11	44.55	123	645
19	7DE3EEF2-0DFF-4F4F-82C5-04C4FDF4257D	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	59.5	4.07	22.7	183	20.35	56.75	91.5	366	960
20	AF596CD2-3BA8-4A21-951A-050A28769067	Plan.BiH:110		DV 137-ST	AD1A17C5	DV 137-ST	3	29.346	1.609	11.881	83.142	7.227	32.494	53.386	320	840

Slika 22: Izvezeni parametri modela mreže EES-a u Excel datoteci

7. Zaključak

Informacijske tehnologije donijele su promjene u načinu odvijanja i pogleda na današnji svijet te poslovne procese koji se u njemu odvijaju. Transformacijom poslovnih procesa putem integracije informacijskih tehnologija u njihov rad olakšava se i ubrzava njihova izvedba te postižu bolji rezultati.

U ovom diplomskom radu težilo se postići upravo to, potpunu web aplikaciju koja bi svojim jednostavnim korištenjem olakšala rad korisnicima, iako je namijenjena krajnjim zaposlenicima i klijentima koji imaju potrebu korištenja podataka iz mreže EES-a Hrvatske. Tema je inspirirana trenutnim načinom odvijanja jednog od poslovnih procesa u tvrtki HOPS. Dohvaćanje podataka odvija se ručno slanjem SQL upita i to predstavlja problem svima koji nemaju potrebno znanje i/ili nisu upoznati sa hijerarhijom baza podataka. Ideja aplikacije je riješiti taj problem. U radu je opisano kako aplikacije ovisno o izboru korisnika dohvaća aktualne elemente EES-a Hrvatske iz postojećih baza podataka i omogućuje izvoz željenih podataka u tablični format.

Django se pokazao kao najbolji odabir za izradu upravo ovakve aplikacije zbog njegovih višestruko iskoristivih komponenti, brzog razvoja, jednostavnog spajanja i rada sa bazama podataka, te jednostavne komunikacije sa React bibliotekom preko API poziva. React je korišten za izradu korisničkog sučelja i njegovih komponenti, a pokazao se kao prikladan izbor radi brzine ažuriranja i dinamike promjena koje se odvijaju u nizu raščlanjivanja strukture stabla pri izboru ponuđenih parametara (npr. nakon izbora grada prikazuju se elektrane i stanica tog područja). Spremanje aplikacije u Docker spremnik omogućilo je rad u različitim okruženjima, a spajanje na Active Directory sustav dodijeljena su prava pristupa za rad korisnicima nad aplikacijom.

Literatura

[1] Holovaty, Kaplan-Moss, „The Definitive Guide to Django“, 2008.

[2] „Django - The web framework for perfectionists with deadlines“, https://www.djangoproject.com/ (posjećeno 1.8.2021.)

[3] Holovaty, Kaplan-Moss, „The Django Book“, 2013.

[4] „Announcing the Django Software Foundation“, https://www. djangoproject. com/ weblog/2008/jun/17/foundation/ (posjećeno 1.8.2021.)

[5] „Complete Django History“, https://www.geeksforgeeks.org/complete-django-history-python/ (posjećeno 2.8.2021.)

[6] „Design Philosophies“, https://docs.djangoproject.com/en/3.2/misc/design-philosophies (posjećeno 2.8.2021.)

[7] „Django Views – 6 Simple Steps to Create View Component for Django Project“, https://data-flair.training/blogs/django-Views/, (posjećeno 2.8.2021.)

[8] „Django Models – Learn to Create Your First Django Model“, https://data-flair.training/blogs/django-models/, (posjećeno 3.8.2021.)

[9] „Django URLs and URLConf – Learn to Map URL Patterns to View Function in 10 Minutes“, https://docs.djangoproject.com/en/3.2/topics/http/urls/, (posjećeno 18.8.2021.)

[10] „Django REST Framework Tutorial“, https://data-flair.training/blogs/django-rest-framework/, (posjećeno 3.8.2021.)

[11] „The Hisory of React.js on a Timeline“, https://blog.risingstack.com/thehistory-of-react-js-on-a-timeline/ (posjećeno 04.08.2021.)

[12] „XHP: A New Way to Write PHP“, https://www.facebook.com/notes/facebookengineering/xhp-a-new-way-to-write-php/294003943919/ (posjećeno 20.08.2021.)

[13] „Why did we build React?“, https://reactjs.org/blog/2013/06/05/whyreact.html, (posjećeno 05.08.2021.)

[14] „React.Component“, https://reactjs.org/docs/react-component.html (posjećeno 05.08.2021.)

[15] „IZRADA EDUKATIVNE IGRE S KARTAMA „FlipFight"“, Ivan Zaharija, 2020.

[16] „React Reconciliation?“, https://medium.com/@ryanbas21/reactreconciliation-7075e3f07437 (posjećeno 07.08.2021.)

[17] „React lifecycle methods digram“, http://projects.wojtekmaj.pl/react-lifecyclemethods-diagram/ (posjećeno 10.08.2021.)

[18] "Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud", https://web.archive.org/web/20190913100835/http://maureenogara.sys-con.com/node/2747331 (posjećeno 11.08.2021.)

[19] "8 surprising facts about real Docker adoption", https://www.datadoghq.com/docker-adoption/ (posjećeno 12.08.2021.)

[20] „Docker image“ , https://searchitoperations.techtarget.com/definition/Docker-image, (posjećeno 12.08.2021.)

[21] „Docker Hub“, https://www.docker.com/products/docker-hub (posjećeno 12.08.2021.)

[22] „Directory System Agent", https://docs.microsoft.com/en-us/windows/ win32/ ad/ directory-system-agent?redirectedfrom=MSDN (posjećeno 13.08.2021.)

[23] Hynes, Byron, "The Future of Windows: Directory Services in Windows Server Longhorn“, 2019.

[24] „Azure Active Directory (Azure AD)“, https://azure.microsoft.com/en-us/services/ active-directory/#overView (posjećeno 15.08.2021.)