Programski sustav za korištenje prototipa generatora modela IT sustava

Bezuk, Dora

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 277

PROGRAMSKI SUSTAV ZA KORIŠTENJE PROTOTIPA GENERATORA MODELA IT SUSTAVA

Dora Bezuk

Zagreb, veljača 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 277

PROGRAMSKI SUSTAV ZA KORIŠTENJE PROTOTIPA GENERATORA MODELA IT SUSTAVA

Dora Bezuk

Zagreb, veljača 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 2. listopada 2023.

DIPLOMSKI ZADATAK br. 277

Pristupnica:	**Dora Bezuk (0036503309)**
Studij:	Računarstvo
Profil:	Programsko inženjerstvo i informacijski sustavi
Mentor:	izv. prof. dr. sc. Stjepan Groš

Zadatak: **Programski sustav za korištenje prototipa generatora modela IT sustava**

Opis zadatka:

Za potrebe pripreme vježbi i eksperimenata u kibernetičkoj sigurnosti korisno je imati sustav za generiranje modela IT sustava na proizvoljnoj razini apstrakcije. Sastavni dio modela IT sustava su i sigurnosne politike. U ovom trenutku postoji prototip alata koji gradi modele IT sustava za opisanu svrhu, no za korištenje tog prototipa potrebna je značajna količina predznanja i vještine. U sklopu diplomskoga rada potrebno je razviti sustav koji će omogućiti udaljeni pristup prototipu za generiranje modela IT sustava putem programskih sučelja. Sustav treba omogućiti upravljanje svim aspektima prototipa. Pozornost obratiti i na upravljanje pristupom kako bi različiti korisnici mogli upotrebljavati sustav bez mogućnosti pregleda tuđih podatka. Programsko rješenje mora biti modularno i razvijeno u skladu s dobrim praksama. Radu priložiti izvorni kôd. Citirati korištenu literaturu i navesti dobivenu pomoć.

Rok za predaju rada: 9. veljače 2024.

SADRŽAJ

UVOD

U doba koje se više nego ikada prije oslanja na digitalnu infrastrukturu i njene sustave, potreba za dubinskim razumijevanjem svakog aspekta sustava te infrastrukture, ključna je, ne samo za operativnu učinkovitost sustava, već i za zaštitu istih od sigurnosnih prijetnji kojima su izloženi.

Složeni Informacijsko-tehnološki sustavi (ITS), koji integriraju različite hardverske, softverske i mrežne komponente, zahtijevaju određeni stupanj povjerljivosti budući da javno poznavanje specifičnosti sustava može izložiti ranjivosti sustava potencijalnim napadačima. S druge strane, tajnost istovremeno ograničava sposobnost sigurnosnih stručnjaka da uče iz najvjerodostojnijeg izvora informacija - stvarnih sustava. Kako bi se ublažio jaz koji nastaje ovim paradoksom u upravljanju informacijama sustava, polje kibernetičke sigurnosti često se oslanja na simulirane modele ITS-a, koji nude kontrolirano okruženje za učenje i razvoj sigurnosnih strategija bez ugrožavanja stvarne sigurnosti sustava. Unatoč njihovoj važnosti, proces stvaranja točnih i sveobuhvatnih modela IT sustava prepun je izazova. Tradicionalno, alati dostupni u tu svrhu bili su složeni, zahtijevali su veliku stručnost i često bili ograničeni u smislu fleksibilnosti i prilagodljivosti.

U cilju smanjenja razlike u pristupačnosti i upotrebljivosti, razvijen je prototip alata specifično kreiranog za stvaranje modela IT sustava na bilo kojoj razini apstrakcije. Postojanje prototipa alata koji konstruira takve modele nudi značajan potencijal za istraživanje i razvoj u području kibernetičke sigurnosti. Međutim, korištenje ovog prototipa trenutno zahtijeva znatnu količinu prethodnog znanja i specijaliziranih vještina, ograničavajući njegovu dostupnost i praktičnu primjenu.

Prepoznajući izazov, ovaj rad fokusira se na razvoj programskog sustava koji olakšava udaljeni pristup takvom prototipu putem programskih sučelja. Stvaranjem sustava koji nudi pojednostavljeno programsko sučelje za komunikaciju s prototipom, ovaj rad nastoji premostiti jaz između sofisticiranih mogućnosti prototipa i njegove praktične upotrebljivosti te ima potencijal doprinijeti području kibernetičke sigurnosti. U okviru ovog radu će se razviti sustav, dokumentirati proces te procijeniti učinkovitost implementacije. U prvom dijelu rada, dan je teorijski pregled prototipa generatora modela IT

sustava i udaljenog pristupa te pregled programskih sučelja. U drugom dijelu rada, dani su zahtjevi na sustav, dok se u trećem dijelu daje kratki pregled korištenih tehnologija te arhitektura sustava. Nakon toga, u četvrtom dijelu rada, opisana je implementacija sustava i dobiveni rezultati. Zatim slijedi kratka diskusija dobivenih rezultata, a potom se daje kratki pregled u potencijalna poboljšanja postojećeg sustava u budućem radu. Zaključno, kratko se sumira sve što je napravljeno.

1. Teorijski pregled

Informacijsko-tehnološki sustavi (ITS) su međusobno povezane komponente koje rade zajedno na prikupljanju, obradi, pohranjivanju i širenju informacija za podršku odlučivanju, koordinaciji, kontroli, analizi i vizualizaciji unutar organizacije. Navedeni sustavi obuhvaćaju širok raspon funkcionalnosti, od jednostavnih alata poput elektroničke pošte za komunikaciju do složenih sustava za planiranje resursa poduzeća koji integriraju sve aspekte poslovanja organizacije. [1]

No, ITS-ovi ne odnose samo na tehnologiju, već o osnovi za pružanje rješenja za poslovne probleme i prilike. Naglašavaju važnost prvo razumijevanja poslovnih potreba, a zatim dizajniranja tehnoloških rješenja za zadovoljenje tih potreba. [2]

Upravo zbog kompleksnosti i raširenosti ITS-a, razumijevanje i poznavanje sustava složen je, ali iznimno važan zadatak, osobito u području kibernetičke sigurnosti koja nastoji svakodnevno štiti navedene sustave. Precizni modeli sustava ključni su za prepoznavanje ranjivosti, optimiziranje performansi i osiguravanje robusnog dizajna sustava. Međutim, generiranje takvih preciznih modela kada oni nisu dostupni, vrlo je izazovno.

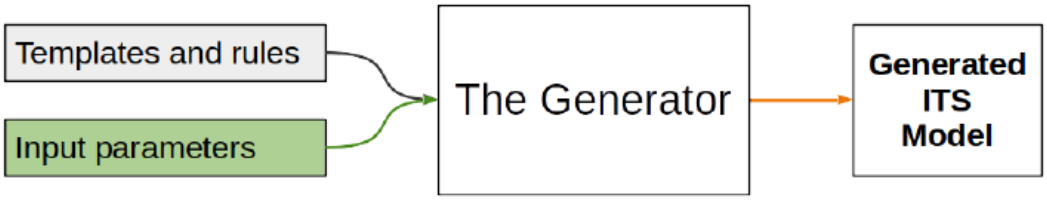
Trenutne metodologije za generiranje ITS modela uključuju manualan rad, automatizirane alate i hibridne pristupe, svaki s određenim ograničenjima. Ručno modeliranje je zahtjevno i podložno ljudskim pogreškama, postojećim automatiziranim alatima često nedostaje fleksibilnost za složene sustave, a hibridne metode, iako su korisne, ne ispunjavaju u potpunosti zahtjeve skalabilnosti i prilagodljivosti različitih IT okruženja.

U nastojanju da se riješe spomenuti izazovi, nastao je prototip alata za izgradnju modela IT sustava na bilo kojoj razini apstrakcije. Spomenuti model predstavljen u radu "Automatsko generiranje modela IT sustava" [3].

1.1. Prototip generatora modela IT sustava

Cilj alata Generator je automatizirati proces stvaranja modela ITS-a koji su korisni u različitim scenarijima kada su potrebne detaljne informacije o ITS-u organizacije, ali nisu odmah dostupne. To uključuje simulirana okruženja za obuku i testiranje kibernetičke sigurnosti i aplikacije strojnog učenja gdje razumijevanje ITS-a može pomoći automatizirati strategije napada i obrane.

Generator, čiji je model prikazan na slici 1.1, zahtijeva specifične i detaljne ulaze za stvaranje ITS modela. Ulazi uključuju organizacijske parametre kao što su broj zaposlenih ili industrija kojoj organizacija pripada, kao i veličina sustava, mrežne konfiguracije, sigurnosni protokoli i drugi parametri. Takvi detalji pomažu Generatoru da razumije opseg i prirodu ITS-a koji treba modelirati. Nakon primanja ulaznih podataka, generator koristi niz sofisticiranih algoritama i metodologija za tumačenje podataka i izradu ITS modela. Ovaj proces može uključivati simulaciju ponašanja mreže, predviđanje interakcija između komponenti i prepoznavanje potencijalnih sigurnosnih ranjivosti. Zamršeni proces osigurava da rezultirajući model nije samo detaljan nego i realan, odražavajući složenost stvarnih IT sustava. Konačno, kulminacija ovog procesa je skup izlaza koji pružaju sveobuhvatan prikaz ITS-a. Rezultati često uključuju detaljne mrežne dijagrame, specifikacije sustava i analitička izvješća. Oni dalje služe kao vrijedni alati za profesionalce koji žele razumjeti, optimizirati ili osigurati svoju IT infrastrukturu, pružajući detaljan uvid u strukturu i ponašanje sustava.[3]

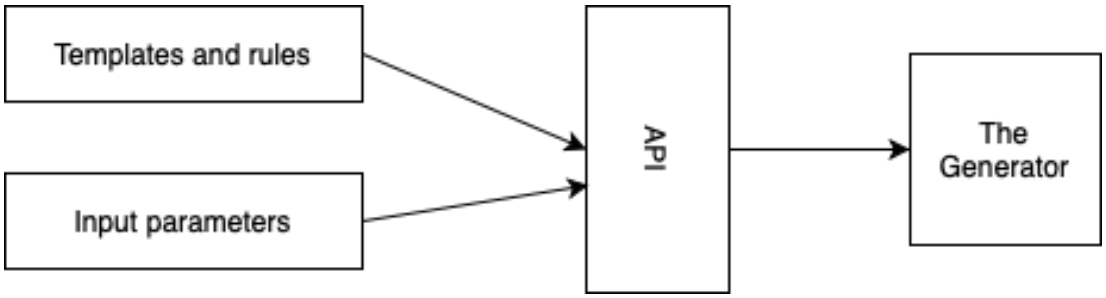


Slika 1.1: Model Generatora s ulazima i izlazima [3]

1.2. Udaljeni pristup i programska sučelja

Usprkos sofisticiranim mogućnostima generatora prototipa, postoji potreba za pospješenjem pristupačnosti i korištenju Generatorovih mogućnosti, posebno za korisnike na udaljenim lokacijama ili one bez opsežnog tehničkog znanja.

Kako bi se poboljšala pristupačnost, ovaj rad predlaže razvoj programskih sučelja, skupa definiranih pravila koji različitim softverskim aplikacijama omogućuju međusobnu komunikaciju.[4] Programska sučelja, vidljiva na slici 1.2, djeluju kao posrednik između korisnika i Generatora, pojednostavljujući proces interakcije i poboljšavajući time pristupačnost, fleksibilnost te efikasnost. Pružaju siguran, standardiziran način razmjene podataka. Navedeno je posebno važno za generator ITS modela, gdje korisnici trebaju unijeti specifične parametre i dohvatiti generirane modele. Korištenje API-ja može olakšati navedene radnje na daljinu, a da korisnik ne treba izravan pristup temeljnom sustavu, čime se pojednostavljuje proces interakcije. Korisnici tako mogu unositi podatke, inicirati generiranje modela i primati izlazne podatke putem pristupačnijeg, pojednostavljenog sučelja. Osim toga, korištenje programskih sučelja otvara mogućnosti za integraciju Generatora s drugim alatima i platformama, pospješujući njegovu primjenjivost u različitim područjima. Takvo korištenje sustava, potencijalno može dovesti do šire upotrebe i, posljedično, napretka u razumijevanju i sigurnosti IT sustava. [5]



Slika 1.2: Inetgracija programskog sučelja u Generator

U kontekstu Generatora, korištenje programskih sučelja nudi niz prednosti. Udaljeni pristup i automatizacija omogućuju sigurno i udaljeno pristupanje Generatoru, dopuštajući korisnicima da specificiraju ulazne parametre, odaberu predloške i postave pravila bez izravnog pristup Generatoru. Standardiziranjem komunikacije pruža se jedinstven i pojednostavljen način komunikacije. Time se uklanja potreba za razumijevanjem složenih unutarnjih mehanizama prototipa Generatora, pa se korisnici mogu koncentrirati na iskorištavanje sustava za vlastite potrebe. Sučelja nude sigurnost i kontrolu pristupa tako da osiguravaju kontroliranu pristupnu točku Generatoru, pritom vodeći računa da samo korisnici s ovlaštenjem mogu generirati modele. To je posebno važno s obzirom na osjetljivu prirodu informacija i konfiguracija uključenih u ITS modele. Sustav koji koristi API-je lakše je skalirati i integrirati s drugim sustavima i aplikacijama. Novi alati i usluge mogu se jednostavnije dodati ili izmijeniti bez prepravljanja cijelog sustava. Konkretno, ovakav sustav može biti upotrijebljen za unos generiranih ITS modela u alate za kibernetičku sigurnost, simulacijska okruženja ili u druge alate za dizajn i planiranje. Fleksibilnost i prilagodljivost sučelja nudi mogućnost prilagodbe specifičnim uvjetima klijentske tehnologije. Mogu se odabrati specifične funkcije ili podaci koji se žele koristiti, a sve to bez potrebe za dubokim tehničkim znanjem o radu prototipa Generatora. Zaključno, sve navedeno dovodi do bržeg razvoja i inovacija jer se pri razvoju mogu iskoristiti postojeći API-ji i dodati nove funkcionalnosti, čime se potiču inovacije i kontinuirano radi na poboljšanju.[5]

Zaključno, korištenje programskih sučelja je pristup koji je adekvatan za potrebe stvaranja efikasnog, pristupačnog i fleksibilnog sustava za udaljeni pristup prototipu generatora modela IT sustava. Ovakav pristup je, ne samo tehnički izvediv, već i korisnički orijentiran, te omogućava korisnicima lako i učinkovito upravljanje svim aspektima prototipa.

2. Zahtjevi na sustav

Kako bi implementirali sustav za korištenje prototipa Generatora putem programskih sučelja, potrebno je proučiti ulaze Generatora definirane u radu [3] koji će biti osnova za konstruiranje modela sustava. Također, potrebno je detaljnije razmotriti specifične funkcionalne zahtjeve i specifikacije potrebne za implementaciju ovakvog sustava.

2.1. Zahtjevi Generatora

Ulazi Generatora dijele se na: Predloške i pravila (engl. *Templates and rules*) i Ulazne parametre (engl. *Input parameters*). Unutar Generatora, Predlošci i pravila te Ulazni parametri imaju različite funkcije i svrhu.

Predlošci su unaprijed definirane strukture koje opisuju uloge zaposlenika (engl. *Employee roles*) i organizacijske usluge (engl. *Organizational services*), kao i programske pakete. Neki od atributa koje sadrže su ovisnosti, pružene usluge, hardverski zahtjevi i troškovi.

Pravila služe za instanciranje skupa podataka (Podatkovna pravila (engl. *Dataset instantiation rules*)) i segmentaciju mreže (Mrežna pravila (engl. *Network segmentation rules*)) kod generiranja sustava. Podatkovna pravila podržavaju tri načina rada (engl. *mode*) u kojima definiraju načine povezivanja podataka s drugim objektima. Načini rada su: Zaposlenički (engl. *Employee mode*), Servisni (engl. *Service mode*) i Softverski (engl. *Software mode*). S druge strane, Mrežna pravila određuju zahtjeve koji upravljaju odabirom mrežnih segmenata za instance skupa podataka i/ili softverske instalacije. Podržanih vrsta pravila (eng. *types of rules*) ima nekoliko, npr. Različite mreže za skupove, Jednake mreže za skupove i mnogi drugi (engl. *RequireDistinctNetworksForSets, RequireSameNetworksForSets, RequireCommonNetworksForSets, LimitAllowedProtectionLevelRange..*).

Parametri su specifične, često numeričke vrijednosti ili definirane karakteristike koje pružaju potrebne detalje za prilagodbu modela IT sustava. Oni uključuju informa-

cije poput broja zaposlenika u svakoj podskupini, potrebne posebne zbirke podataka, vanjske usluge koje organizacija pruža i prilagođena mrežna pravila.[3]

Predlošci su općenitiji i mogu se primijeniti na različite organizacije, dok su parametri prilagođeni posebnim potrebama jedne organizacije. Na temelju navedenih ulaza Generatora, opisanih u radu [3], napravljen je detaljniji pregled atributa organizacijskog modela koji će imati sustav, a čije je razumijevanje važno za njegovu detaljniju izgradnju.

Atributi organizacijskog modela:

- *Kolekcije podataka (engl. Data collections)* - lista skupova podataka koji trebaju biti postavljeni u IT sustavu pojedine organizacije, određuje kako i gdje se ti skupovi podataka trebaju instancirati u sustavu. Svaki skup podataka ima definirane atribute u skladu s Definicijom Podatka (engl. *Data Definition*). Proces kreiranja i organiziranja skupova podataka u sustavu je kontroliran skupom unaprijed definiranih Podatkovnih pravila u odgovarajućem Načinu rada.
- *Mrežne politike (engl. Network policies)* - popis mrežnih pravila o sigurnosnim politikama po kojima se obavlja segmentacija mreže.
- *Računalne politike (engl. Computer policies)* - popisuju dodatne politike koje su, što se tiče samog unosa, identične mrežnim politikama, ali se u vanjskom alatu ne koriste za segmentaciju mreže, već za razdvajanje softvera i podataka na veći broj računala.
- *Pružene vanjske usluge (engl. Provided external services)* - popisuju vanjske usluge opisane u predlošcima koje organizacija nudi putem Interneta.
- *Grupe zaposlenika (engl. Employee groups)* - rječnik koji opisuje skupine zaposlenika. Po definiranom popisu vrsta zaposlenika daje popis grupa vrsta zaposlenika i za svaku vrstu popis skupina takvih zaposlenika u organizaciji.
- *Kombinirane uloge zaposlenika (engl. Composite employee roles)* - rječnik koji definira nove uloge zaposlenika kao ključeve i daje popis postojećih korisničkih uloga koje se kombiniraju.
- *Predlošci komponenti (engl. Component templates)* - opisuju složenije komponente koje je potrebno ugraditi u IT sustav organizacije.

2.2. Funkcionalni zahtjevi

Definiranje funkcionalnih zahtjeva prvi je korak u procesu izgradnje sustava za korištenje prototipa Generatora putem programskih sučelja. Funkcionalni zahtjevi izrazito su važni kod razvoja sustava jer osiguravaju da je sustav usklađen s potrebama korisnika i očekivanjima dionika. Oni djeluju kao nacrt, definirajući što sustav mora raditi i kako mora raditi. [6] Primarni cilj ovog sustava je pojednostaviti proces unosa, upravljanja i korištenja osnovnih zahtjeva za organizaciju IT sustava. Ima ključnu ulogu u izradi i rukovanju ulaznih JSON-a, koji su sastavni dio ostalih komponenti unutar sustava, osiguravajući kohezivan i učinkovit tijek rada.

Kratki pregled zahtjeva slijedi:

1. *Unos i organizacija zahtjeva* - sustav mora omogućavati unos i rukovanje zahtjevima za organizaciju IT sustava i na temelju njih izgraditi ulazni JSON.
2. *Upravljanje unosima* - sustav mora omogućavati upravljanje unosima tako da:
   (a) *k*orisnici moraju moći pohraniti unesene unose i dodijeliti im imena za jednostavniju evidenciju.
   (b) *k*orisnici moraju moći vidjeti, kopirati i uređivati pohranjene unose.
3. *Integracija s API-jem za izgradnju modela IT sustava* - sustav mora slati unose i pohranjivati modele izgrađene putem API-ja za izgradnju modela IT sustava.
4. *Interakcija s modelima IT sustava* - korisnici moraju moći komunicirati s izgrađenim modelima IT sustava, uključujući pregled, stvaranje novih modela i uređivanje postojećih.
5. *Grafičko uređivanje modela IT sustava** - korisnici moraju moći grafički učitati i uređivati ključne detalje modela IT sustava.
6. *Upravljanje uređenim modelima IT sustava* - korisnici moraju moći pohranjivati, pregledavati i uređivati više verzija uređenog modela IT sustava.
7. *Stvaranje i pohranjivanje konfiguracije* - sustav bi trebao olakšati stvaranje konfiguracija za ciljno simulacijsko okruženje i pohraniti rezultirajuće konfiguracije.
8. *Interakcija s pohranjenim konfiguracijama* - korisnik bi trebao moći pregledavati ranije generirane konfiguracije za pojedini uređeni model IT sustava.

* Zahtjevi vezani uz grafičke prikaze i manipulacije, prelaze opseg ovog rada.

2.3. Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi ključni su za osiguranje upotrebljivosti, pouzdanosti i učinkovitosti sustava. Za razliku od funkcionalnih zahtjeva koji definiraju specifično ponašanje ili funkcije, nefunkcionalni zahtjevi usredotočeni su na to da sustav to čini na način koji je usklađen s očekivanjima korisnika i industrijskim standardima. Oni postavljaju standarde za to kako sustav obavlja određene zadatke, a ne koje zadatke obavlja. [7]

Kratki pregled nefunkcionalnih zahtjeva slijedi:

1. *Sigurnost - Kontrola pristupa* - sustav treba omogućiti pristup samo autentificiranim korisnicima.

2. *Privatnost i izolacija korisničkih podataka* - korisnici mogu imati pristup samo datotekama i informacijama koje su sami unijeli u sustav i koje su vraćene iz sustava na temelju prethodno unesenih podataka.

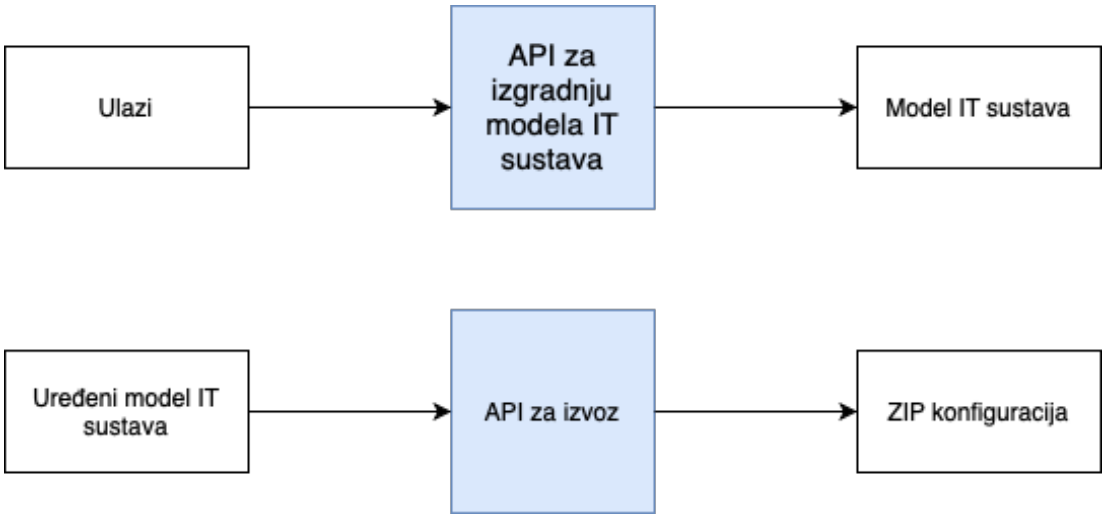
3. *Validacija i rukovanje pogreškama* - sustav treba uključivati mehanizme provjere valjanosti unosa od strane korisnika, pri čemu treba pružiti jasne, informativne poruke o pogrešci ili povratne informacije kada se naprave netočni unosi, usmjeravajući korisnike da ih isprave.

3. Korištene tehnologije

Za konkretniju implementaciju sustava za korištenje prototipa Generatora putem programskih sučelja, potrebno je osmisliti i izraditi arhitekturu sustava te odabrati prikladne tehnologije i prikladnu bazu podataka.

3.1. Arhitektura sustava

Na temelju tehničkih zahtjeva sustava, napravljena je vizualizacija protoka podataka kroz sustav i ključni entiteti s kojima korisničko sučelje i sustav radi, vidljiva na slici 3.1. Korisnici komuniciraju sa sustavom putem korisničkog sučelja, unoseći podatke za izradu ili uređivanje modela IT sustava (tzv. ulazi). Korisničko sučelje komunicira sa sustavom putem odgovarajućih API-ja slanjem podatak i primanjem podataka. Sustav komunicira s vanjskim API-jem za izgradnju modela IT sustava i drugim vanjskim API-jem za izvoz, u cilju dobivanja modela IT sustava i konfiguracije te podatke pohranjuje u bazu podataka. API za izvoz uzima uređene modele i pretvara ih u formate prikladne za simulacijska okruženja, a zatim ih korisniku prikazuje kroz korisničko sučelje. Baza podataka djeluje kao središnje spremište, čuvajući sve modele, izmjene i konfiguracijske podatke.



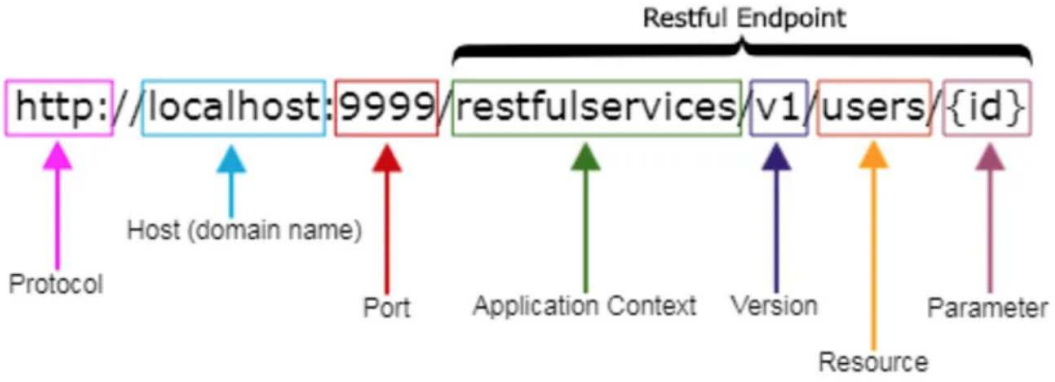
Slika 3.1: Ključni entiteti s kojima radi sustav

S obzirom na zahtjeve i prirodu sustava, koji uključuje sve već navedeno, odabran je REST arhitekturalni stil za izgradnju API-ja i upravljanje interakcijama sustava.

REST je arhitektonski stil za distribuirane sustave. REST nije standard ili protokol, već skup arhitektonskih ograničenja koja se primjenjuju kao cjelina. [8] Korištenje REST arhitekture nudi mnogo prednosti. Jedna od prednosti je jednostavnost i lakoća korištenja. REST se temelji na standardnim HTTP metodama, što ga čini lakim za razumijevanje i implementaciju. Ne zahtijeva opsežne alate ili biblioteke, što olakšava brži razvoj i integraciju.[8] REST je bez stanja (engl. *stateless*), što znači da su u svakom zahtjevu klijenta prema poslužitelju sadržane sve informacije potrebne za razumijevanje i obradu zahtjeva. To pojednostavljuje dizajn poslužitelja i poboljšava skalabilnost. [9] Mogućnost predmemoriranja (engl. *cacheability*) omogućuje da se odgovori mogu definirati kao predmemorirani ili ne. Predmemoriranje može eliminirati potrebu za nekim interakcijama klijent-poslužitelj, poboljšavajući učinkovitost i performanse sustava.[10] REST-ova opcija slojevite arhitekture sustava omogućava da između klijenta koji zahtijeva prikaz stanja resursa i poslužitelja koji šalje odgovor može postojati nekoliko poslužitelja, a klijent ne zna koliko je takvih slojeva, ima li ih uopće te tko odgovara na njegov zahtjev. To omogućuje uravnoteženje opterećenja i pruža način za neovisno proširenje sustava. [8] REST koristi jedinstveno sučelje između komponenti, pojednostavljujući arhitekturu i povećavajući vidljivost interakcija. Ova ujednačenost omogućuje odvajanje klijenta i poslužitelja, što zauzvrat omogućuje neovisno razvijanje. [8]

Pri radu s REST API-jem koriste se njegove četiri glavne komponente: krajnja točka URL-a (engl. *URL endpoint*), HTTP glagol (engl. *HTTP verb*), tijelo (eng. *Body*) i HTTP statusni kodovi (engl. *HTTP status codes*). Krajnja točka URL-a pred-

stavlja resurs, HTTP glagol specificira radnju, a tijelo sadrži detalje poruke. REST sustav obično komunicira preko protokola HTTP s raznim glagolima . Neki od glagola su: GET, POST, PUT i DELETE, koji odgovaraju operacijama stvaranja, čitanja, ažuriranja i brisanja (skr. CRUD). Resursi, kao što su podaci i funkcionalnost, identificirani su s pomoću URI-ja, a resursima se manipulira s pomoću ovih standardnih operacija. Po primitku HTTP zahtjeva, poslužitelj će vratiti odgovarajući HTTP statusni kod zajedno sa svim dodatnim JSON korisnim sadržajima. [11] Izgled REST krajnje točke, vidljiv na slici 3.2, primjer je dobre prakse u kreiranju krajnjih točaka, u kojem se kod složenijih sustava prate verzije servisa unutar kojih se kreiraju parametrizirani resursi.

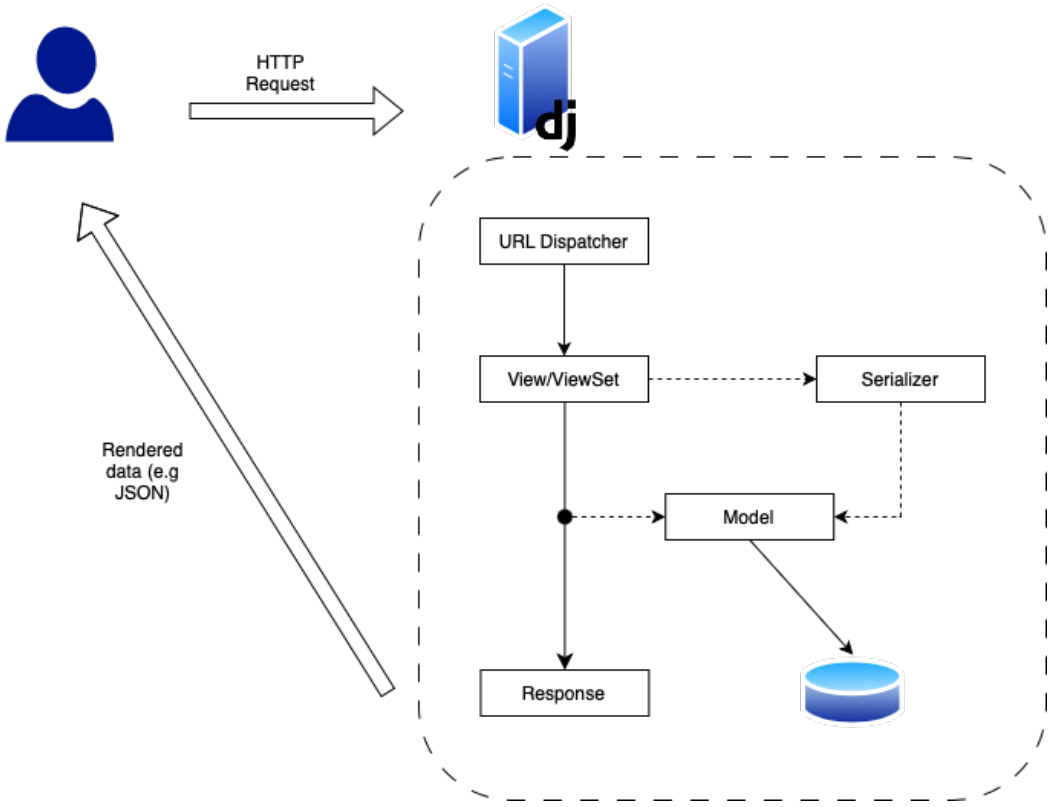


Slika 3.2: Izgled REST krajnje točke s naglašenim dijelovima [12]

3.2. Backend tehnologija

Za potrebe izgradnje sustava, konkretno backend tehnologije, koristit će se programski okvir Django. Django je Python programski okvir (engl. *web framework*), poznat po njegovom pristupu "baterije uključene" (engl. *batteries-included*), dizajniranom da omogući brz razvoj i čist te pragmatičan dizajn. [13] Django je poznat po svojoj skalabilnosti i potiče brzi razvoj koristeći princip "ne ponavljaj se" (engl. *DRY*) kojim se olakšava razvojni proces i smanjuje vrijeme kodiranja. [13] Sigurnost je također prednost Djanga; pruža robusne mehanizme za upravljanje korisničkim računima i lozinkama i ima ugrađenu zaštitu od mnogih uobičajenih sigurnosnih prijetnji. [14] Djangova zajednica je velika i aktivna, nudi obilje resursa, uključujući sveobuhvatnu dokumentaciju, forume, pakete trećih strana (engl. *third-party libraries*). Okvir je nevjerojatno svestran te se koristi za širok raspon aplikacija, a ako opsežna ugrađena funkcionalnost ne zadovoljava specifične potrebe, jednostavno se mogu uključiti paketi trećih strana ili izraditi vlastite komponente. Ranije spomenuta odabrana arhitektura je REST. Django nudi alat za izradu API-ja koji se zove Django Rest Framework. Dizajniran je za pružanje jednostavnog, intuitivnog i bržeg razvoja REST API-ja, s karakteristikama serijalizacije, autentifikacije i prilagodljivih pogleda. Iskorištava osnovne mogućnosti Djanga, dodajući sloj koji je posebno prilagođen razvoju API-ja, poboljšavajući funkcionalnost Djanga s posebnim fokusom na konstrukciju API-ja. [15]

Tradicionalno, Django koristi MVT arhitekturu, a DRF zamjenjuje korištenje predložaka (engl. *Template*) sa sustavom serijalizatora za izlaz podataka u raznim formatima, npr. JSON ili XML. Protok podataka u DRF, vidljiv na slici 3.3, započinje tako da klijent pošalje HTTP zahtjev API-ju.



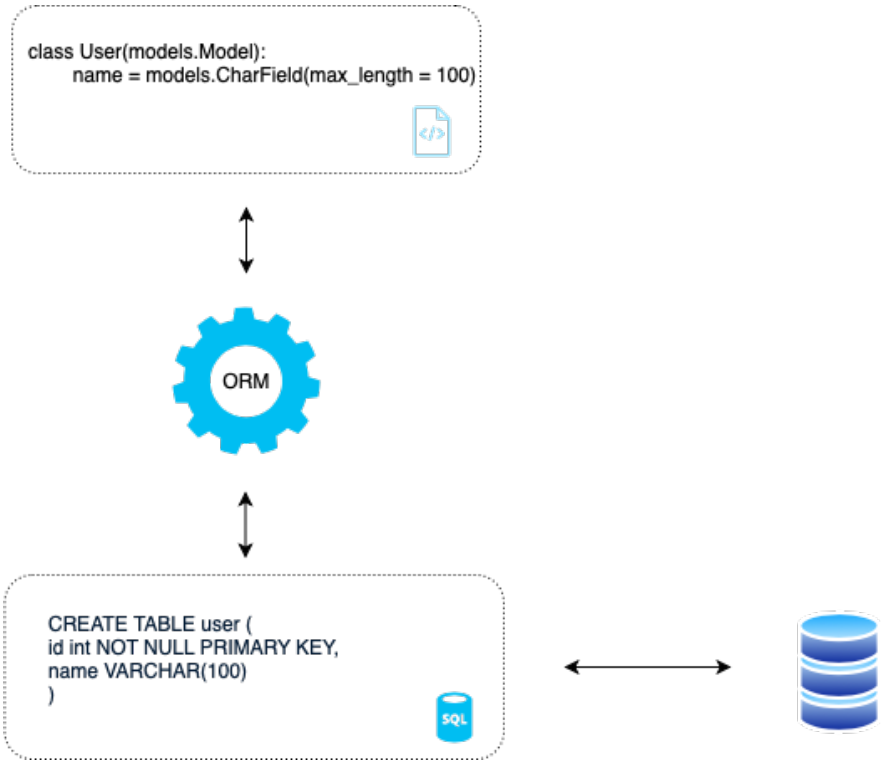
Slika 3.3: DRF protok podataka

Zahtjev dolazi do Django poslužitelja i obrađuje ga URL dispečer (urls.py). URL dispečer koristi URL uzorke definirane u aplikaciji kako bi odredio koji pogled (engl. *view*) treba obraditi zahtjev. Zahtjev se zatim prosljeđuje odgovarajućem pogledu, gdje se implementira logika aplikacije. Pogled je u interakciji sa serijalizatorom, čiji je zadatak transformirati složene tipove podataka (npr. instance Django modela) u Python tip podatka koji se zatim može lako prikazati u JSON-u ili XML-u. Ako je glagol u zahtjevu POST, PUT ili PATCH, serijalizator će također validirati dolazne podatke. Ako zahtjev uključuje dohvaćanje podataka ili manipulaciju, pogled ili serijalizator će komunicirati s modelom kroz Djangov ORM. ORM prevodi Python kod visoke razine u SQL upite i šalje ih u bazu podataka. Baza podataka obrađuje upit i vraća odgovarajuće podatke natrag u ORM. Jednom kada pogled ima podatke (bilo iz baze podataka ili izravno iz zahtjeva), koristi se serijalizator da ih transformira u odgovarajući format. Pogled zatim stvara HTTP odgovor sa serijaliziranim podacima. Na kraju se odgovor šalje nazad klijentu. Klijent prima podatke u traženom formatu, najčešće JSON, i zatim ih može obraditi i prikazati prema potrebi.

3.3. Odabir baze

Kao bazu podataka sustava, koristit će se SQLite. SQLite je bibllioteka koja nudi sistem za upravljanje relacijskim bazama podataka. SQLite je jedinstven jer mu nije potreban poslužitelj (engl. *serverless*), bez konfiguracije je (engl.*zero-configuration*) i samostalan. Upravo zato je vrlo popularan izbor za aplikacije kojima je potrebna lagana baza podataka. Od verzije 3.9.0 iz 2015. godine, SQLite uključuje ugrađeno JSON proširenje koje mu omogućuje izvršavanje složenih upita prema JSON nizovima. S obzirom da sustav treba moći pohranjivati JSON zapise, takva karakteristika je vrlo važna. [16] Jedna od prednosti korištenja SQLitea je Djangova ugrađena podrška za SQLite, koji je njegova zadana baza podataka i pri tome ne zahtijeva posebnu dodatnu konfiguraciju.

Jedna od najistaknutijih značajki Djanga je njegov ORM, koji programerima omogućuje interakciju s bazom podataka, vidljivo na slici 3.4, putem Python koda umjesto pisanja SQL naredbi, čime se poboljšava čitljivost, lakoća održavanja i sigurnost. [17] Također, ovakva mogućnost znatno ubrzava razvojni proces. ORM sustav besprijekorno radi sa SQLiteom.



Slika 3.4: Prikaz ORM preslikavanja u bazu podataka

4. Implementacija i rezultati

U ovom poglavlju, pregledat će se implementacijski detalji i rezultati glavnih funkcionalnosti sustava.

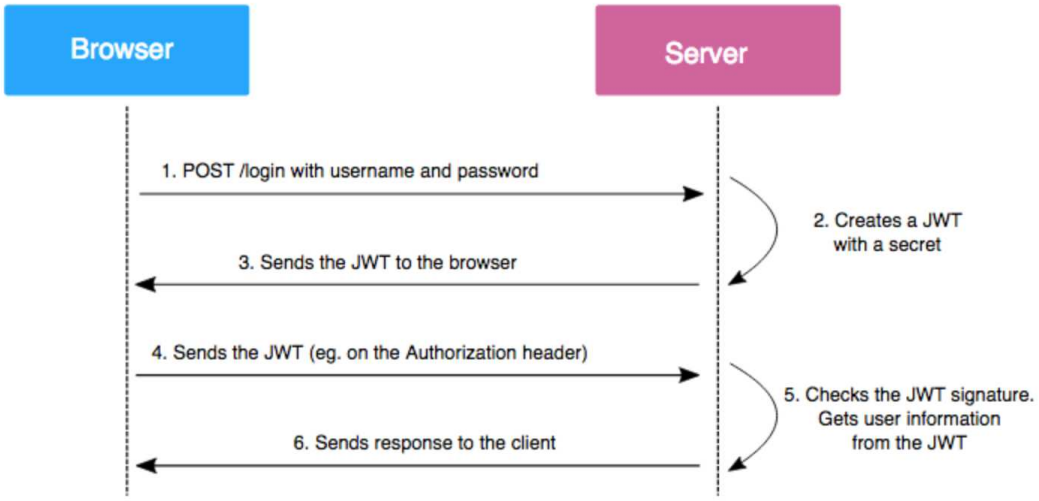
4.1. Upravljanje korisnicima i sigurnost

Kako bi korisnicima omogućili sigurnu registraciju i prijavljivanje u sustav, potrebno je implementirati sustav autentifikacije i autorizacije. U navedenu svrhu, koristit će se JWT. U sustavu koji koristi JWT, vidljivo na slici 4.1, pristupni token (engl. *access token*) korisniku daje privremeni pristup resursima i mora biti kratkog vijeka trajanja da bi se smanjili sigurnosni rizici. Token za osvježavanje (engl. *refresh token*), s druge strane, ima dulji životni vijek od pristupnog, te služi za dobivanje novog pristupnog tokena kada trenutni istekne.

```
SIMPLE_JWT = {
        'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
        'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
}
```

Ispis 4.1: Podešavanje trajanja JWT tokena unutar datoteke settings.py

Ovo odvajanje povećava sigurnost: pristupni token koristi se često i stoga ima veći rizik od izlaganja, dok je rjeđe korišteni token za osvježavanje bolje zaštićen i može sigurno ponovno uspostaviti sesiju korisnika, smanjujući potrebu za čestim prijavama i održavajući ravnotežu između fleksibilnosti i sigurnosti. [18]



Slika 4.1: Protok JWT tokena kroz sustav [18]

U sustavu, za implementaciju JWT tokena koriste se dvije različite biblioteke: "dj-rest-auth"[19] i "rest-framework-simplejwt" [20]. Isprva se može učiniti suvišnim, ali korištenjem dviju biblioteka omogućuje se stvaranje robusnijeg, sigurnijeg i značajkama bogatijeg sustava provjere autentičnosti. Biblioteka "dj-rest-auth" prilagođenija je za općenite zadatke provjere autentičnosti dok je "rest-framework-simplejwt" bolja za naprednije JWT upravljanje. Konkretno, za potrebe izgradnje ovog sustava, preferirano je rukovanje tokenom za osvježavanjem i vraćanjem oba tokena pri prijavljivanju u sustav kako to radi "rest-framework-simplejwt", dok je za upravljanje registriranjem, odjavom i promjenom lozinke jednostavnije korištenje "dj-rest-auth" biblioteke, vidljivo u kodu. Zajedno, ove biblioteke kombiniraju jednostavnost korištenja sa snažnim sigurnosnim praksama. Ovaj modularni pristup prilično je uobičajen u Djangovom ekosustavu, gdje se različite aplikacije i paketi često kombiniraju kako bi se okvir prilagodio specifičnim potrebama nekog projekta.

```
path('rest-auth/registration/', RegisterView.as_view(), name='rest_register'),
path('rest-auth/login/', TokenObtainPairView.as_view(), name='rest_login'),
path('rest-auth/logout/', LogoutView.as_view(), name='rest_logout'),
path('rest-auth/password/change/', PasswordChangeView.as_view(), name='
    rest_password_change'),
path('rest-auth/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'
    ),
path('rest-auth/user/', CustomUserDetailsView.as_view(), name='user-detail'),
```

Ispis 4.2: Krajnje korisničke točke unutar datoteke urls.py

Kako bi se smanjila kompleksnost korisničkih profila i omogućilo postavljanje samo osnovnih informacija o korisniku, kreiran je poseban serijalizator za korisnike vidljiv u kodu.

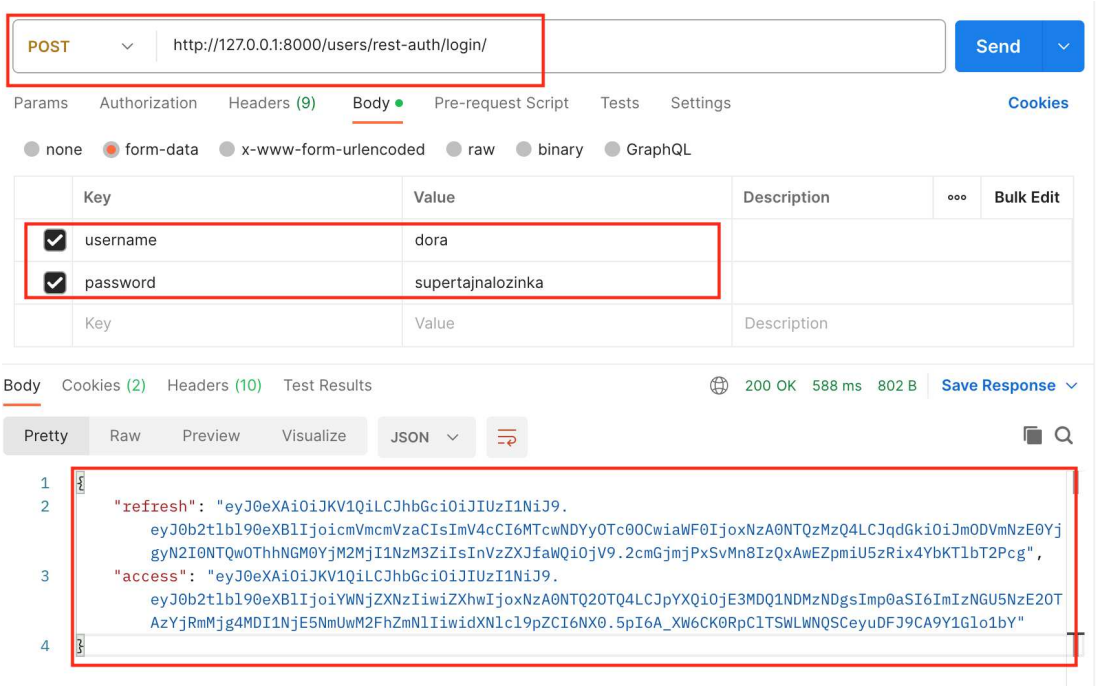
```
class CustomUserDetailsSerializer(UserDetailsSerializer):
    class Meta:
    model = get_user_model()
    fields = ('username', 'email', 'first_name', 'last_name', 'password')

    def update(self, instance, validated_data):
    # Update email if provided
    if validated_data.get('email'):
    instance.email = validated_data['email']

    return super().update(instance, validated_data)
```

Ispis 4.3: Poseban serijalizator korisnika unutar datoteke serializers.py

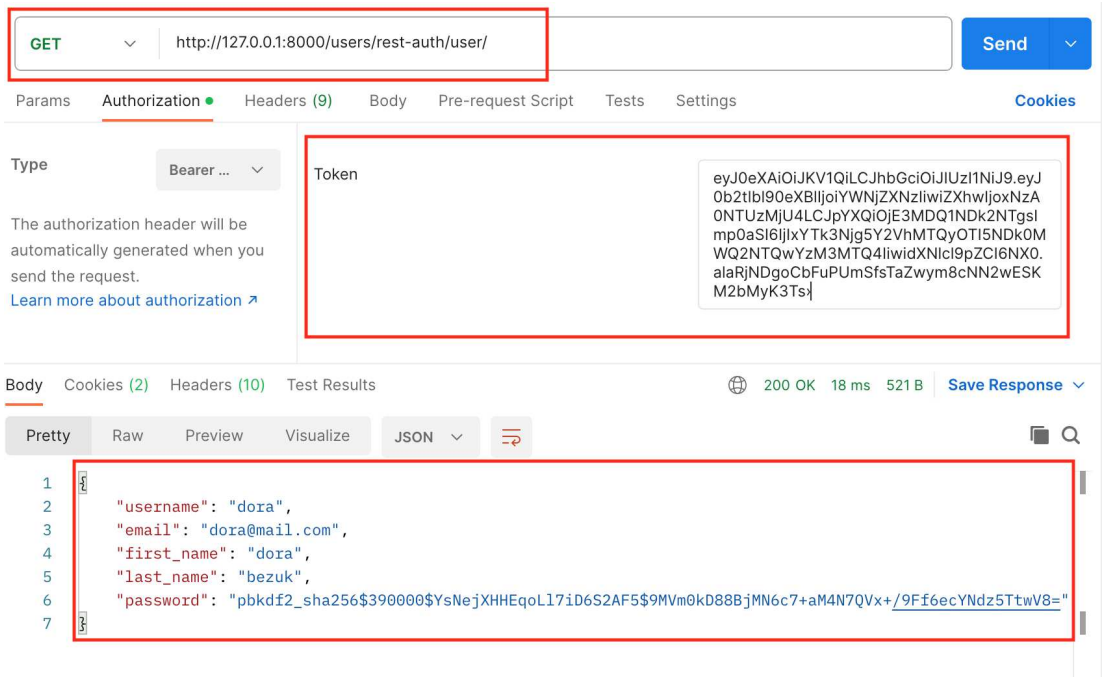
Nakon svih navedenih postavki, u primjerima, možemo vidjeti kako pozivi i odgovori izgledaju u alatu Postman[21]. Prijavljivanjem u sustav, vidljivo na slici 4.2, generiraju se oba tokena.



Slika 4.2: Generiranje tokena za osvježavanje i pristupnog tokena prilikom prijave

Dohvaćanje podataka o korisniku sa specificiranjem Bearer tokena u autorizacijskom zaglavlju, vidljivo na slici 4.3. Ako token istekne ili se uopće ne pošalje s zah-

tjevom, dobiva se poruka "401 Unathorized".

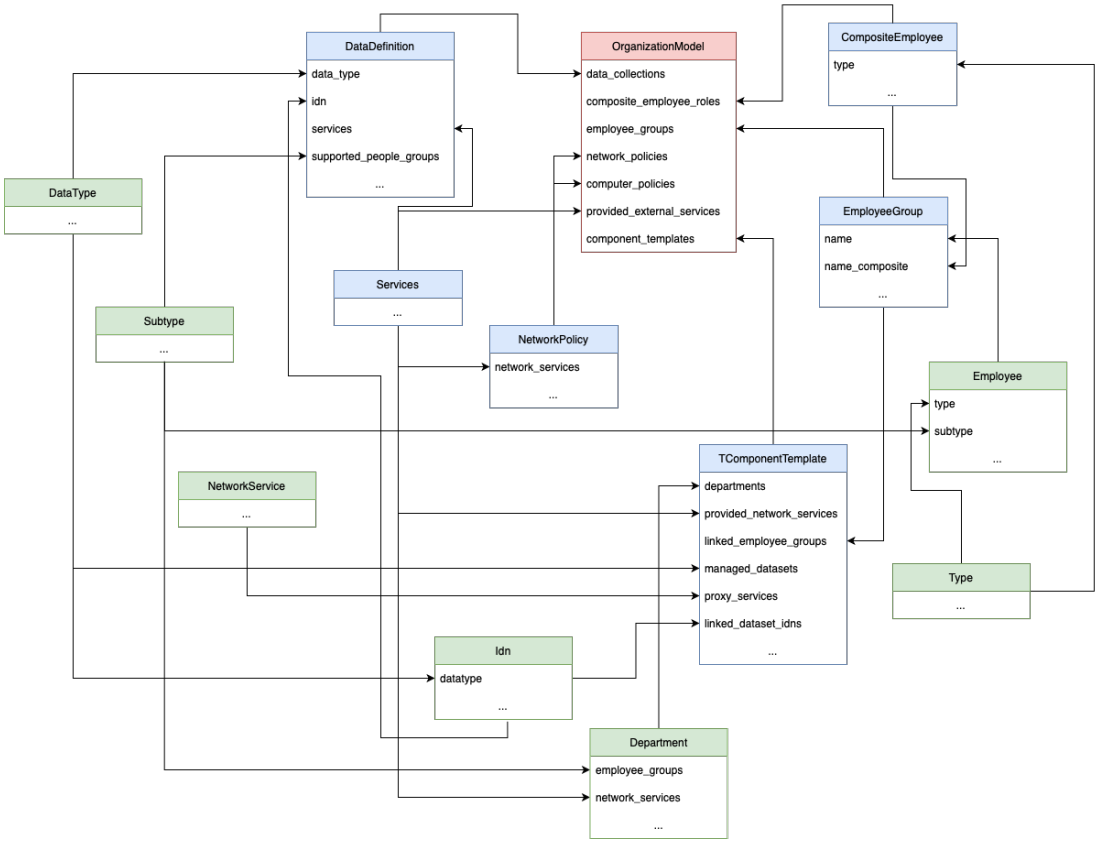


Slika 4.3: Slanje pristupnog tokena za dobivanje informacija o korisniku

4.2. Upravljanje ulazima za izgradnju modela IT sustava

Prvi dio arhitekture sustava koncentrira se na izgradnju ulaza, na temelju kojih se gradi ulazni JSON kojeg će se koristiti u ostalim komponentama u sustavu. Kako bi se korisnicima omogućilo jednostavnije kreiranje i rukovanje ulazima te mogućnost ponovne upotrebe pojedinih ulaza, u izgradnji, primijenjen je princip modularnosti. Modularnost nudi prednosti kao što je enkapsulacija resursa, svaki ulaz poseban je modul. Enkapsulira svoje podatke i ponašanje, komunicirajući s drugim resursima samo putem svog jedinstvenog identifikatora. Također, moduli su relativno neovisni i mogu se zasebno razvijati, testirati i razumjeti. Promjene u jednom modulu, poput unutarnje strukture ulaza, ne moraju nužno utjecati na druge. Dok god su jedinstveni identifikator i krajnja točka nepromijenjeni, neovisnost omogućuje jednostavniju izmjenu pojedinačnog modula bez utjecaja na cijeli sustav.

Za potrebe izgradnje organizacijskog modela unutar relacijske sheme, kreiran je veći broj klasa, čija se kompleksnost može vidjeti iz ERD dijagrama na slici 4.4 .



Slika 4.4: ERD dijagram organizacijskog modela

Ulazni JSON može sadržavati atribute: data collections, composite emplyoee roles, employee groups, network policies, computer policies, provided external services i component templates. Svaki od navedenih atributa predstavljen je jednim poljem modela Organization Model koje odgovara istoimenoj klasi. Predstavljanje svakog atributa klasom omogućava iskorištavanje funkcionalnosti koje nudi relacijska baze podataka za konstruiranje ulaznog JSON-a i primjenu relacijskih veza između pojedinih dijelova.

```
class OrganizationModel(models.Model):
        name = models.CharField(max_length=255)
        data_collections = models.ManyToManyField(DataDefinition)
        composite_employee_roles = models.ManyToManyField(CompositeEmployee,
            blank=True)
        employee_groups = models.ForeignKey(EmployeeGroup, on_delete=models.
            CASCADE)
        network_policies = models.ManyToManyField(NetworkPolicy, blank=True,
            related_name='network_policies')
        computer_policies = models.ManyToManyField(NetworkPolicy, blank=True,
            related_name='computer_policies')
        provided_external_services = models.ManyToManyField(Services, blank=True)
        component_templates = models.ManyToManyField(TComponentTemplate, blank=
            True)
```

Ispis 4.4: Klasa OrganizationModel unutar datoteke models.py

Korisnik svaki od atributa može kreirati, uređivati i brisati bez da utiče na ostale atribute. Također, pojedini atributi mogu biti kompleksni te i sami imati veze prema drugim klasama kao npr. model klase DataDefinition. U modelima je definirana struktura podataka te ograničenja koja se odnose na integritet podataka u bazi podataka.

```
class DataDefinition(models.Model):
        data_type = models.ForeignKey(DataType, on_delete=models.CASCADE)
        database_stored = models.BooleanField(default = True)
        idn = models.ForeignKey(Idn, on_delete=models.CASCADE)
        services = models.ManyToManyField(Services, blank=True)
        ...
```

Ispis 4.5: Klasa DataDefinition unutar datoteke models.py

Kako bi mogli kreirati ili pregledavati ulaze putem API-ja, potrebno je specificirati pogled u kojem se obrađuje HTTP zahtjev koji dolazi s krajnje točke.

```
        path('', ConfigurationList.as_view(), name='configuration_list'),
```

```
path('<int:pk>', ConfigurationDetail.as_view()),
```

Ispis 4.6: Krajnja točka konfiguracije unutar datoteke urls.py

U pogledu za ulaz same konfiguracije, kroz metode, osigurat će se da korisnici mogu vidjeti samo vlastite konfiguracije, a pri kreiranju konfiguracije vlasnik postaje trenutno ulogirani korisnik.

```
class ConfigurationList(ListCreateAPIView):
        permission_classes = [IsAuthenticated]
        serializer_class = ConfigurationSerializer

        def get_queryset(self):
        return Configuration.objects.filter(owner=self.request.user)

        def perform_create(self, serializer):
        serializer.save(owner=self.request.user)
```

Ispis 4.7: Klasa ConfigurationList unutar datoteke views.py

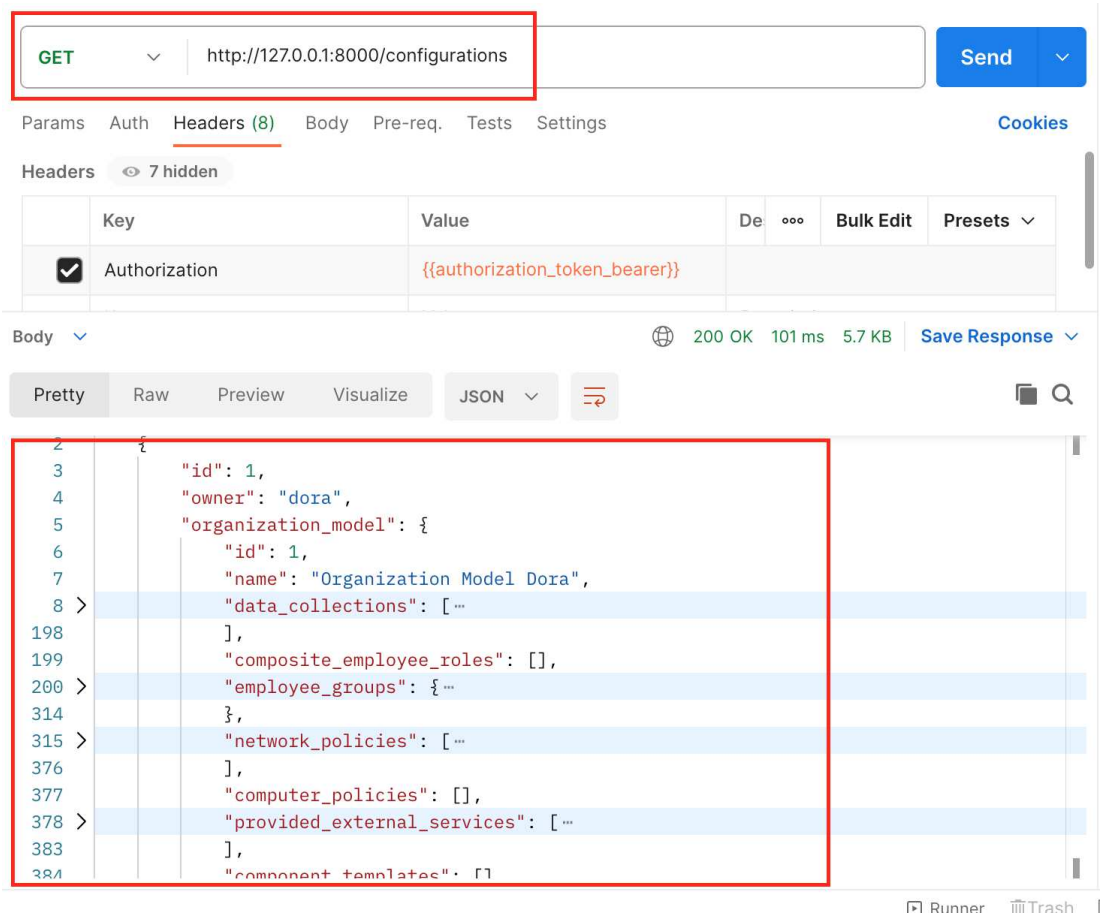
U serijalizatoru koji upravlja pretvorbom podataka, kod svih kompleksnijih modela, definirat ćemo posebna pravila za prikaz putem API-ja metodom GET, kako bi se prikazali cijeli objekti klasa koje su samo atributi u konkretnom modelu.

```
class ConfigurationSerializer(serializers.ModelSerializer):
        ....
        class Meta:
        model = Configuration
        fields = ['id', 'owner', 'organization_model', 'title']

        def to_representation(self, instance):
                rep = super().to_representation(instance)
                rep['organization_model'] = OrganizationModelSerializer(instance.
                    organization_model).data
                return rep
```

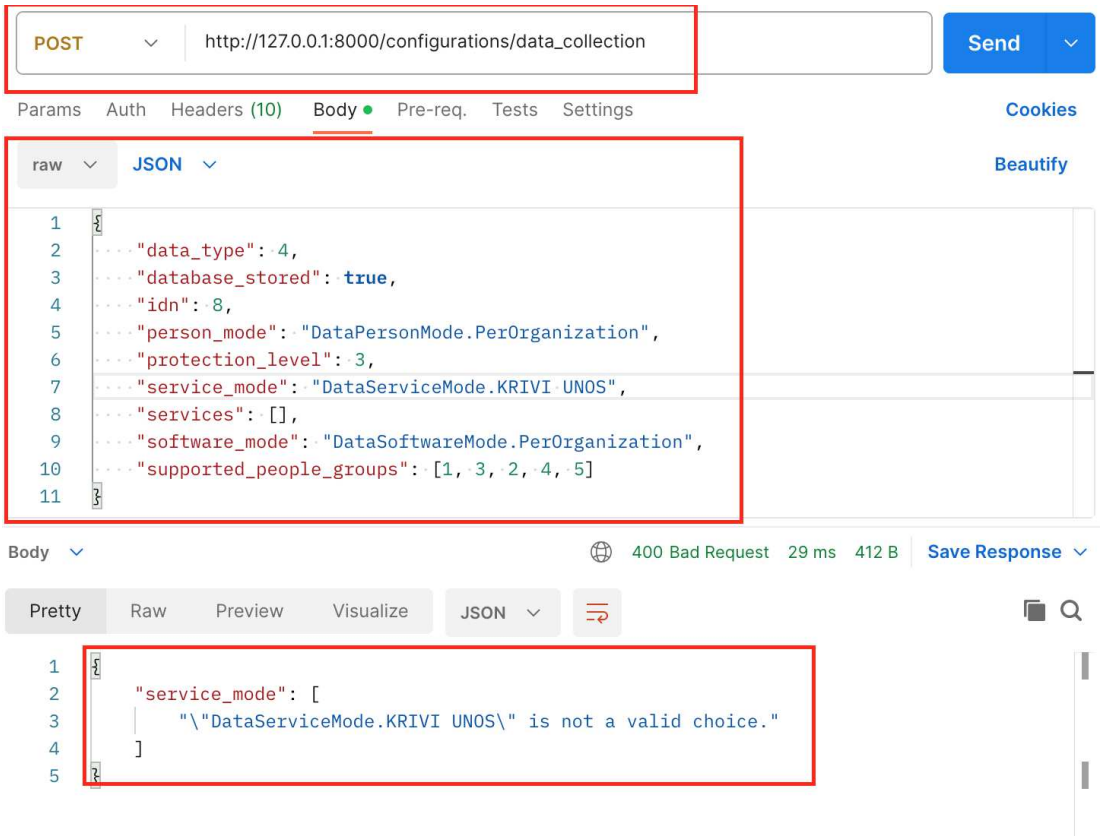
Ispis 4.8: Klasa ConfigurationSerializer unutar datoteke serializers.py

Ovakvo podešavanje, vidljivo u kodu, omogućit će prikaz cijelog organizacijskog modela konfiguracije čiji je vlasnik trenutno ulogirani korisnik, vidljivo na slici 4.5 .



Slika 4.5: Prikaz cjelovitog konfiguracijskog modela

Također, pri kreiranju novih ulaza, kao što je DataDefinition sa slike 4.6, korisnik dobiva informaciju o mjestu i prirodi pogreške u unosu.



Slika 4.6: Prikaz pogreške pri unosu

Kreiranjem ulaza i spajanjem svih željenih atributa u organizacijski model odnosno konfiguracijski model, izgrađen je ulazni JSON koji se koristi za generiranje ostalih komponenti sustava, i koji će biti proslijeđen vanjskom API-ju za izgradnju modela IT sustava.

Slanje konfiguracije na vanjski API definirano je u prikladnom pogledu koji koristi krajnja točka koja prihvaća samo POST metodu i izvršava se nad konfiguracijom koju želimo poslati.

```
path('<int:pk>/send_to_api1_sim/', SendToAPI1View_sim.as_view(), name='send-to-
    api1_sim'),
```

Ispis 4.9: Krajnja točka slanja na API unutar datoteke urls.py

Unutar pogleda izvršava se provjera je li korisnik koji je zatražio slanje onaj koji je i vlasnik konfiguracije koju šalje te se obavlja slanje i prihvaćanje odgovora instanciranjem objekta klase EditedConfiguration u kojoj će u polje odgovora s API-ja biti pohranjen izgrađen model. Izgrađenim model može se uređivati i pregledavati.

```
class SendToAPI1View_sim(APIView):
        ....
        def post(self, request, pk, format=None):
```

```
            if configuration.owner != request.user:
            return Response({'detail': 'Permission denied. Not owner of configuration.
                '}, status=status.HTTP_403_FORBIDDEN)
            ....
            edited_config = EditedConfiguration (
            owner = configuration.owner,
            configuration=configuration,
            title=f'Edited configuration for {configuration.title}',
            api1_response=api1_response
            )
            edited_config.save()
        return Response({'message': 'Data successfully sent to API1.'})
```

Ispis 4.10: Klasa SendToAPI1Viewsim unutar datoteke models.py

Za potrebe pohrane modela, koristit će se polje JSONField, koje će omogućiti pohranu modela u JSON obliku, između ostalog, za potrebe grafičkog uređivanja.

```
class EditedConfiguration(models.Model):
    ...
    api1_response = JSONField(blank=True, null=True)
```

Ispis 4.11: Klasa EditedConfiguration unutar datoteke models.py

4.3. Upravljanje uređenim modelima IT sustava

Za mogućnost upravljanje uređenim modelima IT sustava s grafičkog sučelja te praćenje više verzija uređenih modela, kreirani su posebni pogledi i modeli, koji omogućuju praćenje verzija promijenjenih uređenih modela IT sustava i održavaju poveznicu s originalnim konfiguracijskim modelom na temelju kojeg je izgrađen model IT sustava. Navedeno je korišteno za potrebe organizacije i vizualizacije na klijentskoj strani.

Novi model sada prati vlasnika, originalnu konfiguraciju, model IT sustava, uređeni model IT sustava, promjene koje dolaze s klijentske strane, verzije i putanju za dohvaćanje ZIP konfiguracije koja dolazi s drugog vanjskog API-ja za izvoz koji će biti detaljnije pojašnjen u nastavku.

```
class JSON_EditedConfiguration(models.Model):
        owner = models.ForeignKey(User, on_delete=models.CASCADE) # 1−N veza
        configuration = models.ForeignKey(JSON_Configuration, related_name='
            json_edited_config_id', on_delete=models.CASCADE)
        title = models.CharField(max_length=255)
        version = models.IntegerField(default=1)
        api1_response_json = JSONField(blank=True, null=True)
        api1_response_json_edit = JSONField(blank=True, null=True)
        api1_response_changed = JSONField(blank=True, null=True)
        created_at = models.DateTimeField(auto_now_add=True)
        config_zip = models.FileField(upload_to='json_edited_configurations_files/', null=
            True, blank=True)
```

Ispis 4.12: Klasa JSONEditedConfiguration unutar datoteke models.py

JSON koji dolazi s klijentske strane služi za generiranje grafičkog prikaza komponenti kao što su računala, čvorovi, poveznice i slično. Primjer jednog takvog JSON-a vidljiv je ispod.

```
"api1_response_json_edit": {
        "nodes": [
        {
                "idn": "admin:0:1",
                "name": "admin:0:1",
                "type": "computers",
                "network_idn": 1
        } ..
        ],
```

```
"links": [
{
        "source": "developer:senior:0:1",
        "target": "INTERNET"
} ..
],
"info": [
{
        "idn": "developer:senior:0:1",
        "data": [],
        "supports_external_services": false,
        "used_hardware_quota": 8
}...
}
]
```


Ispis 4.13: Primjer odgovora s API-ja u obliku JSON

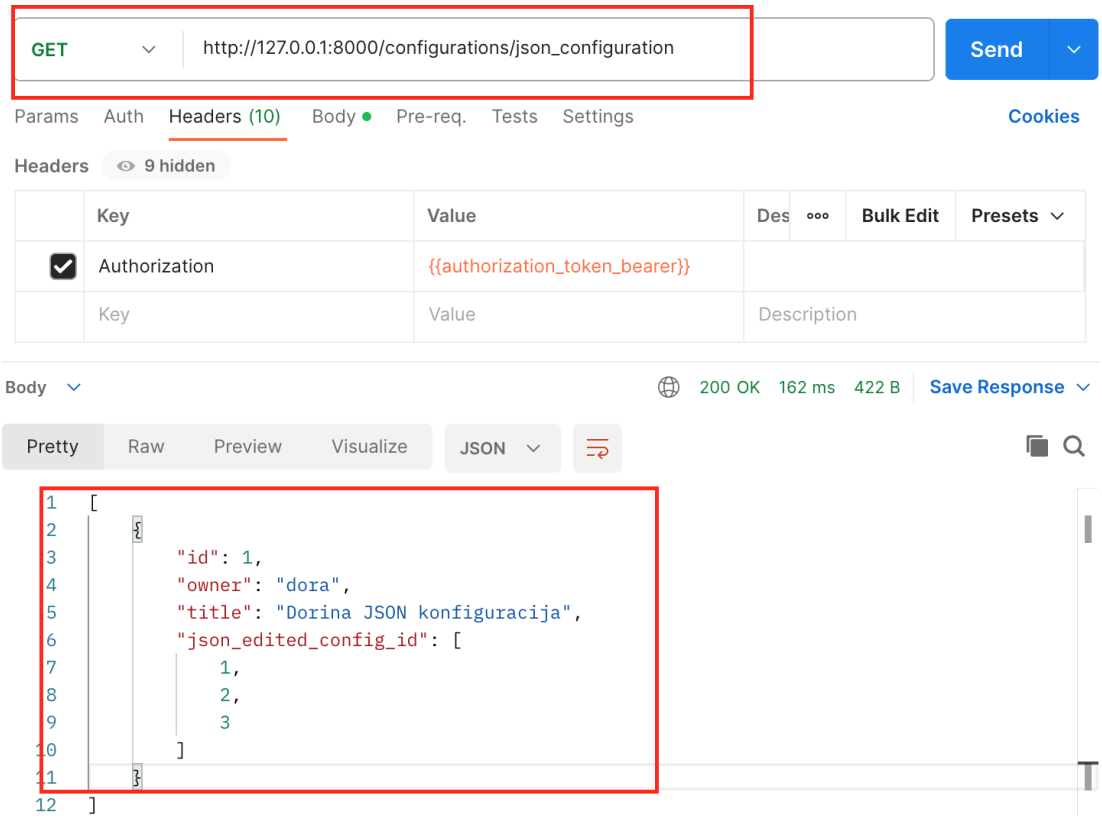
U samom pogledu prate se verzije sustava. Ukratko, slanjem konfiguracije s id=1, kreira se editirana konfiguracija sa svojim jedinstvenim identifikatorom, npr. id=1. Za sustav verzioniranja, navedeno predstavlja stvaranje nove verzije, odnosno na temelju konfiguracije 1 stvara se editirana konfiguracija 1, što ju čini verzijom 1. Kada se na temelju iste konfiguracije s id=1, ponovno kreira editirana konfiguracija, tada će ona biti verzija 2.

```
latest_config = JSON_EditedConfiguration.objects.filter(owner=configuration.owner
    , configuration=configuration).order_by('-version').first()

if latest_config:
new_version = latest_config.version + 1
else:
new_version = 1
```

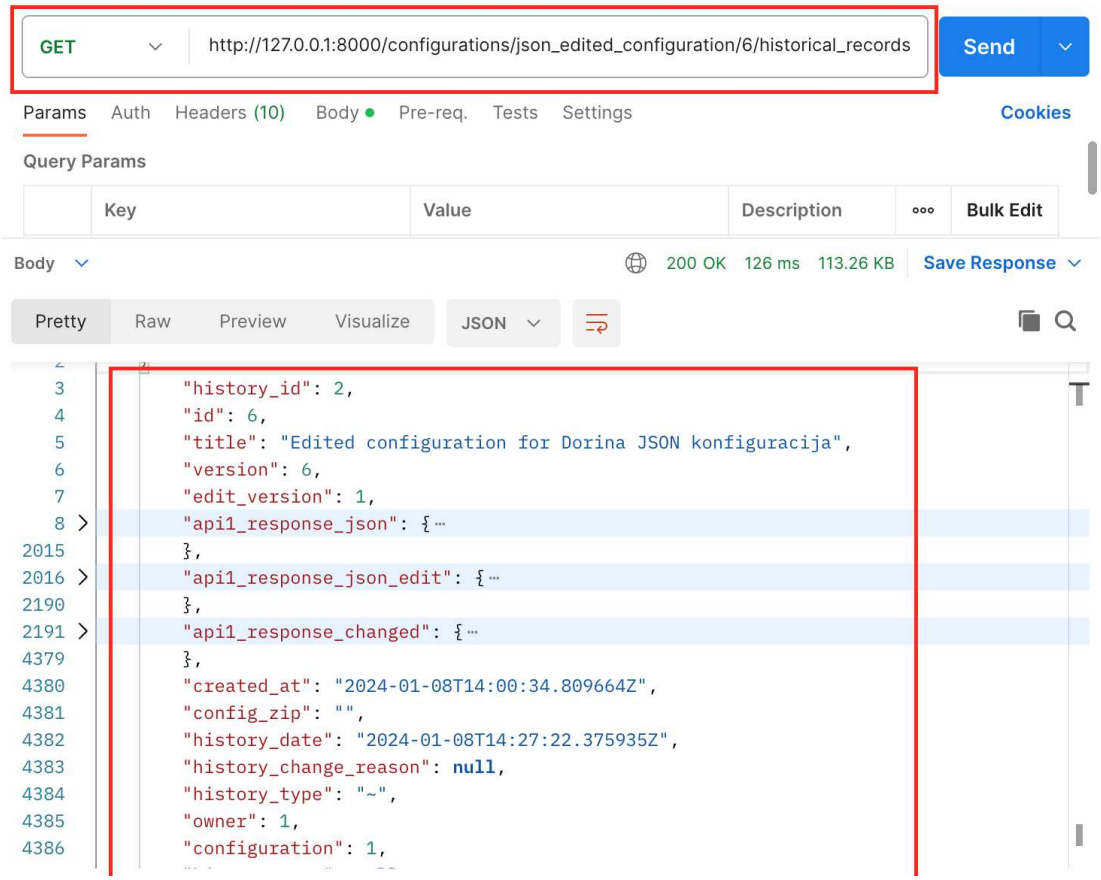
Ispis 4.14: Podešavanje verzija konfiguracija unutar datoteke views.py

Pregledom konfiguracija korisnika, možemo jednostavno vidjeti verzije kojima se putem pojedinačnih putanji pristupa i potencijalno ponovno uređuje, vidljivo na slici 4.7.



Slika 4.7: Prikaz odgovora s verzijama konfiguracija

Za verzioniranje verzija uređenih modela IT sustava, koristi se vanjska biblioteka "simple history" [23], koja omogućava automatsku pohranu svih izmijenjenih zapisa te verzija pojedine instance modela JSON EditedConfiguration, vidljivo na slici 4.8. Kada god se naprave izmjene nad uređenim modelom IT sustava, stvorit će se nova verzija s izmjenom, a stara pohraniti u povijest. Pregled povijesti omogućava korisniku pregled svih promjena te, ako to želi, u zip konfiguracije.



Slika 4.8: Prikaz povijesti verzija uređenih IT modela

Dovoljno je definirati pogled za povijesni pregled i definirati za koje će se modele pohranjivati povijest.

```
class HistoricalRecordView(APIView):
        def get(self, request, pk):
        instance = get_object_or_404(JSON_EditedConfiguration, pk=pk)
        history = instance.history.all()
        serializer = HistoricalRecordSerializer(history, many=True)
        return Response(serializer.data)
```

Ispis 4.15: Klasa HistoricalRecordView unutar datoteke views.py

Konačno, uređeni modeli IT sustava šalju se na vanjski API za izvoz, pri čemu će se poveznica s mjestom gdje je pohranjena zip datoteka pohraniti u polje modela.

Pogled koji omogućava slanje na API za izvoz, prima odgovor i sprema ga u .zip datoteku s imenom i verzijom vodeći pritom računa da korisnik ne može pokrenuti takvu akciju ako i sam nije vlasnik konfiguracije.

```
class JSON_SendToAPI2View_sim(APIView):
        ....
```

```
    def post(self, request, pk, format=None):
        edited_config = get_object_or_404(JSON_EditedConfiguration, pk=pk)

        if edited_config.owner != request.user:
        return Response({'detail': 'Permission denied.'}, status=status.
            HTTP_403_FORBIDDEN)
        ....
        filename = f"{edited_config.title}_{datetime.now().strftime('%Y-%m-%d
            ')}.zip"
        with zipfile.ZipFile(filename, 'w') as myzip:
            myzip.writestr('api2.json', content_str)
            with open(filename, 'rb') as zip_file:
            djangofile = File(zip_file)
            edited_config.config_zip.save(filename, djangofile)
            edited_config.save()

        return Response({'message': 'Configuration successfully generated.'})
```

Ispis 4.16: Klasa JSONSendToAPI2View unutar datoteke models.py

Kako bi korisnik preuzeo konfiguraciju dovoljno je izvršiti POST zahtjev na putanju, pri čemu će se automatski preuzeti .zip datoteka.

```
http://127.0.0.1:8000/configurations/json_edited_configurations/ID/download_config_zip/
```

Sustav ima ugrađenu Swagger i Redoc dokumentaciju, koja olakšava dokumentiranje krajnjih točaka i ostavlja mogućnost isprobavanja krajnjih točaka izravno bez ručnog slanja zahtjeva. Također, rad sadrži kolekciju upita u alatu Postman, kao i dijagrami generirani iz samog sustava koji opisuju veze između entiteta.

5. Diskusija

Implementirani programski sustav za korištenje prototipa generatora modela IT sustava, pruža sustav u kojem je omogućeno upravljanje korisnicima, ulazima za izgradnju modela IT sustava, te uređenim modelima IT sustava i generiranim konfiguracijskim datotekama. Navedene funkcionalnosti ostvarene su putem programskih sučelja korištenjem modularnog pristupa, a sustav je razvijen u skladu s dobrim praksama.

Sustav zadovoljava postavljene funkcionalne zahtjeve, stavljajući posebnu pažnju na sigurnost. Posebna pozornost kod implementacije, obratila se na upravljanje pristupom u cilju osiguranja upotrebljavanja sustava od strane različitih korisnika bez mogućnosti pregleda tuđih podatka. Također, u sustavu su implementirani načini automatskog generiranja dokumentacije krajnjih točaka, da bi se omogućilo olakšano razvijanje klijentske strane sustava.

Najveći izazov pri izradi i implementaciji sustava bilo je osmisliti na koji način definirati ulaze sustava i koja pravila moraju zadovoljavati da bi bili valjani u kontekstu ulaza Generatora. Prije toga, pomno je istraženo funkcioniranje Generatora.

Korištenje odabrane tehnologija i radnog okvira s raznim proširenjima u obliku knjižnica, omogućilo je koncentriranje na izradu ulaza i izlaza sustava radije nego na implementiranje korisničkog sustava iz nule. Naravno, kao i drugi radni okviri, Django nudi mnoge prednosti, ali i ograničenja u smislu predefiniranih načina konfiguracije strukture programskog koda koji ponekad može dodatno kompliciratiimplementaciju ako želimo specifična rješenja.

6. Budući rad

Implementirani programski sustav za korištenje prototipa generatora modela IT sustava otvara mogućnost za mnoga proširenja i unaprjeđenja koja bi dodatno obogatila postojeće funkcionalnosti.

Korištenjem REST arhitekture omogućena je potencijalna integracija s bilo kojom klijentskom tehnologijom, bez potrebe za zadovoljavanjem određenih posebnih zahtjeva. U tom smjeru, krajnjim korisnicima će implementirani sustav biti dostupan kada se kreira prikladna klijentska strana, odnosno korisničko i grafičko sučelje koje podržava rad svih funkcionalnosti implementiranih u trenutnom rješenju.

Kao jedna od mogućnosti unaprjeđivanja implementiranog sustava je proširenje baze podataka na neku od NoSQL baza podataka koja nudi pohranjivanje JSON zapisa uz bolje performanse. Korištenje NoSQL baze podataka omogućilo bi pohranu JSON zapisa, koji se znatno razlikuju po strukturi unutar sustava, na prikladniji način.

Također, boljom organizacijom i definiranjem validacijskih pravila za JSON zapise koji bilježe uređene modele IT sustava, otvorile bi se mnogobrojne mogućnosti vizualizacije na grafičkom sučelju.

Sustav korisnika također se može proširiti tako da korisnici administratori mogu kreirati grupe korisnika iste organizacije, koji bi učinkovito mogli dijeliti razvijene ulaze između određenih skupina korisnika. Ovakva funkcionalnost dodatno bi obogatila postojeći sustav i omogućila da se princip ponovnog korištenja gotovih ulaza koristi još optimalnije unutar istih interesnih grupa.

Integriranje drugih alata i usluga, kao što je autentifikacija u sustav putem servisa Googleova Gmaila i ostali, dodatno bi proširila sustav prijave i registracije te olakšala korištenje sustava.

ZAKLJUČAK

Cilj diplomskog rada bio je implementirati programski sustav za korištenje prototipa generatora modela IT sustava. Implementirani sustav uspješno je zadovoljio postavljene funkcionalne i nefunkcionalne zahtjeve, otvorivši mogućnost izgradnje prikladnog korisničkog sučelja, čime bi se cijeli sustav otvorio prema krajnjim korisnicima i omogućio učinkovito korištenje funkcionalnosti ovog sustava bez obzira na njihovu razinu tehničkog znanja.

U radu se opisuje metodologija razvoja, počevši od ususstavljivanja zahtjeva i arhitekture sustava, odabira tehnologije i baze podataka. U implementacijskom pregledu dan je opis implementiranih funkcionalnosti.

Implementiran je siguran način upravljanja korisnicima, kreiranjem sustava autentifikacije i autorizacije koji prati industrijske standarde korištenjem dvije vrste JWT tokena. Upravljanje ulazima za izgradnju modela IT sustava implementirano je na intuitivan, modularan i jednostavan način, a korisnici mogu kreirati ulaze na temelju kojih se, povezivanjem s drugim vanjskim programskim sučeljima kao API za izgradnju modela IT sustava, kreiraju modeli IT sustava. Omogućeno je upravljanje i uređivanje modela, te je otvorena mogućnost upravljanja uređenim modelima IT sustava putem grafičkog sučelja. Također, sustav podržava i rad s generiranim datotekama dostupnih s drugog vanjskog API-ja za izvoz koje predstavljaju konfiguracije za ciljno simulacijsko okruženje.

Konačno, ishod rada je sustav koji povećava sposobnost širokog spektra korisnika da se uključe i daju svoj doprinos području kibernetičke sigurnosti, što dovodi do razvoja robusnijih i dobro zaokruženih sigurnosnih rješenja za IT sustave.

LITERATURA

1. Laudon, K.C. and Laudon, J.P. (2012) Management Information Systems: Managing the Digital Firm. 13th ed. Pearson. Dostupno na: https://repository.dinus.ac.id/docs/ajar/Ke _Management_Information_Sysrem_13th_Edition_.pdf. Posjećeno: 2.siječnja 2024.

2. Turban, E., Pollard, C. and Wood, G. (2018) Information Technology for Management: On-Demand Strategies for Performance, Growth, and Sustainability. 11th ed. Wiley. Dostupno na: https://www.homeworkforyou.com/static_media/uploadedfiles/1676 Posjećeno: 2.siječnja 2024.

3. Kovačević, I., Groš, S., and Ante-Derek. (2022) 'Automatically Generating Models of IT Systems', IEEE Access. Dostupno na: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnur Posjećeno: 2.siječnja 2024.

4. IBM. What is an Application Programming Interface (API)? Dostupno na: https://www.ibm.com/topics/api#:~text=An%20API%20is%20a%20set,opportunities%20for%20bus Posjećeno: 2.siječnja 2024.

5. Documoto. REST API: Web Services, Scalability, Flexibility, and Security. Dostupno na: https://www.documoto.com/blog/rest-api-web-services-scalability-flexibility-and-security. Posjećeno: 2.siječnja 2024.

6. Baeldung. "Requirements: Functional vs. Non-functional." Baeldung on Computer Science, Baeldung, Dostupno na: https://www.nuclino.com/articles/functional-requirements. Posjećeno: 2.siječnja 2024.

7. AltexSoft. "Nonfunctional Requirements (NFR): Examples, Types, Approaches." AltexSoft, Dostupno na: https://www.altexsoft.com/blog/non-functional-requirements/. Posjećeno: 2.siječnja 2024.

8. Fielding, R.T. (2000) 'Architectural Styles and the Design of Network-based Software Architectures', Doctoral dissertation, University of California, Irvine.

Dostupno na: https://ics.uci.edu/f̃ielding/pubs/dissertation/top.htm. Posjećeno: 2.siječnja 2024.

9. Richardson, L. and Ruby, S. (2007) 'RESTful Web Services', O'Reilly Media, Inc. Dostupno na: https://books.google.hr/books?redir_esc=y&hl=hr&id=XUaErakHsoAC&q=res Posjećeno: 2.siječnja 2024.

10. Fielding, R.T. and Taylor, R.N. (2002) 'Principled Design of the Modern Web Architecture', ACM Transactions on Internet Technology (TOIT), vol. 2, no. 2, pp. 115-150. Dostupno na: https://ics.uci.edu/t̃aylor/documents/2002-REST-TOIT.pdf. Posjećeno: 2.siječnja 2024.

11. Howie, M. 'REST APIs Explained - 4 Components', Mannhowie.com. Dostupno na: https://mannhowie.com/rest-api. Posjećeno: 2.siječnja 2024.

12. Nadin Pethiyagoda, 'REST API Naming Conventions and Best Practices', Medium, Dostupno na: https://medium.com/@nadinCodeHat/rest-api-naming-conventions-and-best-practices-1c4e781eb6a5. Posjećeno: 2.siječnja 2024.

13. Holovaty, A. and Kaplan-Moss, J. (2009) 'The Django Book', Second Edition. Dostupno na: https://www.moreware.org/books/The%20Definitive%20Guide%20to%20Django.p Posjećeno: 2.siječnja 2024.

14. Django Software Foundation (2024) 'Django Documentation, Dostupno na: https://docs.djangoproject.com/en/5.0/. Posjećeno: 2.siječnja 2024.

15. "Django REST framework." Django REST Framework Dostupno na: https://www.django-rest-framework.org.Posjećeno: 2.siječnja 2024.

16. SQLite.org. (2024). SQLite Introduction and JSON1 Dostupno na: https://www.sqlite.org/json1.ht Posjećeno: 2.siječnja 2024.

17. Django Software Foundation. (2021). "Making queries." Django documentation. Dostupno na: https://docs.djangoproject.com/en/3.2/topics/db/queries/. Posjećeno: 2.siječnja 2024.

18. Garcia, R. (2016). JWT tokens and security – working principles and use cases. Vaadata. Dostupno na: https://www.vaadata.com/blog/jwt-tokens-and-security-working-principles-and-use-cases/. Posjećeno: 2.siječnja 2024.

19. Offical Documentation dj-rest-auth. API endpoints, Dostupno na: https://dj-rest-auth.readthedocs.io/en/latest/api_endpoints.html. Posjećeno: 2.siječnja 2024.

20. Django Rest Framework SimpleJWT. Django Rest Framework SimpleJWT Documentation. Dostupno na: https://django-rest-framework-simplejwt.readthedocs.io/en/latest/. Posjećeno: 2.siječnja 2024.

21. Postman. Tools. Dostupno na: https://www.postman.com/product/tools/. Posjećeno: 2.siječnja 2024.

22. Drawio. Tools. Dostupno na: https://www.drawio.com. Posjećeno: 2.siječnja 2024.

23. Django Simple History Documentation. Dostupno na: https://django-simple-history.readthedocs.io/en/latest/. Posjećeno: 2.siječnja 2024.

SAŽETAK

Programski sustav za korištenje prototipa generatora modela IT sustava

Sažetak

Polje kibernetičke sigurnosti često se oslanja na simulirane modele informacijsko-tehnoloških sustava, koji nude kontrolirano okruženje za učenje i razvoj sigurnosnih strategija bez ugrožavanja sigurnosti stvarnih sustava. Trenutno dostupni alati koji omogućavaju izradu simuliranih modela su složeni i zahtijevaju znatnu količinu stručnog znanja i specijaliziranih vještina, čime se ograničava njihova dostupnost i upotreba. Prepoznajući ovaj izazov, cilj ovog rada je razvoj programskog sustava koji omogućava udaljeni pristup jednom takvom postojećem alatu putem programskih sučelja. U radu se opisuje metodologija razvoja i implementacija programskog sustava za korištenje prototipa generatora modela IT sustava kroz aplikacijska programska sučelja korištenjem Django REST arhitekture. Rezultat rada je implementiran sustav koji podržava upravljanje korisnicima, upravljanje ulazima za izgradnju modela IT sustava i upravljanje uređenim modelima IT sustava te konfiguracijskim datotekama. Sustav podržava rad s JSON tipom podataka.

Ključne riječi: Prototip generatora modela IT sustava, Informacijsko tehnološki sustavi, Django Rest Framework, JSON, Aplikacijska programska sučelja

SUMMARY

Program system for using prototype generator of IT system models

Abstract

The field of cyber security often relies on simulated models of information technology systems, which offer a controlled environment for learning and developing security strategies without compromising the security of real systems. Currently available tools that enable the creation of simulated models are complex and require a considerable amount of expertise and specialized skills, thus limiting their availability and use. Recognizing this challenge, the goal of this paper is the development of a software system that enables remote access to such an existing tool through software interfaces. The paper describes the methodology of development and implementation of the program system for using the IT system model generator prototype through application programming interfaces using the Django REST architecture. The result of the work is an implemented system that supports user management, management of inputs for building IT system models and management of edited IT system models and configuration files. The system supports working with JSON data type.

Keywords: IT system model generator prototype, Information technology systems, Django Rest Framework, JSON, Application programming interfaces

SKRAĆENICE

ITS	Information technology system	Informacijsko-tehnološki sustavi
IT	Information technology	Informacijska tehnologija
API	Application Programming Interface	Aplikacijska programska sučelja
JSON	JavaScript Object Notation	JavaScript objektna notacija
REST	REpresentational State Transfer	Prijenos reprezentativnog stanja
URL	Uniform Resource Locator	Internet poveznica
HTTP	Hypertext Transfer Protocol	Protokol prijenosa hiperteksta
DRY	Don't Repeat Yourself	Princip neponavljanja
DRF	Django Rest Framework	Django Rest radni okvir
MVT	Model-View-Template	Model-Pogled-Predložak obrazac
XML	Extensible Markup Language	Jezik za označavanje podataka
ORM	Object-Relational Mapper	Objektno-relacijsko preslikavanje
SQL	Structured Query Language	Strukturirani upitni jezik
JWT	JSON Web Token	JSON internet token
ERD	Entity Relationship Diagram	Dijagram entiteta i veza